Naive Bayes Classifier

Alice Rivi
Robotics Engineering
University of Genoa, Genoa
S5135011@studenti.unige.it

***Abstract*—The assignment's objective is to create a Naive Bayes Classifier that can predict potential weather conditions for tennis play. Data pre-processing is the first thing that needs to be done. A data set with a specific number of observations is given to us. In particular, our data set has fourteen rows and four columns, with each row's columns indicating a different attribute.**

In addition, there is a fifth column in which the decision, in our case, whether to play tennis, is stored. Our first objective is to turn the provided data set into a numeric one so that MATLAB, the program we're using, can read it. In order to interpret the data set, we are also given a file with a brief summary of the project's objectives and the data set's features. We started writing the code as soon as the data set was ready for the task.

The project consists of two different files: the first implemented is a function that accepts the training data set and the test set as inputs and outputs the target, an array containing the results, and the error rate, which is a number that represents the error calculated and committed due to the decision made.

The second file is a script in which we use the function; here, we load the data set and choose the percentage of rows being the training set and the test set: a comma separated list. The construction of the Naive Bayes Classifier is the subject of the second task, which is broken up into two stages: training and testing. By successfully navigating an earlier check of the matrices dimensions, all of this is possible. We determined the probability for the various combinations present in the test set, and the findings were then entered into a cell array.

In order to determine the likelihood of detecting a value while knowing the potential outcomes from other observations, the third task to be implemented is to enhance the classifier with Laplace additive smoothing. The last outcome we got was the error rate, which was calculated as an error among all the options to assess where we stand in the various conditions analyzed.

1. Introduction

In practical learning issues, the assumption that the qualities used to derive a prediction are independent of one another, given the anticipated value, is rarely true. This is the foundation of the Naive Bayes algorithm. It has been demonstrated that the independence assumption is less limiting than could be anticipated for classification issues where the predicted value is categorical. Naive Bayes has much lower error rates than more complex learning schemes, like those that learn uninvariate decision trees, for a number of real-world classification problems.

The objective of this project is to construct a Naive Bayes Classifier using a set of known data. Based on the Bayes theorem, a Naive Bayes Classifier calculates conditional probabilities, or the likelihood that something will happen if another event has already happened. Utilizing prior knowledge, we may determine the conditional probability of a specific event.

The Naive Bayes Classifier forecasts the likelihood that a given record or data point belongs to a specific class for each class, for example. The most likely class is the one with the highest likelihood. The Naive Bayes Classifier makes the premise that all features are independent of one another, therefore the presence or absence of one feature has no bearing on the presence or absence of another. According to the data set provided for the experiment, we have multiple classes, and we want to make a forecast, thus we want to know whether or not a specific event will occur based on the data.

Since we have a data set, we randomly divide it into two halves, with the first portion serving as the training set and the second serving as the test set. This method is frequently used in machine learning. The training set gives us a broad understanding of the probabilities and represents all of the data we have. Thanks to the training set, we may further develop our conclusions after examining the test set. As a result, based on what we discovered in the first stage of the technique, we can predict a potential result for a certain set of features taken from the test set. To improve the outcomes of our decisions, we added a Laplace additive smoothing at the end of the Nave Bayes Classifier implementation.

2. Data processing

The so-called Data Processing is the first duty we improved.

A textual data set is sent to us, as in the Figure 1.

Since we are unable to expound on literal data, we translated it into a numerical format using a legend with only positive numbers: it is not necessary to assign each word a specific value, only that the data be consistent.

