Lecture 9 - Conjugate Priors & The Naive Bayes Classifier

Review of Gaussian Distribution

(Read section 2.3 "The Gaussian Distribution" from the Bishop textbook)

Univariate Gaussian

The Gaussian distribution is a widely used probabilistic model for the probability density function (pdf) of continuous random variables.

The Gaussian distribution can model both univariate (1-D) or multivariate (multi-dimensional) samples.

In the **univariate** case, the pdf of a Gaussian distribution for a random variable $X \in \mathbb{R}$ can be written as

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

In this case, we say that X follows a Gaussian distribution with mean μ and variance σ^2, or, $X \sim N(\mu, \sigma^2)$.

- We can define the **precision** parameter β as the inverse of the variance, that is, $\beta = \frac{1}{\sigma^2}$.

- A Gaussian distribution is called **Normal** when the mean is $\mu = 0$ and variance is $\sigma^2 = 1$, $X \sim N(0, 1)$.

In [1]:

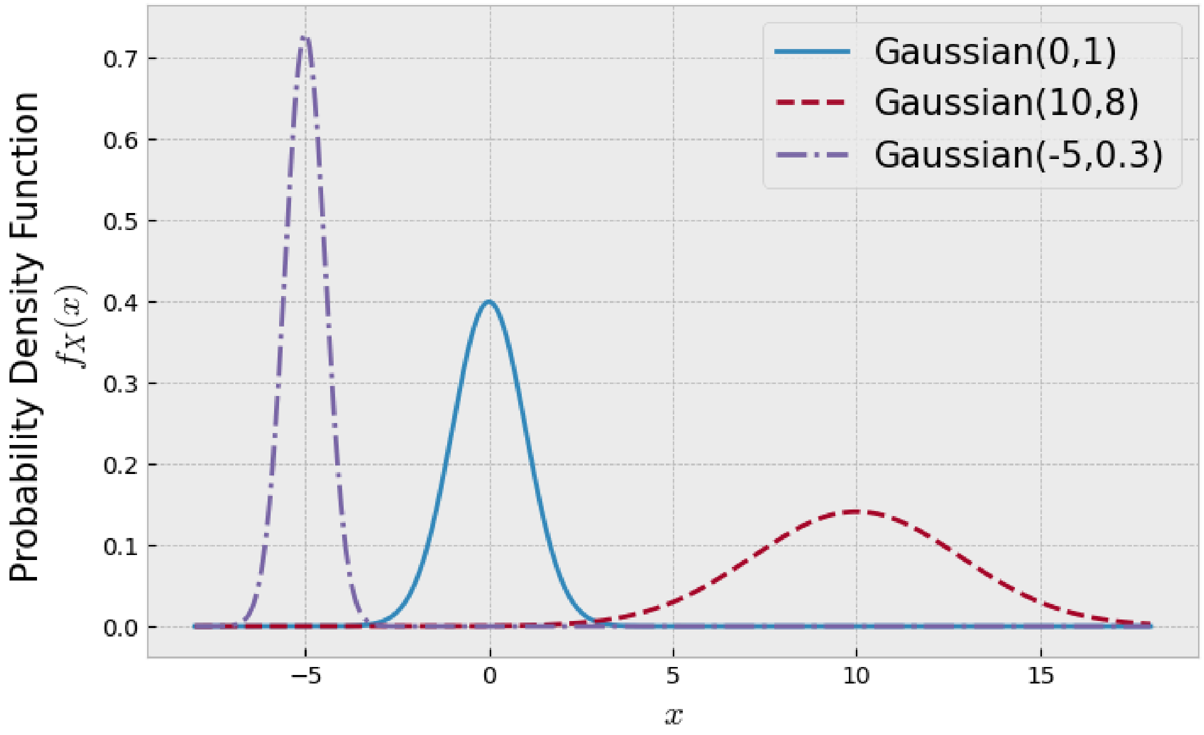
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('bmh')
import scipy.stats as stats
```

In [2]:

```
G1=stats.norm(0,1) # G(mean=0, variance=1^2)

G2=stats.norm(10,np.sqrt(8)) #G(mean=10, variance=8)

G3=stats.norm(-5,np.sqrt(0.3)) #G(mean=-5, variance=0.3^2)
```

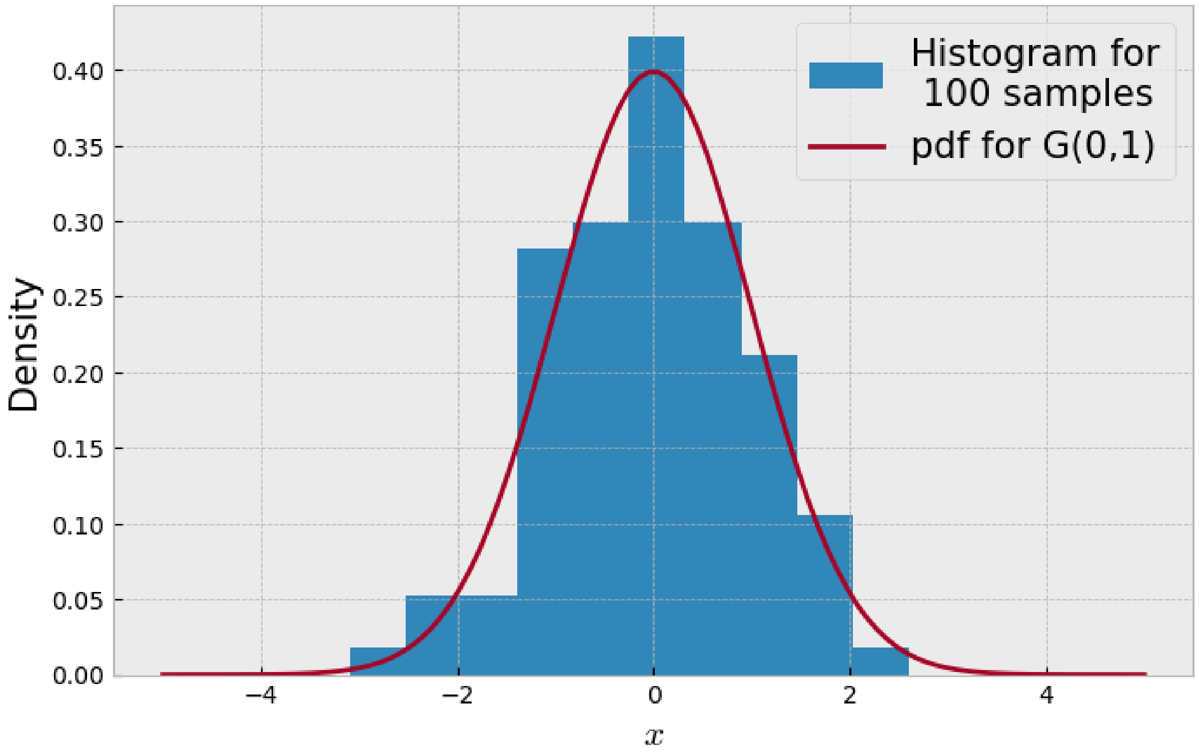
In [3]:

