# Naive Bayes Predicts Diabetes

The test problem we will use in this example is the Pima Indians Diabetes problem. This problem consists of 767 observations of medical data for Pima Indians patients. The medical data are records including measurements taken from the patient which we consider as attributes for classification. All patients are women aged 21 or older. All attributes are numeric, and their units vary from attribute to attribute. There are eight attributes and these are: 1. Number of times pregnant, 2. Plasma glucose concentration every 2 hours in an oral glucose tolerance test, 3. Diastolic blood pressure (mm Hg), 4. Triceps skin fold thickness (mm), 5. 2-Hour serum insulin (mu U/ml), 6. Body mass index (weight in kg/(height in m)^2), 7. Diabetes pedigree function, 8. Age (years). The last column is the Class variable which is either 0 or 1 and indicates whether the patient has suffered an onset of diabetes within the 5 years of when the measurements were taken (1) or not (0). This is a standard machine learning dataset in literature.

## Training stage

The first step in classification is **Handle Data.** This means that we have to load the data from a CSV file and split it into two datasets, training and test. Next we need to split the load into a training dataset that the Naive Bayes algorithm can use to obtain the necessary data needed for prediction, and into test data. The test data is used to evaluate the accuracy of the classificator, by predicting the test sets and comparing it with real data. We take two thirds of the data as the training dataset and the other one third as the test data (this is a ratio is commonly used for testing an algorithm)

```
DataSet := csv read("dataset.csv" , true)
TrainingSet := subset(DataSet , 0 , 0 , 510 , 8)
TestSet := subset(DataSet , 511 , 0 , 766 , 8)
number_class := 2
number_attributes := 8
s_class := [0; 1]
subset(TrainingSet , 0 , 0 , 0 , 8) = [1  85  66  29  0  26.6  0.351]
```

Naive Bayes training function prepares the data to be used in the Naive Bayes classification function. All attributes are considered to be statistical processes according to Gaussian distribution. The training process is to determine the mean and standard deviation for all attributes per class and the probability of each class as well. These values are used to calculate the probabilities which are necessary to make an informed decision. The next line displays how the training function is called in MatDeck. After that, the result of the training function is displayed.

```
MM1 := naivebayest(TrainingSet , number_attributes , number_class , s_class)
```

Below is a sample to be classified as possible diabetes onset or not.

```
TestData := subset(TestSet , 0 , 0 , 0 , 7)
TestData = [9  91  68  0  0  24.2  0.2]
```

## Classification stage

In order to determine which class posterior is greater, in this case male or female, we have to calculate the probabilities based on mean and standard deviation for all attributes per class. The greatest posterior determines the class for the test data.

```
naivebayesc(MM1 , number_attributes , number_class , s_class , TestData) = 0
```

We compare the predicted value with the given diabetes onset for the given TestData sample, and we see that the decision is correct in this case.

```
TestSet[8] = 0
```

## Classification accuracy

In order to determine the accuracy of a classification, we need to take a bigger dataset TestSet having obtained 256 records as explained above. For each record we use the Naive Bayes classification to make the decision on diabetes. Further, we count the number of correct decisions and determine the probability that the correct decision was made.

```
accuracy := nb_accuracy(TestSet)
accuracy = 0.773

nb_accuracy(TSet)
{
  Count := 0
  N_s := rows(TSet )
  for(i := 0 , i < N_s , i += 1 )
  {
    Tdata := subset(TSet , i , 0 , i , 7)
    Temp1 := naivebayesc(MM1 , number_attributes , number_class , s_class , Tdata )
    ind := i · (number_attributes + 1 ) + number_attributes
    if(Temp1 == TSet[ind])
    {
      Count += 1
    }
  }
  return(Count / N_s )
}
```