# Naive Bayes & Introduction to Gaussians

Andreas C. Kapourani

2 March 2017

## 1 Naive Bayes classifier

In the previous lab we illustrated how to use Bayes Theorem for pattern classification, which in practice reduces to estimate the likelihood $P(\mathbf{x}|C_k)$ and the prior probabilities $P(C_k)$ for each class.

The prior probabilities are often easy to compute, however, to estimate the likelihood we need a statistical model which can provide a likelihood estimate. In the `Fish` example, the likelihood was estimated using a histogram for each class: the likelihood of a fish with length $x$ coming from class $k$ was estimated as the relative frequency of fish of length $x$ estimated from the training data for class $k$.

Now, imagine if in addition to the length we also had some other information such as weight and circumference, and each of these features could take a large number of possible values. Thus, we would need many more examples to obtain reasonable estimates of all possible combinations based on relative frequencies. This problem is known as the *curse of dimensionality*, where as the number of feature dimensions increases, the total number of possible feature vectors increases exponentially.

Naive Bayes approximation overcomes this problem by assuming that the different feature dimensions are *conditionally independent* given the class labels $C_k$. That is, the likelihood function becomes:

$$
\begin{aligned}
P(\mathbf{x}|C_k) &= P(x_1, x_2, ..., x_D|C_k) && (1)\\
&\approx P(x_1|C_k)P(x_2|C_k)...P(x_D|C_k) && (2)\\
&= \prod_d^D P(x_d|C_k) && (3)
\end{aligned}
$$

Hence, the posterior probability can be written as follows:

$$
\begin{aligned}
P(C_k|\mathbf{x}) &= \frac{P(C_k)\prod_d^D P(x_d|C_k)}{P(\mathbf{x})} && (4)\\
&\propto P(C_k)\prod_d^D P(x_d|C_k) && (5)
\end{aligned}
$$

and a test example can be classified again using the Bayes decision rule in exactly the same way as was described in the previous lab. Note that in Eq. 5, we have omitted the denominator from the Bayes rule (i.e. the evidence), since it does not depend on $C_k$. The $\propto$ symbol means `proportional to`.

### 1.1 Are they Scottish? dataset

To illustrate the Naive Bayes classifier, we will use an example dataset from David Barber's notes (`http://www.inf.ed.ac.uk/teaching/courses/lfd/lectures/lfd_2005_naive.pdf`).

Consider the following vector of attributes:

(likes shortbread, likes lager, drinks whiskey, eats porridge, watched England play football)

A vector $\mathbf{x} = (1, 0, 1, 1, 0)$ would describe that a person likes shortbread, does not like lager, drinks whiskey, eats porridge, and has not watched England play football. Together with each feature vector, there are class labels describing the nationality of the person, Scottish or English. Given a new test vector $\mathbf{x}^*$ we wish to classify it as either Scottish or English.

The Scottish and English training data are stored in the `xS` and `xE` matrices. Each row of the matrix corresponds to one training example, which consists of the 5 features described above.

```
% Scottish training examples, 7x5 matrix
xS = [1 0 0 1 1;
      1 1 0 0 1;
      1 1 1 1 0;
      1 1 0 1 0;
      1 1 0 1 1;
      1 0 1 1 0;
      1 0 1 0 0];

% English training examples, 6x5 matrix
xE = [0 0 1 1 1;
      1 0 1 1 0;
      1 1 0 0 1;
      1 1 0 0 0;
      0 1 0 0 1;
      0 0 0 1 0];
```

We can compute prior probabilities by finding the proportion of English and Scottish example out of the total observations:

```
% Total number of Scottish examples
N_S = size(xS, 1);
% Total number of English examples
N_E = size(xE, 1);

% Total examples
N_total = N_S + N_E;

% prior probability of Scottish example
prior_S = N_S / N_total;
% prior probability of English example
prior_E = 1 - prior_S;
```

Since in our example we have binary features (and we assume that they are drawn independently) our likelihood function follows a Bernoulli distribution. Let $p(x_d = 1|C_k) \equiv \theta_d$ be the probability that feature $x_d$ occurs in an example of class $k$. Then the probability that this feature $x_d$ doe not occur in an example of this class is $p(x_d = 0|C_k) = 1 - p(x_d = 1|C_k) \equiv 1 - \theta_d$, due to the normalization requirement.

