Implementation of Naive Bayes in MatDeck

The Naive Bayes algorithm is an effective algorithm and it is one the very first methods used in classification and machine learning implemented in MatDeck. The Naive Bayes algorithm is a method that uses the probabilities of all attributes and features belonging to each class to make an informed decision about the classification. The Naive Bayes simplifies the calculation of probabilities by assuming that the probability of each attribute in a given class is independent of all other attributes. This assumption results in a fast and all round effective method.

Training stage

In the next segment, we give a standard example of sex classification used by Wikipedia to mathematically illustrate the Naive Bayes approach and its benefits. In this problem, the program must classify whether a given person is a male or a female based on given features. These features (attributes) are height, weight and foot size.

The starting point for this classification is using training set which consists of a given set of measured attributes and classes. An example of a training set is given below. In this example there are two classes, male and female, and three main attributes which are height, weight and foot size. We define all the required variables in this manner.

```
TrainingSet := [ 182  81.6  30   "male"
                 180  86.2  28   "male"
                 170  77.1  30   "male"
                 180  74.8  25   "male"
                 152  45.4  15   "female"
                 168   68   20   "female"
                 165   59   18   "female" ⋮ ]

number_class := 2
number_attributes := 3

s_class := [ "female"
             "male" ]
```

Naive Bayes training function prepares the data for the Naive Bayes classification function. All attributes are considered to be statistical processes in accordance to Gaussian distribution. The training process is used to determine mean, and standard deviation for all attributes per class, and probability of each class. These values are used to calculate all 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.

```
MM := naivebayesT(TrainingSet , number_attributes , number_class , s_class)
```

Below is a sample to be classified as male or female.

$$\text{TestData} := \begin{bmatrix} 170 & 65 & 30 \end{bmatrix}$$

$$\text{MM} = \begin{bmatrix} \begin{bmatrix} \begin{bmatrix} 165 & 60.1 & 19 \end{bmatrix} & \begin{bmatrix} 178 & 79.925 & 28.25 \end{bmatrix} \end{bmatrix} \\ \begin{bmatrix} \begin{bmatrix} 8.337 & 9.248 & 2.915 \end{bmatrix} & \begin{bmatrix} 4.690 & 4.371 & 2.046 \end{bmatrix} \end{bmatrix} \\ \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \end{bmatrix}$$

Data obtained by training, mean and standard deviation per class, and class probabilities

Classification stage

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

$$\text{naivebayesC}(\text{MM}, \text{number\_attributes}, \text{number\_class}, \text{s\_class}, \text{TestData}) = \text{"male"}$$

Script codes for Naive Bayes functions

Naive Bayes Training function

```
1   naivebayesT(Train, N_a, N_c, c_in)
2   {
3     counter := vector create(N_c, false, 0)
4     VecM := vector create(N_c, true, 0)
5
6      // Create a vector for every class, with dimension equal to the number of attributes
7     for(k := 0; k < N_c; k += 1)
8     {
9       VecM[k] = vector create(N_a, true, 0)
10    }
11
12     // Classify training data, matrix Train, into classes
13    for(i := 0; i < rows(Train); i += 1)
14    {
15
16      for(j := 0; j < N_c; j += 1)
```

```
17
18          {
19            ind := i * (N_a + 1) + N_a
20
21             // If the row is a memeber of class j, update the counter per class
22            if(Train[ind] == c_in[j])
23            {
24
25               // Put row into class
26              if(counter[j] == 0)
27              {
28                VecM[j] = subset(Train, i, 0, i, N_a - 1)
29              }
30              if(counter[j] > 0)
31              {
32                VecM[j] = join mat rows(VecM[j], subset(Train, i, 0, i, N_a - 1))
33              }
34              counter[j] = counter[j] + 1
35            }
36          }
37        }
38
39        Vec_mean := vector create(N_c, true, 0)
40        Vec_dev := vector create(N_c, true, 0)
41        N_samp := 0
42         // Calculate mean and average per class
43        for(i := 0; i < N_c; i += 1)
44        {
45          Temp := VecM[i]
46          Vec_mean[i] = col average(Temp)
47          Vec_dev[i] = col deviation(Temp)
48          N_samp += counter[i]
49        }
50
51        counter = counter / N_samp
52
53         // Prepare training data
54        Traindata := vector create(3, false, 0)
55        Traindata[0] = Vec_mean
56        Traindata[1] = Vec_dev
57        Traindata[2] = counter
58        return(Traindata)
59      }
```

Naive Bayes Classification function

```
naivebayesC(Train_D, N_a, N_c, c_in, Data_in)
{
  // Adjust training data
  Avg := Train_D[0]
  Dev := Train_D[1]
  p_c := Train_D[2]
  p_p_c := vector create(N_c, false, 0)
  evidence := 0
  p_M := 0.0
  M := 0

  // Test the probability that a given data belongs to a certain class
  for(i := 0; i < N_c; i += 1)
  {
    p_temp := p_c[i]

    // Caluclate the probabilities for all attributes
    for(j := 0; j < N_a; j += 1)
    {
      p_temp = p_temp * normaldens(Data_in[j], Avg[i][j], Dev[i][j])
    }

    p_p_c[i] = p_temp
    evidence += p_temp

    // Find the maximum probability per class, determine the class index M
    if(p_temp > p_M)
    {
      p_M = p_temp
      M = i
    }
  }
  return(c_in[M])
}
```