```
plt.figure(figsize=(8,5))

x=np.linspace(-8,18,1000)
plt.plot(x,G1.pdf(x),label='Gaussian(0,1)')
plt.plot(x,G2.pdf(x),'--',label='Gaussian(10,8)')
plt.plot(x,G3.pdf(x),'-.',label='Gaussian(-5,0.3)')
plt.legend(fontsize=15)
plt.xlabel('$x$',size=15)
plt.ylabel('Probability Density Function \n $f_X(x)$',size=15);
```



In [4]:

```
samples=G1.rvs(size=100)

plt.figure(figsize=(8,5))
x=np.linspace(-5,5,100)
plt.hist(samples,density=True, label='Histogram for\n 100 samples')
plt.plot(x, G1.pdf(x), label='pdf for G(0,1)')
plt.legend(fontsize=15)
plt.xlabel('$x$',size=15)
plt.ylabel('Density',size=15);
```



Multivariate Gaussian

In the **multivariate** case, the pdf of a Gaussian distribution for a random variable $X \in \mathbb{R}^D$ can be written as

$$f_X(x) = \frac{1}{\sqrt{(2\pi)^d \, |\Sigma|}} \exp\left(-\frac{1}{2} (\mathbf{x} - \mu)^T \Sigma^{-1} (\mathbf{x} - \mu) \right)$$

In this case, we say that X follows a Gaussian distribution with mean μ and covariance Σ, or, $X \sim N(\mu, \Sigma)$.

- μ is a D-dimensional mean vector
- Σ is a $D \times D$ covariance matrix
- $|\Sigma|$ denotes the determinant of Σ
- The precision parameter in a d-dimensional space is equal to $\beta = \Sigma^{-1}$

Let $X = [X_1, X_2]$. The **covariance** Σ measures the amount of variance is each individual dimension, X_1 and X_2, as well as the amount of covariance between the two. We can write the covariance as

$$\begin{aligned}
\operatorname{cov}(X_1, X_2) &= E\big[(X_1 - E[X_1])\,(X_2 - E[X_2])\big] \\
&= \begin{bmatrix} \operatorname{var}(X_1) & \operatorname{cov}(X_1, X_2) \\ \operatorname{cov}(X_1, X_2) & \operatorname{var}(X_2) \end{bmatrix} \\
&= \begin{bmatrix} \sigma_{X_1}^2 & \operatorname{cov}(X_1, X_2) \\ \operatorname{cov}(X_1, X_2) & \sigma_{X_2}^2 \end{bmatrix}
\end{aligned}$$

- The **Pearson's correlation coefficient** between random variables X_1 and X_2 is defined as:

$$r = \frac{\text{cov}(X_1, X_2)}{\sqrt{\text{var}(X_1)}\sqrt{\text{var}(X_2)}} = \frac{\text{cov}(X_1, X_2)}{\sigma_{X_1}\sigma_{X_2}}$$

In [5]:

```
mu = np.array([0,0]) # mean vector
cov  = np.array([[2,0],[0,2]]) # covariance matrix
print(cov)
G = stats.multivariate_normal(mu, cov)
G
```

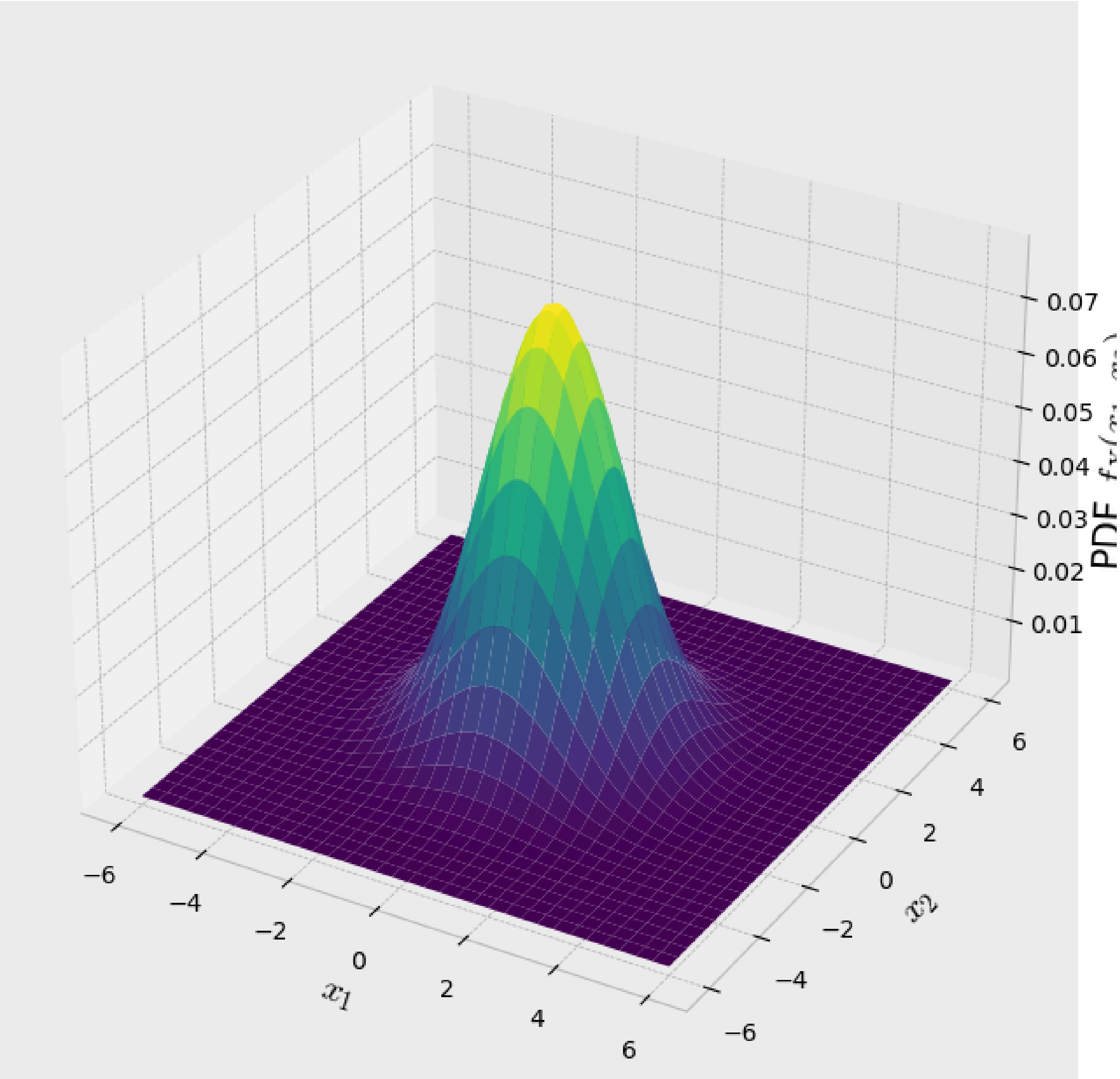
```
[[2 0]
 [0 2]]
```

Out[5]:

```
<scipy.stats._multivariate.multivariate_normal_frozen at 0x12e7afb10>
```

In [6]:

```
x, y = np.mgrid[-6:6:100j, -6:6:100j]
xy = np.column_stack([x.flat, y.flat])
z = stats.multivariate_normal.pdf(xy, mean=mu, cov=cov)
z = z.reshape(x.shape)

fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(x,y,z, rstride=3, cstride=3, linewidth=1, antialiased=True,
                cmap=plt.cm.viridis)
ax.set_xlabel('$x_1$',size=15)
ax.set_ylabel('$x_2$',size=15)
ax.set_zlabel('PDF $f_X(x_1,x_2)$',size=15);
```



Let's use the Mathematica's demonstration "Joint Density of Bivariate Gaussian Random Variables" to better understand the role of the covariance matrix.

Example: Multivariate Gaussian-Gaussian Conjugate Prior Relationship

For a D-dimensional Gaussian data likelihood with mean μ and covariance $\beta\mathbf{I}$ and a prior distribution with mean μ_0 and covariance Σ_0

$$\begin{aligned} P(\mathbf{t}|\mathbf{w}) &\sim \mathcal{N}(\mu, \beta\mathbf{I}) \\ P(\mathbf{w}) &\sim \mathcal{N}(\mu_0, \Sigma_0) \end{aligned}$$

The posterior distribution

$$P(\mathbf{w}|\mathbf{t}) \sim \mathcal{N}\left(\mu_N, \Sigma_N\right)$$

$$\mu_N = \Sigma_N \left(\Sigma_0^{-1}\mu_0 + \beta\mathbf{X}^T\mathbf{t}\right)$$

$$\Sigma_N^{-1} = \Sigma_0^{-1} + \beta\mathbf{X}^T\mathbf{X}$$

where $\mathbf{X}$ is the feature matrix of size $N \times M$.

- What happens with different values of β and Σ_0?

To simplify, let's assume the covariance of the prior to be **isotropic**, that is, it is a diagonal matrix with the same value along the diagonal, $\Sigma_0 = \alpha^{-1}\mathbf{I}$. And, let $\mu_0 = [0, 0]$, thus

$$\mu_N = \beta\Sigma_N\mathbf{X}^T\mathbf{t}$$

and

$$\Sigma_N^{-1} = \alpha\mathbf{I} + \beta\mathbf{X}^T\mathbf{X}$$

Let's consider the example presented in the Bishop textbook (Figure 3.7 in page 155).

Consider a single input variable $\mathbf{x}$, a single target variable $\mathbf{t}$ and a linear model of the form $y(\mathbf{x}, \mathbf{w}) = w_0 + w_1\mathbf{x}$. Because this has just two parameters coefficients, $w = [w_0, w_1]^T$, we can plot the prior and posterior distributions directly in parameter space (2-dimensional parameter space).

Let's generate some synthetic data from the function $f(x, a) = w_0 + w_1 x$ with parameter values $w_0 = -0.3$ and $w_1 = 0.5$ by first choosing values of x_n from the uniform distribution $U(x_n| - 1, 1)$, then evaluating $f(x_n, \mathbf{w})$, and finally adding Gaussian noise with standard deviation of $\sigma = 0.2$ to obtain the target values t_n.

$$t_n = f(x_n, \mathbf{w}) + \epsilon = -0.3 + 0.5x_n + \epsilon$$

where $\epsilon \sim \mathcal{N}(0, \beta\mathbf{I})$.

- **Our goal is to recover the values of w_0 and w_1 from such data, and we will explore the dependence on the size of the data set.**

For some data, $\{x_n, t_n\}_{n=1}^N$, we can pose this problem in terms of **Regularized Least Squares**:

$$
\begin{aligned}
J(\mathbf{w}) &= \frac{1}{2}\sum_{n=1}^{N}(t_n - y_n)^2 + \frac{\lambda}{2}\sum_{i=0}^{1} w_i^2 \\
&= \frac{1}{N}\sum_{n=1}^{2}(t_n - y_n)^2 + \frac{\lambda}{2}\left(w_0^2 + w_1^2\right) \\
&\Rightarrow \arg_{\mathbf{w}} \min J(\mathbf{w})
\end{aligned}
$$

- Using **MAP**, we can rewrite our objective using the **Bayesian interpretation**:

$$\arg_{\mathbf{w}} \max P(\epsilon|\mathbf{w})P(\mathbf{w})$$

Let's consider the data likelihood, $P(\epsilon|\mathbf{w})$, to be a Gaussian distribution with mean $\mu = 0$ and variance $\sigma^2 = \beta\mathbf{I}$. And let's also consider the prior distribution, $P(\mathbf{w})$, to be a Gaussian distribution with mean μ_0 and variance $\sigma_0^2 = \alpha^{-1}\mathbf{I}$. Then, using the derivations from above, we can rewrite our optimization as:

$$
\begin{aligned}
&\arg_{\mathbf{w}} \max \mathcal{N}(\epsilon|0, \beta\mathbf{I})\mathcal{N}(\mathbf{w}|\boldsymbol{\mu_0}, \alpha^{-1}\mathbf{I}) \\
\propto\ &\arg_{\mathbf{w}} \max \mathcal{N}\left(\beta\Sigma_N^{-1}\mathbf{X}^T\mathbf{t}, \Sigma_N\right)
\end{aligned}
$$

where $\mu_0 = [0, 0]$, $\mathbf{X}$ is the polynomial feature matrix, and $\Sigma_N = \left(\alpha^{-1}\mathbf{I} + \beta\mathbf{X}^T\mathbf{X}\right)^{-1}$ is the covariance matrix of the posterior distribution.

Note that we **do not known** the parameters of the prior distribution (μ_0 and σ_0 are unknown). The parameters of the prior distribution will have to be chosen by the user. And they will essentially *encode* any behavior or a priori knowledge we may have about the weights.

- **Both our data likelihood and prior distributions are in a 2-dimensional space (this is because our *model order* is $M = 2$ -- we have 2 parameters!).**

We are going to generate data from $t = -0.3 + 0.5x + \epsilon$ where ϵ is drawn from a zero-mean Gaussin distribution.

- **The goal is to estimate the values $w_0 = -0.3$ and $w_1 = 0.5$**
- The feature matrix $\mathbf{X}$ can be computed using the polynomial basis functions
- **Parameters to choose:** β and α

We want to implement this scenario for a case that we are getting more data every minute. As we get more and more data, we want to **update our prior distribution using our posterior distribution (informative prior)**, because they take the have the same distribution form. This is only possible because because Gaussian-Gaussian have a conjugate prior relationship. That is, the posterior distribution is also a Gaussian.

In [7]:

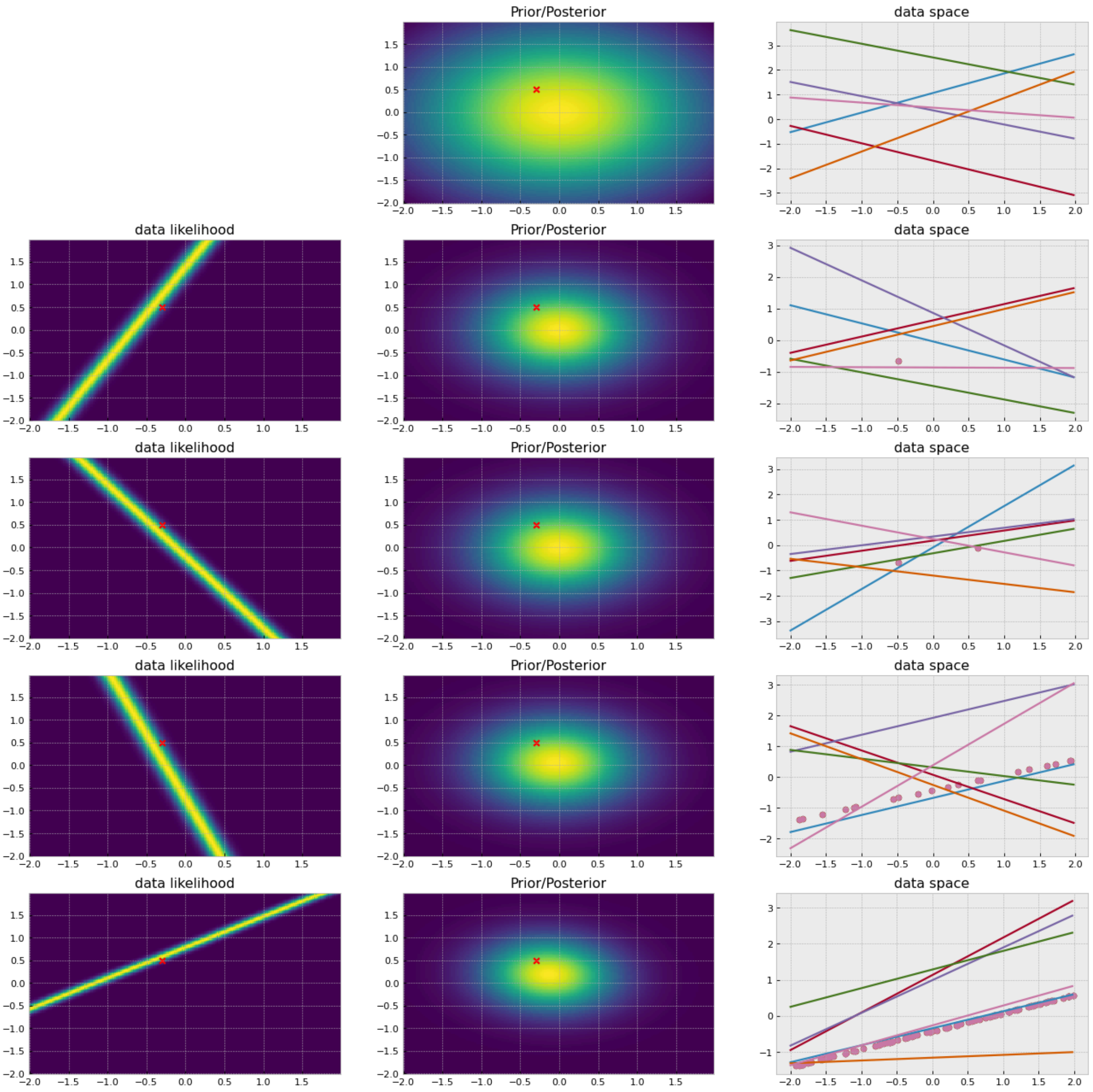
```
from scipy.stats import multivariate_normal
import textwrap
```