Hence, Eq. 1 can be written as follows:

$$P(\mathbf{x}|C_k) = \prod_d^D P(x_d|C_k) = \prod_d^D \theta_d^{x_d}(1-\theta_d)^{(1-x_d)} \tag{6}$$

We can now estimate the likelihoods $\theta_d$ for each feature for both classes; that is, we compute the probability that feature $x_d$ occurs in the Scottish and English class.

```
% Estimated likelihood vector for each feature, Scottish class
lik_S = sum(xS) / N_S; % or just use : mean(xS)

% Estimate likelihood vector for each feature, English class
lik_E = sum(xE) / N_E;
```

Now, we assume that we have a test example $\mathbf{x}^* = (1, 0, 1, 1, 0)$ that we need to classify either as Scottish or English. We can compute directly Eq. 5, by plugging in Eq. 6 as follows:

```
% Test example
>> x=[1 0 1 1 0];

% Compute P(C = S) * P(x|C = S) = P(x, C = S)
>> test_prod_S = prior_S * prod((lik_S.^x) .* (1-lik_S).^(1-x))
test_prod_S =
    0.0404

% Compute P(C = E) * P(x|C = E) = P(x, C = E)
>> test_prod_E = prior_E * prod((lik_E.^x) .* (1-lik_E).^(1-x))
test_prod_E =
    0.0096

% or equivalently we could compute the Bayes decision rule
>> test_prod_S / test_prod_E
ans =
    4.1983
```

Hence, this example would be classified as Scottish. We could also compute the posterior probability as follows:

```
% Compute posterior probability
>> test_post_S = test_prod_S / (test_prod_S + test_prod_E)
test_post_S =
    0.8076
```

## Exercises

- Read what the `prod` function does.
- Use different priors and check if your classification assignment changes.
- Try to classify the vector $x = (0, 1, 1, 1, 1)$. In the training data, all Scottish people say they like shortbread. This means that $p(x, C_S) = 0$, and hence that $p(C_S|x) = 0$. This demonstrates a difficulty with sparse data - very extreme class probabilities can be made. One way to ameliorate this situation is to *smooth* the probabilities in some way, for example by adding a certain small number M to the frequency counts of each class. This ensures that there are no zero probabilities in the model.
- Create a test point based on your personal preferences and run the Naive Bayes classifier. Were you classified as Scottish or English?

### 1.2 Naive Bayes... a generative model

In the previous section we illustrated how we can use Naive Bayes to perform classification, e.g. assign people as Scottish or English; however, Naive Bayes can also be used to model how the data were actually generated. In Naive Bayes, since we compute the joint probability $P(\mathbf{x}, C)$ (by computing the likelihood $P(\mathbf{x}|C)$ and the prior $P(C)$), we essentially model the distribution of individual classes; hence, we are able to generate synthetic data for each individual class.

On the `Are they Scottish?` dataset, we will use the prior probabilities and the likelihoods that we learned in order to generate data using the following simple procedure:

1. Choose a class based on prior probabilities $P(C)$, i.e. sample a number uniformly between (0,1), if it is smaller than the prior probability of being Scottish, consider the class as Scottish, otherwise as English.
2. Sample a new feature vector based on the likelihoods $P(\mathbf{x}|C)$ from the class chosen in Step 1.

Let's illustrate this using a simple MATLAB code:

```
% Extract the number of features, i.e. 5
>> num_features = size(lik_S, 2);

% set seed for repeatable results (if not you will get different results)
>> rng(2);

% Step 1:
% The random sample is less than prior_S = 0.5385, so that class is
   considered as Scottish
>> sample = rand(1)
sample =
    0.4360

% Step 2:
% For each feature we get a sample uniformly in (0,1)
>> f_samples = rand(1, num_features)
f_samples =
    0.0259    0.5497    0.4353    0.4204    0.3303

% Get actual features (0 or 1), by checking if the samples are bigger
   than the actual likelihoods for the Scottish class or not.
>> generated_sample = f_samples < lik_S
generated_sample =
     1     1     0     1     1
```

Following the same procedure, we can create as many synthetic data as we want from each class. However, these generated samples make sense only if we have a good estimate of both the prior $P(C)$ and the likelihood $P(\mathbf{x}|C)$.

### Exercise

Create a function, that given the prior probabilities and the likelihoods for each class, it will generate N samples from each class. I.e. your function's signature should be something like `function [data_S, data_E] = generateData(prior_S, prior_E, lik_S, lik_E, N)` where `data_S` and `data_E` would be matrices of length N.

### 1.3 Further reading

There are two broad classes of classifiers, *generative models*, such as Naive Bayes, and *discriminative models*, such as logistic regression. Although both models are used for classification, generative classifiers learn a model of the joint probability, $P(\mathbf{x}, C) = P(\mathbf{x}|C)P(C)$, of the inputs $\mathbf{x}$ and the classes $C$, and make their predictions by using Bayes rule to compute the posterior $P(C|\mathbf{x})$, as we showed previously. On the other hand, discriminative classifiers model the posterior $P(C|\mathbf{x})$ directly, or learn a direct map from inputs $\mathbf{x}$ to the class labels $C$ (if you are interested you can read this nice chapter `https://www.cs.cmu.edu/~tom/mlbook/NBayesLogReg.pdf` by Tom M. Mitchell).

## 2 The Gaussian distribution

The second part of the lab focuses on building probabilistic models for real-valued data. The *Gaussian* or *Normal* distribution is the most common model to represent continuous-valued random variables whose distributions are not known.

The *probability density function (pdf)* is the usual way of describing the probabilities associated with a continuous random variable $X$, and gives us the the probability that $X$ lies in a small interval centred on $x$. The 1-dimensional (or univariate) Gaussian distribution has the following probability density function:

$$p(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right) \tag{7}$$

where $\mu$ is the mean of the Gaussian (i.e. where it is centred) and $\sigma^2$ is the variance. We wrote $p(x|\mu, \sigma^2)$ because the value of the *pdf* of $x$ depends on the parameters $\mu$ and $\sigma^2$. Another way to write the pdf is $\mathcal{N}(x; \mu, \sigma^2)$, where $\mathcal{N}$ denotes the Normal or Gaussian distribution.

Since we have the form for the *pdf* of the Gaussian, we can create a function `gaussian1D` which will evaluate Eq. 7 for a given $x$ and a set of parameters $\mu$ and $\sigma^2$. Write the following code to a file named `gaussian1D.m`:

```
function p = gaussian1D(mu, var, x)
    % Evaluate a 1D-Gaussian
    diff = (x - mu);
    % Evaluate pdf of Gaussian using Eq. 1
    p = 1/sqrt(2*pi*var) * exp(-0.5*diff.*diff/var);
end
```

Now, we can generate sequential data $x$ on a given region and evaluate the pdf for different combinations of $\mu$ and $\sigma^2$.

```
% Generate sequential data in the (-6,6) region
x = (-6:.1:6);

% Gaussian parameters N(x; 0,1)
mu = 0; var = 1;

% Evaluate the pdf using the above parameters
y = gaussian1D(mu, var, x);
```

Checking the values of the variables $x$ and $y$, you will observe that the values of $y$ are symmetric and the the largest value (`y = 0.3989`) is when $x = 0$. This result is expected since we defined the mean value of the Gaussian to be $\mu = 0$.

We can also plot the pdf of $x$ given the parameters, which will create the well known *bell curve* shape of the Gaussian distribution. We can evaluate the pdf for the same $x$ using different parameters:

```
% Gaussian parameters N(x; 0,5)
mu2 = 0; var2 = 5;
% Evaluate the pdf using the above parameters
y2 = gaussian1D(mu2, var2, x);
```

Check the values of the `y2` variable, and compare them with the `y` variable. What is its largest value? Do you understand why?