```python
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline

# Fixing the random seed generator
np.random.seed(0)

def likelihood_prior_func(beta = 2, alpha = 1, draw_num=(0,1,10,20,50,100)):
    '''Online Update of the Posterior distribution for a Gaussian-Gaussian c
    Parameter:
    beta - variance of the data likelihood (of the additive noise)
    alpha - precision value or 1/variance of the prior distribution
    draw_num - number of points collected at each instance.

    This function will update the prior distribution as new data points are
    The prior distribution will be the posterior distribution from the last

    fig = plt.figure(figsize=(20, 20), dpi= 80, facecolor='w', edgecolor='k'

    # true (unknown) weights
    a = -0.3 # w0
    b = 0.5  # w1

    # set up input space
    rangeX = [-2, 2] # range of values for the input
    step = 0.025 # distance between points
    X = np.mgrid[rangeX[0]:rangeX[1]:step] # creates a grid of values for ir

    #initialize prior/posterior and sample data
    S0 = (1/alpha)*np.eye(2) # prior covariance matrix
    sigma = S0 # copying it so we can update it later
    mean = [0,0] # mean for prior

    # Draws samples from Uniform(-1,1) distribution
    draws = np.random.uniform(rangeX[0],rangeX[1],size=draw_num[-1])
    # Generate the noisy target samples
    T = a + b*draws + np.random.normal(loc=0, scale=np.sqrt(beta))

    for i in range(len(draw_num)):
        if draw_num[i]>0: #skip first image

            # INPUT DATA
            #Feature Matrix (Polynomial features with M=2)
            FeatureMatrix = np.array([draws[:draw_num[i]]**m for m in range(
            #Target Values
            t = T[0:draw_num[i]]

            # POSTERIOR PROBABILITY
            # Covariance matrix
            sigma = np.linalg.inv(S0 + beta*FeatureMatrix.T@FeatureMatrix)
            # Mean vector
            mean = beta*sigma@FeatureMatrix.T@t

            # PARAMETER SPACE
            # create a meshgrid of possible values for w's
            w0, w1 = np.mgrid[rangeX[0]:rangeX[1]:step, rangeX[0]:rangeX[1]:
```

```
            # Define the Gaussian distribution for data likelihood
            p = multivariate_normal(mean=t[draw_num[i]-1], cov=beta)
            # Initialize the PDF for data likelihood
            out = np.empty(w0.shape)
            # For each value (w0,w1), compute the PDF for all data samples
            for j in range(len(w0)):
                out[j] = p.pdf(w0[j]+w1[j]*draws[draw_num[i]-1])