We can easily visualize the two Gaussian *pdf*s using the following code:

```
% Plot the pdf for N(x; 0,1)
plot(x, y); hold on;
% Plot the pdf for N(x; 0,5)
plot(x, y2);
% Add labels
xlabel('x'); ylabel('p(x|m,s)');
title('pdf of Gaussian distributions')
legend('p(x|m=0, s=1)', 'p(x|m=0, s=5)')
```

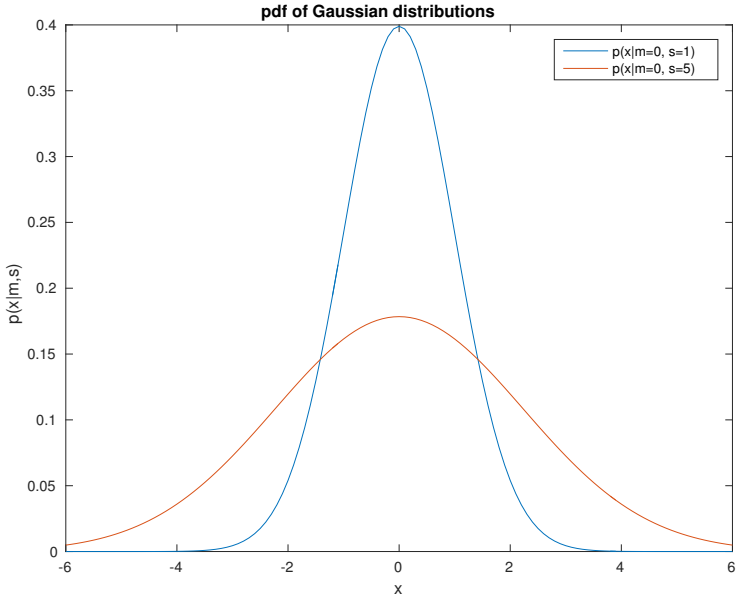



Figure 1: *Pdfs of two different Gaussian distributions.*

## Exercise

*Figure 1* shows the pdfs of the two Gaussians with zero mean and different variance. As you can observe the mean $\mu$ specifies where the Gaussian will be centred and the variance $\sigma^2$ denotes how dispersed the values would be around the mean. Evaluate the pdf for the following Gaussians in the (-7, 7) region and show all the plots in the same figure similarly to *Figure 1* (observe that the y-axis value can be more than 1) :

- $\mathcal{N}(x; \mu = 0, \sigma^2 = 0.1)$
- $\mathcal{N}(x; \mu = 0, \sigma^2 = 3)$
- $\mathcal{N}(x; \mu = 4, \sigma^2 = 0.5)$
- $\mathcal{N}(x; \mu = -5, \sigma^2 = 0.5)$

### 2.1 Parameter Estimation

In real life problems, we do not have the actual parameters of the distribution (in the Gaussian case are $\mu$ and $\sigma^2$), but we have collected some data, and based on those we can estimate the parameters. This process of estimating the parameters from the training data is referred to as *fitting* the model to the data. One way to choose the parameters is to maximise the likelihood of the model generating the training data. For the Gaussian distribution the *Maximum Likelihood Estimate* (MLE) of the mean and the variance are:

$$\hat{\mu} = \frac{1}{N}\sum_{n}^{N} x_n \tag{8}$$

$$\hat{\sigma}^2 = \frac{1}{N}\sum_{n}^{N}(x_n - \hat{\mu})^2 \tag{9}$$

where $x_n$ denotes the feature value of the $n^{th}$ sample and $N$ is the number of samples in total.