            # Plot the data likelihood
            ax = fig.add_subplot(*[len(draw_num),3,(i)*3+1])
            ax.pcolor(w0, w1, out)
            # Add the current value for parameters w=(w0,w1)
            ax.scatter(a,b, c='r',marker='x')
            myTitle = 'data likelihood'
            ax.set_title("\n".join(textwrap.wrap(myTitle, 100)))

        # PARAMETER SPACE
        # create a meshgrid of possible values for w's
        w0, w1 = np.mgrid[rangeX[0]:rangeX[1]:step, rangeX[0]:rangeX[1]:step

        # POSTERIOR PROBABILITY
        # initialize the matrix with posterior PDF values
        pos = np.empty(w1.shape + (2,))
        # for w0
        pos[:, :, 0] = w0
        # and for w1
        pos[:, :, 1] = w1
        # compute the PDF
        p = multivariate_normal(mean=mean, cov=sigma)

        #Show prior/posterior
        ax = fig.add_subplot(*[len(draw_num),3,(i)*3+2])
        ax.pcolor(w0, w1, p.pdf(pos))
        # Add the value for parameters w=(w0,w1) that MAXIMIZE THE POSTERIOR
        ax.scatter(a,b, c='r',marker='x')
        myTitle = 'Prior/Posterior'
        ax.set_title("\n".join(textwrap.wrap(myTitle, 100)))

        # DATA SPACE
        ax = fig.add_subplot(*[len(draw_num),3,(i)*3+3])
        for j in range(6):
            # draw sample from the prior probability to generate possible va
            w0, w1 = np.random.multivariate_normal(mean=mean, cov=sigma)
            # Estimated labels
            t = w0 + w1*X
            # Show data space
            ax.plot(X,t)
            if draw_num[i] > 0:
                ax.scatter(FeatureMatrix[:,1], T[0:draw_num[i]])
            myTitle = 'data space'
            ax.set_title("\n".join(textwrap.wrap(myTitle, 100)))
```

In [21]:

```
# Parameters
# beta - variance of the data likelihood (for the additive noise)
# alpha - precision value or 1/variance for the prior distribution
# draw_num - number of points collected at each instance

likelihood_prior_func(beta = 0.01, alpha = 0.5, draw_num=(0,1,2,20,100))
```



Introduction to Classification & Probabilistic Generative Models

So far we have focused on regression. We will begin to discuss **classification**.

Suppose we have training data from two classes, C_1 and C_2, and we would like to train a classifier to assign a label to incoming test points as C_1 or C_2.

There are *many* classifiers in the machine learning literature. We will cover a few in this course. Today we will focus on probabilistic generative approaches for classification.

There are two types of classification algorithms: **discriminative** or **generative**.

- A **discriminative** approach for classification is one in which we partition the feature space into regions for each class. Then, when we have a test point, we evaluate in which region it landed on and classify it accordingly.
- A **generative** approach for classification is one in which we estimate the parameters for distributions that generate the data for each class. Then, when we have a test point, we can compute the posterior probability of that point belonging to each class and assign the point to the class with the highest posterior probability.

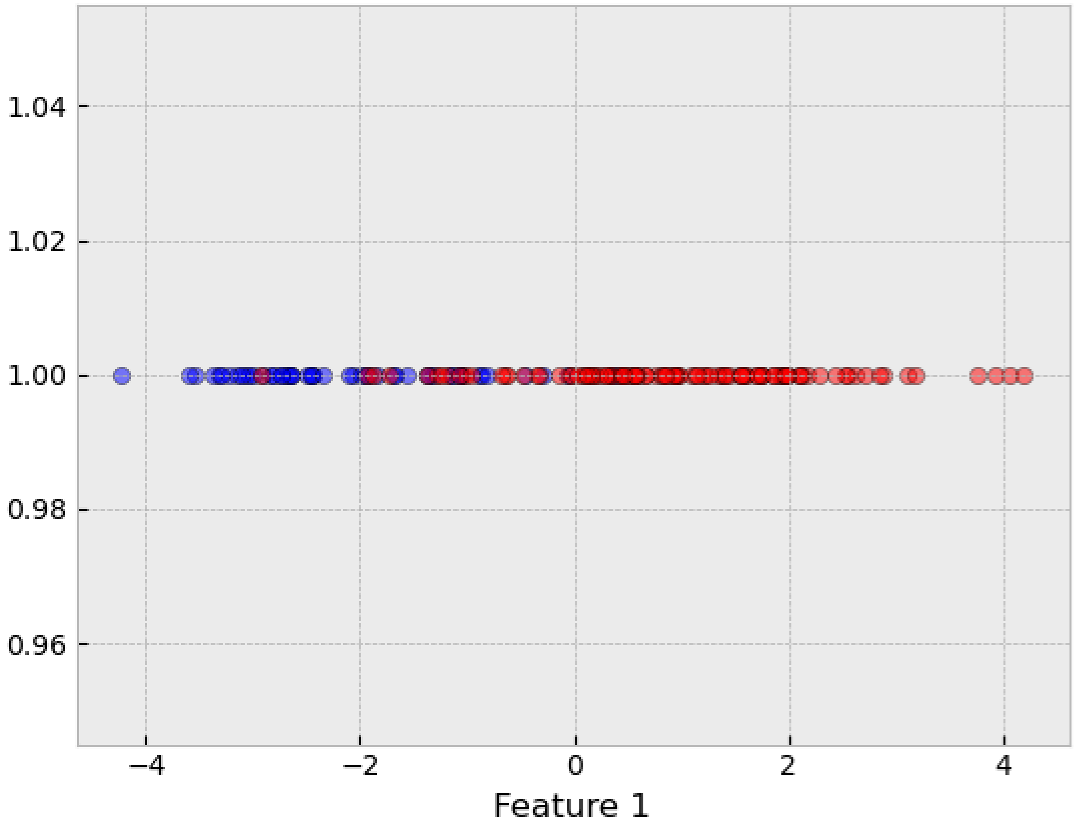
In [9]:

```python
def generateData(mean1, mean2, cov1, cov2, N1, N2):
    # We are generating data from two Gaussians to represent two classes
    # In practice, we would not do this - we would just have data from the p
    data_C1 = stats.multivariate_normal(mean1, cov1).rvs(size=N1)
    data_C2 = stats.multivariate_normal(mean2, cov2).rvs(size=N2)

    # Entire Training Dataset
    data = np.concatenate((data_C1, data_C2))
    labels = np.concatenate((np.ones(N1),2*np.ones(N2)))

    if len(mean1)>1:
        plt.scatter(data[labels==1,0], data[labels==1,1], c='b', alpha=0.5,
        plt.scatter(data[labels==2,0], data[labels==2,1], c='r', alpha=0.5,
        plt.xlabel('Feature 1'); plt.ylabel('Feature 2');
    else:
        plt.scatter(data[labels==1], np.ones(N1), c='b', alpha=0.5, edgecolo
        plt.scatter(data[labels==2], np.ones(N2), c='r', alpha=0.5, edgecolo
        plt.xlabel('Feature 1');

    return data, labels
```

Case 1: Univariate Data Likelihood

In [10]:

```python
mean1 = [-2]
mean2 = [1]
var1 = [1]
var2 = [2]
N1 = 50 # C1 - blue
N2 = 100 # C2 - red

data, labels = generateData(mean1, mean2, var1, var2, N1, N2)
```



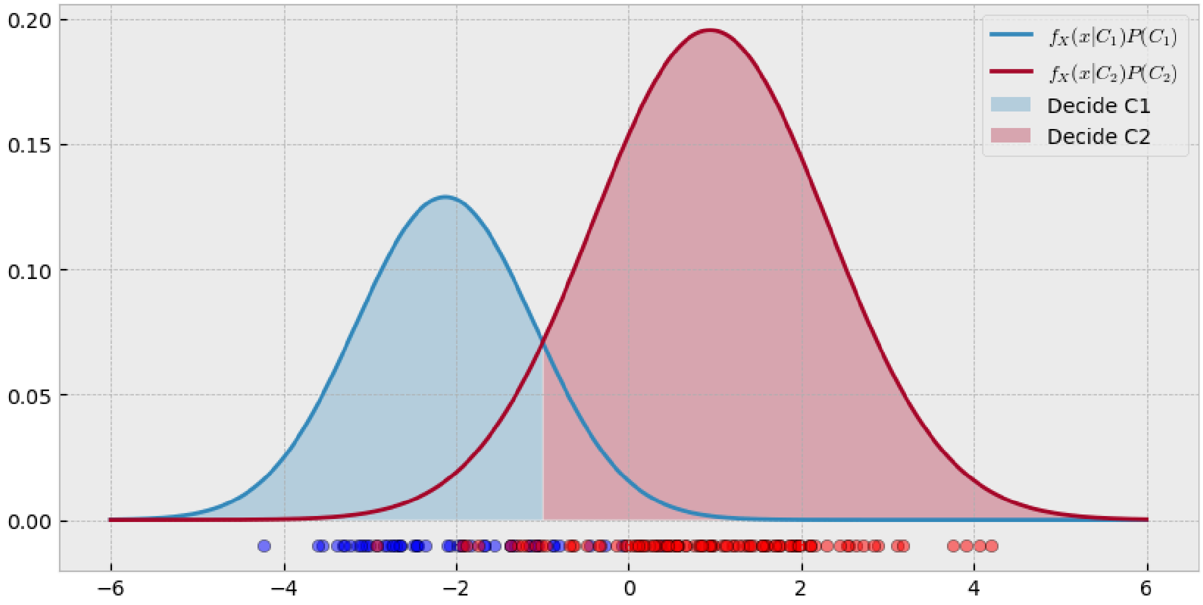
In [11]:

```python
def drawMAP(data, labels):

    #### Estimate parameters (MLE solution)
    # Means
    mu1 = np.mean(data[labels==1])
    mu2 = np.mean(data[labels==2])

    # Variances
    var1 = np.cov(data[labels==1])
    var2 = np.cov(data[labels==2])

    #### Estimate Prior Probabilities - relative frequency
    N = len(data)
    N1 = np.sum(labels==1)
    N2 = N - N1
    p1 = N1/N # prior probability for C1
    p2 = N2/N # prior probability for C2

    #### Define data likelihoods
    G1=stats.norm(loc=mu1,scale=np.sqrt(var1)) # P(x|C1)
    G2=stats.norm(loc=mu2,scale=np.sqrt(var2)) # P(x|C2)
    x=np.linspace(-6,6,1001)

    #### Plot the weighted densities
    # these are proportional to the posteriors
    plt.figure(figsize=(10,5))
    plt.plot(x,p1*G1.pdf(x),label='$f_X(x|C_1)P(C_1)$')
    plt.plot(x,p2*G2.pdf(x),label='$f_X(x|C_2)P(C_2)$')

    #### Determine the regions where the posterior for deciding C1
    # and the posterior for deciding C2
    R1=x[np.where(p1*G1.pdf(x)>= p2*G2.pdf(x))]
    R2=x[np.where(p1*G1.pdf(x)< p2*G2.pdf(x))]

    # Fill under the regions found above
    plt.fill_between(R1,p1*G1.pdf(R1),alpha=0.3,label='Decide C1')
    plt.fill_between(R2,p2*G2.pdf(R2),alpha=0.3,label='Decide C2')
    plt.scatter(data[labels==1], -0.01*np.ones(N1), c='b', alpha=0.5, edgeco
    plt.scatter(data[labels==2], -0.01*np.ones(N2), c='r', alpha=0.5, edgeco
    plt.legend()

    # Print the MAP threshold
    print('MAP decision threshold to decide C2 is >',round(R2[0],2))
```

In [12]:

```
drawMAP(data,labels)
```

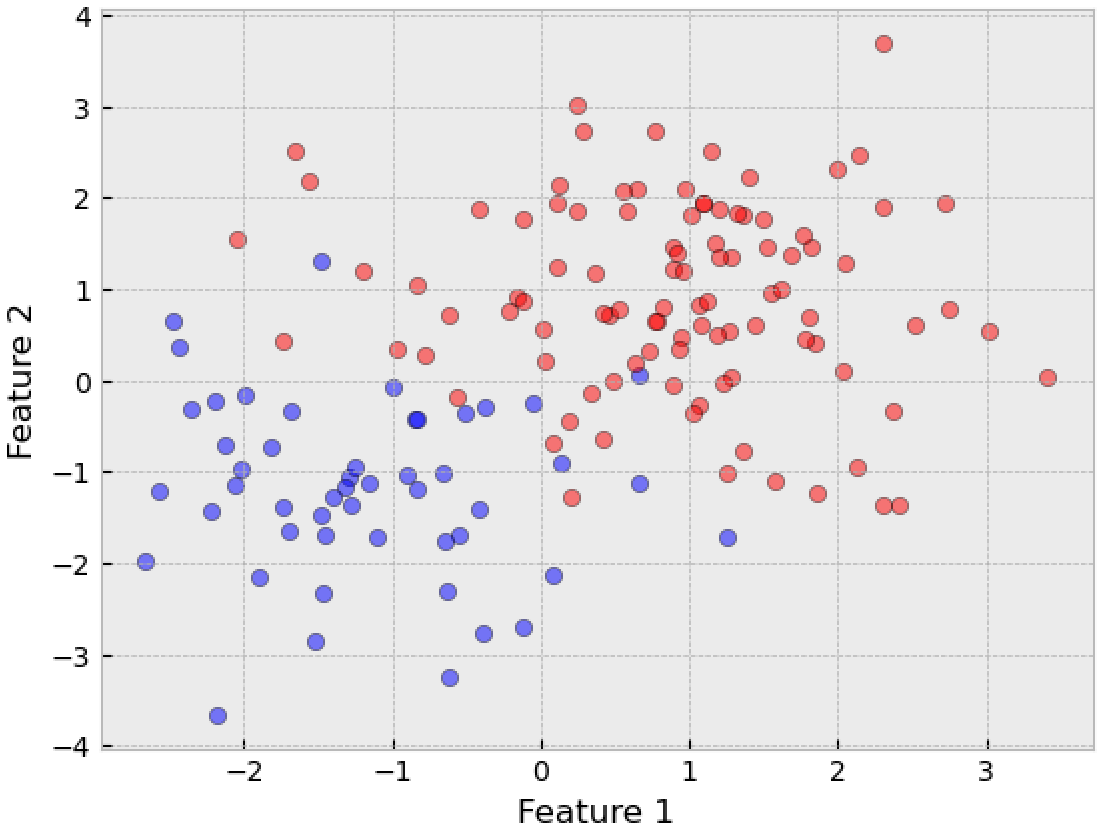
```
MAP decision threshold to decide C2 is > -0.98
```



Case 2: Multivariate Data Likelihood

In [13]:

```
mean1 = [-1, -1]
mean2 = [1, 1]
cov1 = [[1,0],[0,1]]
cov2 = [[1,0],[0,1]]
N1 = 50
N2 = 100

data, labels = generateData(mean1, mean2, cov1, cov2, N1, N2)
```



In the data we generated above, we have a "red" class and a "blue" class. When we are given a test sample, we will want to assign the label of red or blue.

We can compute the **posterior probability** for class C_1 as follows:

$$P(C_1|x) = \frac{P(x|C_1)P(C_1)}{P(x)}$$

Understanding that the two classes, red and blue, form a partition of all possible classes, then we can utilize the *Law of Total Probability*, and obtain:

$$P(C_1|x) = \frac{P(x|C_1)P(C_1)}{P(x|C_1)P(C_1) + P(x|C_2)P(C_2)}$$

Similarly, we can compute the posterior probability for class C_2:

$$P(C_2|x) = \frac{P(x|C_2)P(C_2)}{P(x|C_1)P(C_1) + P(x|C_2)P(C_2)}$$

Note that $P(C_1|x) + P(C_2|x) = 1$.

The Naive Bayes Classifier

Therefore, for a given test point $\mathbf{x}^*$, our decision rule is:

$$P(C_1|\mathbf{x}^*) \underset{C_2}{\overset{C_1}{\gtrless}} P(C_2|\mathbf{x}^*)$$

Using the Bayes' rule, we can further rewrite it as:

$$\begin{aligned} \frac{P(\mathbf{x}^*|C_1)P(C_1)}{P(\mathbf{x}^*)} &\underset{C_2}{\overset{C_1}{\gtrless}} \frac{P(\mathbf{x}^*|C_2)P(C_2)}{P(\mathbf{x}^*)} \\ \iff P(\mathbf{x}^*|C_1)P(C_1) &\underset{C_2}{\overset{C_1}{\gtrless}} P(\mathbf{x}^*|C_2)P(C_2) \end{aligned}$$

We assign $\mathbf{x}^*$ as class 1 if $p(\mathbf{x}^*|C_1)p(C_1) > p(\mathbf{x}^*|C_2)p(C_2)$, or assign $\mathbf{x}^*$ to class 2 if $p(\mathbf{x}^*|C_1)p(C_1) < p(\mathbf{x}^*|C_2)p(C_2)$.

This defines the **Naive Bayes Classifier**.

Training a Generative Classifier

- So, **to train the classifier**, what we need to do is to determine the parametric forms and the associated parameters for $P(x|C_1)$, $P(x|C_2)$, $P(C_1)$ and $P(C_2)$.

For example, we can assume that the data samples coming from either C_1 and C_2 are distributed according to Gaussian distributions. In this case,

$$P(x|C_k) = \frac{1}{(2\pi)^{1/2}|\Sigma_k|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\mu_k)^T\Sigma_k^{-1}(\mathbf{x}-\mu_k)\right\}, \forall k = \{1,2\}$$

- We can consider any distributional form we want.

What about the $P(C_1)$ and $P(C_2)$?

- We can consider the relative frequency of each class, that is, $P(C_i) = \frac{N_i}{N}$, where N_i is the number of points in class C_i and N is the total number of samples.

MLE Parameter Estimation Steps

For simplification, let's consider the covariance matrix Σ_k for $k = 1, 2$ to be **isotropic** matrices, that is, the covariance matrix is diagonal and the element along the diagonal is the same, or: $\Sigma_k = \sigma_k^2\mathbf{I}$.

- What are the parameters? The mean and covariance of the Gaussian distribution for both classes.

Given the assumption of the Gaussian form, how would you estimate the parameters for $p(x|C_1)$ and $p(x|C_2)$? We can use **maximum likelihood estimate** for the mean and covariance, because we are looking for the parameters of the distributions that *maximize* the data likelihood!

Assumption: Assuming the classes follow a (bivariate or 2-D) Gaussian distribution and, for simplicity, let's assume the covariance matrices are **isotropic**, that is, $\Sigma_k = \sigma_k^2\mathbf{I}$.

The MLE steps for parameter estimation are:

1. Write down the observed data likelihood, $\mathcal{L}^0$

$$
\begin{aligned}
\mathcal{L}^0 &= P(x_1, x_2, \ldots, x_N|C_k) \\
&= \prod_{n=1}^{N} P(x_n|C_k), \text{ data samples are i.i.d.} \\
&= \prod_{n=1}^{N} \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left\{-\frac{1}{2}(x_n-\mu_k)^T\Sigma_k^{-1}(x_n-\mu_k)\right\} \\