Suppose that we have the following data:

| | | | | | | |
|---|---|---|---|---|---|---|
| Class S | 10 | 8 | 10 | 10 | 11 | 11 |
| Class T | 12 | 9 | 15 | 10 | 13 | 13 |

We assume that each class may be modelled by a Gaussian distribution. Using these data, we will estimate the mean and variance parameters for each class using the equations shown above, and then we will sketch the pdf for each class.

```
% Data in class S
>> xS = [10 8 10 10 11 11];
% Data in class T
>> xT = [12 9 15 10 13 13];
% Get the length of the data set
>> N = length(xS);

% Compute mean and variance for S class
>> mu_S_hat = 1/N * sum(xS)
mu_S_hat =
    10
>> s_S_hat = 1/N * sum( (xS - mu_S_hat).^2)
s_S_hat =
     1
% Compute mean and variance for T class
>> mu_T_hat = 1/N * sum(xT)
mu_T_hat =
    12
>> s_T_hat = 1/N * sum((xT - mu_T_hat).^2)
s_T_hat =
     4
```

### Exercises

- Plot the estimated Gaussian pdfs for the two different classes.
- Assume that for class S you observed a new data point x = 18, i.e. your vector of observations is the following `xS = [10 8 10 10 11 11 18]`. Estimate again the mean and variance for the S class and then plot the estimated Gaussian pdf. What do you observe?

## 3 Multivariate Gaussian distribution

We can extend the univariate Gaussian to the multivariate (i.e. multidimensional) case when we have data with more than one variables. The D-dimensional Gaussian distribution is parameterised by a mean vector, $\boldsymbol{\mu} = (\mu_1, ..., \mu_D)^T$, and a $D \times D$ covariance matrix, $\boldsymbol{\Sigma} = (\sigma_{ij})$. The probability density function is given by:

$$p(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\Big( -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\Big) \tag{10}$$

where $|\boldsymbol{\Sigma}|$ is the determinant of the covariance matrix $\boldsymbol{\Sigma}$.

Let us create a function `gaussianMV` which will evaluate the multivariate Gaussian distribution with mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ for a given data point $\mathbf{x}$.

```
function p = gaussianMV(mu, covar, x)
    % Evaluate a multivariate Gaussian pdf

    % Get dimensions of covariance matrix
    [d b] = size(covar);

    % Check that the covariance matrix is square
    if (d ~= b)
        error('Covariance matrix should be square');
    end

    % force MU and X into column vectors
    mu = reshape(mu, d, 1);
    x = reshape(x, d, 1);

    % Evaluate quntity shown in Eq. 10
    p = 1/sqrt((2*pi)^d*det(covar)) * ...
           exp(-0.5*(x-mu)'*inv(covar)*(x-mu));
end
```

Given a specific mean vector and a covariance matrix we can compute the pdf of the multivariate Gaussian for any given point $\mathbf{x}$. Let's see some examples:

```
% Zero mean MV Gaussian
>> mu = [0 0];

% Covariance matrix is the identity matrix
>> Sigma = [1 0; 0 1];

% Evaluate the pdf for the point x1 = (0,0)
>> p1 = gaussianMV(mu, Sigma, [0 0])
p1 =
    0.1592

% Evaluate the pdf for the point x2 = (2,2)
>> p2 = gaussianMV(mu, Sigma, [2 2])
p2 =
    0.0029
```

As we expected, the probability of seeing a point at $\mathbf{x}_1 = (0, 0)$ is higher than $\mathbf{x}_2 = (2, 2)$ given that our Gaussian distribution is centred at the location $\boldsymbol{\mu} = (0, 0)$.