&= \prod_{n=1}^{N} \frac{1}{(2\pi)^{d/2}|\sigma_k^2\mathbf{I}|^{1/2}} \exp\left\{-\frac{1}{2\sigma_k^2}(x_n-\mu_k)^T\mathbf{I}(x_n-\mu_k)\right\} \\
&= \prod_{n=1}^{N} \frac{1}{(2\pi)^{d/2}(\sigma_k^2)^{d/2}} \exp\left\{-\frac{1}{2\sigma_k^2}(x_n-\mu_k)^T(x_n-\mu_k)\right\}
\end{aligned}
$$

where d is the dimensionality of the data space.

2. Take the log-likelihood, $\mathbf{L}$. This *trick* helps in taking derivatives.

$$
\begin{aligned}
\mathcal{L} &= \ln\left(\mathcal{L}^0\right) \\
&= \sum_{n=1}^{N}\left(-\frac{d}{2}\ln 2\pi - \frac{d}{2}\ln\sigma_k^2 - \frac{1}{2\sigma_k^2}(x_n-\mu_k)^T(x_n-\mu_k)\right)
\end{aligned}
$$

3. Take the derivative of the log-likelihood function with respect to the parameters of interest. For Gaussian distribution they are the mean and covariance.

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial\mu_k} &= 0\\ \sum_{n\in C_k}\frac{1}{\sigma_k^2}(x_n-\mu_k) &= 0\\ \sum_{n\in C_k}(x_n-\mu_k) &= 0\\ \sum_{n\in C_k}x_n-\sum_{n\in C_k}\mu_k &= 0\\ \sum_{n\in C_k}x_n-N_k\mu_k &= 0\end{aligned}$$

$$\mu_k=\frac{1}{N_k}\sum_{n\in C_k}x_n$$

This is the sample mean for each class. And,

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial\sigma_k^2} &= 0\\ \sum_{n\in C_k}-\frac{d}{2\sigma_k^2}+\frac{2(x_n-\mu_k)^T(x_n-\mu_k)}{(2\sigma_k^2)^2} &= 0\\ \sum_{n\in C_k}-d+\frac{(x_n-\mu_k)^T(x_n-\mu_k)}{\sigma_k^2} &= 0\\ \frac{\sum_{n\in C_k}(x_n-\mu_k)^T(x_n-\mu_k)}{\sigma_k^2} &= dN_k\end{aligned}$$

$$\sigma_k^2=\frac{\sum_{n\in C_k}(x_n-\mu_k)^T(x_n-\mu_k)}{dN_k}$$

This is the sample variance for each class. Then we can create $\Sigma_k=\sigma_k^2\mathbf{I}$, which is the (biased) sample covariance for each class.

In practice, if we want to estimate an entire covariance matrix, we would have to take the derivative of the log-likelihood function with respect to every entry in the covariance matrix. Covariance matrices are symmetric, so we only need to determine the upper (or lower) half of the covariance matrix.

We can determine the values for $p(C_1)$ and $p(C_2)$ from the number of data points in each class:

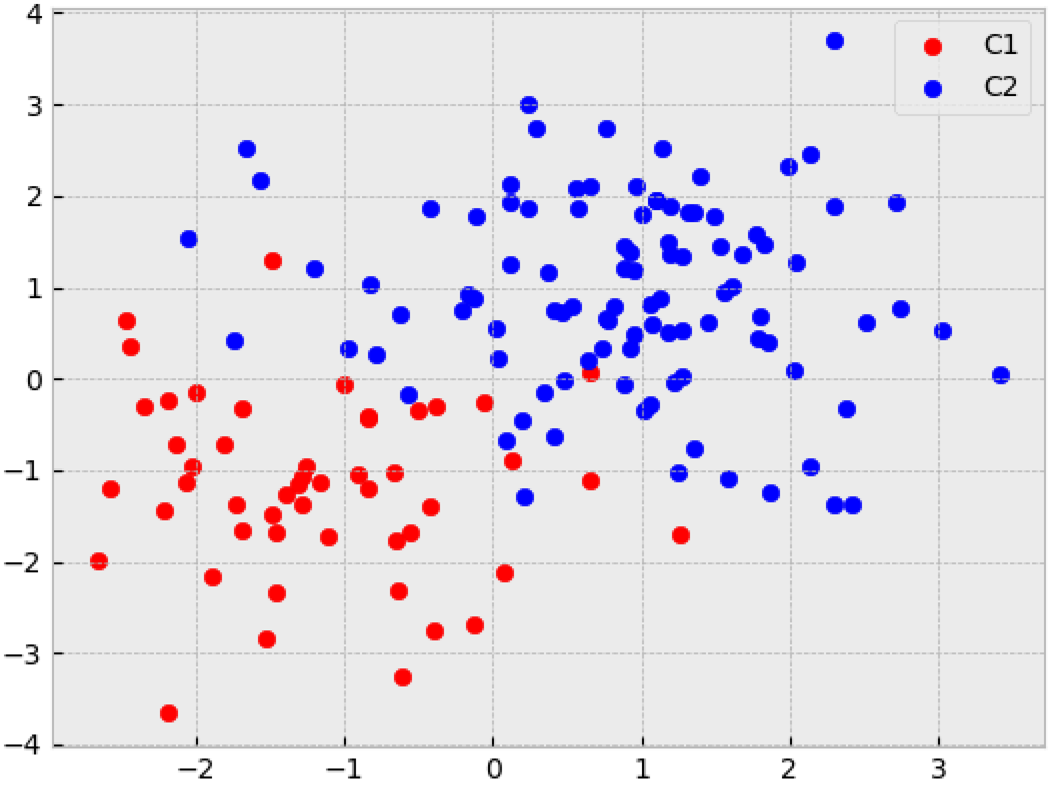
$$p(C_k)=\frac{N_k}{N}$$

where N is the total number of data points.

In [14]:
```
data.shape
```

Out[14]:
```
(150, 2)
```

In [15]:
```
plt.scatter(data[labels==1,0], data[labels==1,1], color='red', label='C1')
plt.scatter(data[labels==2,0], data[labels==2,1], color='blue', label='C2')
plt.legend();
```



In [16]:
```
#### Estimate parameters (with MLE solutions)
# Means
mu1 = np.mean(data[labels==1], axis=0)
print('Mean of Class 1: ', mu1)
mu2 = np.mean(data[labels==2], axis=0)
print('Mean of Class 2: ', mu2)

# Covariances - in this example we are showing the case where we estimate th
cov1 = np.cov(data[labels==1,:].T) # np.cov expects input to be D-by-N
print('Covariance of Class 1: ',cov1)
cov2 = np.cov(data[labels==2, :].T)
print('Covariance of Class 2: ',cov2)

#### Estimate Prior Probabilities
N = N1+N2
p1 = N1/N
print('Probability of  Class 1: ',p1)
p2 = N2/N
print('Probability of  Class 2: ',p2)
```

```
Mean of Class 1:  [-1.17739307 -1.18934324]
Mean of Class 2:  [0.89740324 0.91711432]
Covariance of Class 1:  [[ 0.81341209 -0.10429242]
 [-0.10429242  0.96472264]]
Covariance of Class 2:  [[ 1.09474519 -0.12481159]
 [-0.12481159  1.11647781]]
Probability of  Class 1:  0.3333333333333333
Probability of  Class 2:  0.6666666666666666
```

Next class.. we will define the data likelihood using these parameters and draw the decision boundary using the Naive Bayes Classifier.