### 3.1 Plot 2D Gaussian distributions

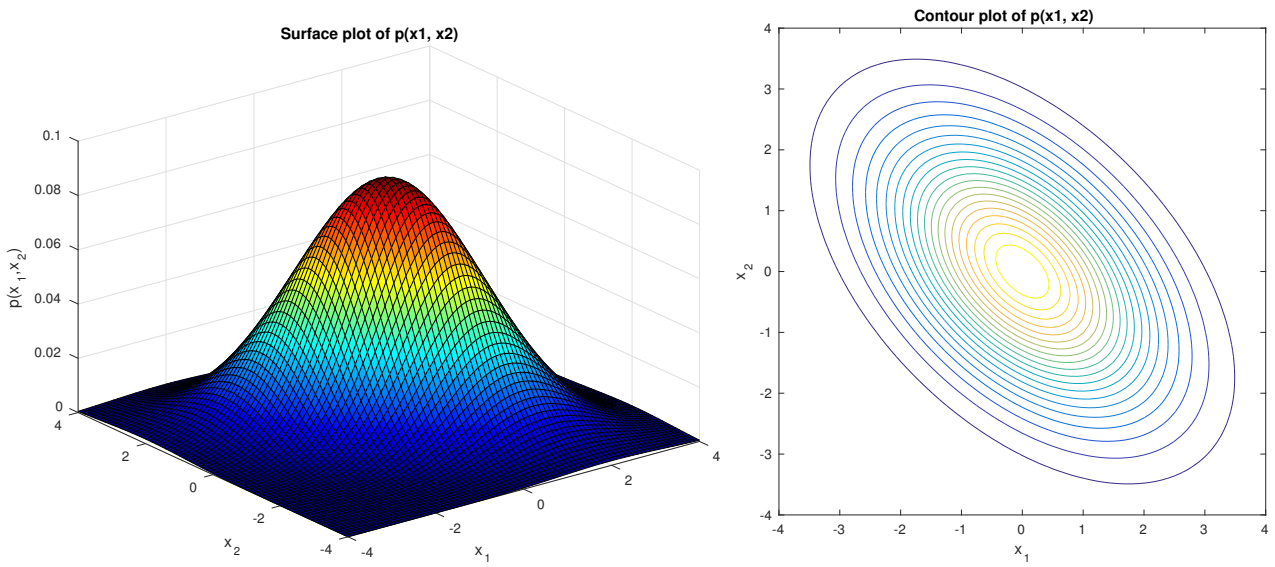
We can create *contour* and 3D plots so as to visualize the pdf of 2-D Gaussian distributions for different combinations of mean vectors and covariance matrices. We consider a two-dimensional Gaussian with the following mean vector and covariance matrix:

$$\boldsymbol{\mu} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad \boldsymbol{\Sigma} = \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix} \tag{11}$$

```
% Initialize mu and Sigma
mu = [0 0];
S = [2 -1; -1 2];

% Get the variance for each axis
var = diag(S);
% keep only the maximum variance
maxsd = max(var(1),var(2));

% plot between +-2SDs along each dimension
% location of points at which x is calculated
x = mu(1)-2*maxsd:0.1:mu(1)+2*maxsd;
% location of points at which y is calculated
y = mu(2)-2*maxsd:0.1:mu(2)+2*maxsd;

% Get the number of data
N = length(x);

% (Inefficient) Compute the Gaussian pdf for each (x,y) pair
z = zeros(N, N);
for i = 1:N
  for j = 1:N
    xx = [x(i) y(j)];
    % Evaluate MV Gaussian pdf for the (x,y) pair
    z(i,j) = gaussianMV(mu, S, xx);
  end
end
```

The given code evaluates the pdf of 2D Gaussian for each combination of $(x, y)$ pairs. Now we can just call the `surf` and `contour` functions to create the figures, as we had shown in Lab session 2.

```
% Make surface plot of 2D Gaussian
figure(1);
colormap(1,'jet');
surf(x,y,z);
xlabel('x_1'); ylabel('x_2');
zlabel('p(x_1,x_2)');

% Make contour plot of 2D Gaussian
figure(2);
contour(x,y,z, 20);
axis equal;
```




(a) (b)

Figure 2: *Surface and contour plots of 2-dimensional Gaussian with covariance structures given in Equation 5.*

## Exercises

- Use different combination of covariance structures and see how the shape of the pdf for the 2D Gaussian changes. Click the `Rotate 3D` button to see the plot from different angles.
- Now keep fixed the covariance matrix and change the mean vector. What do you observe?
- The code given above for evaluating the pdf for each (x,y) pair is inefficient, since we use two `for` loops. How would you change the code so as to be more efficient? **Hint**: Use the `meshgrid` function and write the quadratic form fully to estimate the pdf for every pair of (x,y) as it is shown in the lecture notes.