#Outlook	Temperature	Humidity	Windy	Play
overcast	hot	high	FALSE	yes
overcast	cool	normal	TRUE	yes
overcast	mild	high	TRUE	yes
overcast	hot	normal	FALSE	yes
rainy	mild	high	FALSE	yes
rainy	cool	normal	FALSE	yes
rainy	cool	normal	TRUE	no
rainy	mild	normal	FALSE	yes
rainy	mild	high	TRUE	no
sunny	hot	high	FALSE	no
sunny	hot	high	TRUE	no
sunny	mild	high	FALSE	no
sunny	cool	normal	FALSE	yes
sunny	mild	normal	TRUE	yes

Figure 1. Dataset

We decided on the following legend [Figure 2]:

```
#Outlook legend          #Humidity legend        #Play legend
Sunny = 3                High = 2                No = 1
Overcast = 2             Normal = 1              Yes = 2
Rainy = 1

#Temperature legend      #Windy legend
Hot = 3                  False = 1
Mild = 2                 True = 2
Cool = 1
```

Figure 2. Legend to convert the dataset into a numerical one

The data set was coded using the respect legend, and we got a matrix of positive integer numbers [Figure 3].

```
2 3 2 1 2
2 1 1 2 2
2 2 2 2 2
2 3 1 1 2
1 2 2 1 2
1 1 1 1 2
1 1 1 2 1
1 2 1 1 2
1 2 2 2 1
3 3 2 1 1
3 3 2 2 1
3 2 2 1 1
3 1 1 1 2
3 2 1 2 2
```

Figure 3. Numerical matrix obtained converting the given data set

After finishing this phase, we had a matrix that we could work with using the MATLAB software.

3. Build a Naive Bayes Classifier

The construction of the classifier was the second problem to be solved. First, before beginning the various phases, we had to confirm that the training set and test set's dimensions were accurate.

As reference we assumed:

- training set which is a matrix with dimensions $n\ x\ d+1$;
- test set which is a matrix with dimensions $m\ x\ c$.

The initial check made sure that the training set matrix's number of columns had dropped by 1 and the test set matrix's number of columns was at least c.

Two more tests needed to be run after this initial check produced a favourable outcome:

- each and every matrix must include only positive entries. The program generates an error message if only one of the numbers is negative;
- if a value in the test set does not follow this rule, the entire row of the test set matrix must be eliminated. All the values in the test set must also be present in the training set..

When both sets were prepared for usage, we began to elaborate the data. In order to use this set in the first portion, we first calculated the number of classes as the number of unique values taken from the last column of the training set matrix.

We calculated the probability for each variable and each class using the number of classes—in our case, there were two classes, each represented by the yes or no decision—and the four variables—Outlook, Temperature, Humidity, and Windy—that we had previously mentioned.

The following formulas were used to determine our probabilities:

$$P(x_k = true|t_i) = \frac{Numberx_k = TrueInClasst_i}{TotObservationOfClassti} \quad (1)$$

$$P(x_k = true|t_i) = \frac{N_{true,t_i}}{N_{t_i}} \quad (2)$$

$$P(x_k = false|t_i) = 1 - P(x_k = true|t_i) = \frac{N_{true,t_i}}{N_{t_i}} \quad (3)$$

where we know the prior probability of the class, calculated as:

$$P(t_i) = \frac{NumberObservationOfClasst_i}{NumberObservationsInTrainingSet} = \frac{N_{t_i}}{N} \quad (4)$$

We filled a cell array in our MATLAB software, initialising it correctly with the necessary dimensions. This is a matrix in which every location $(i; j)$ is an array (there is also the option to have a different dimension for each array), and we then entered the calculated probability in each array location.

In our situation, the cell array was a $(2, 4)$ matrix in which the corresponding arrays had a dimension of 2 or 3 depending on the various potential values of each characteristic.

Once the probability cell array has been adequately filled, we must compute the decision's outcome based on the data analysed.

We had to take the calculated probability into account in order to make a choice. Every choice we make has a price, and if that price is high, we know we made the incorrect choice.

A $loss\ function\ \lambda(y, \omega)$ is used to calculate the cost, with y representing the decision made and representing the natural state. The loss function's job is to reduce risk, which is conditional due of the made-based observation.

Following is a calculation of the risk:

$$R(y_i|\mathbf{x}) = \sum_{j=1}^{c} \lambda(y_j, \omega_j) P(\omega_j|\mathbf{x}) \tag{5}$$

and when we get the experimental observation **x**, it is the conditional risk of a decision y_i.

Of course, the risk needs to be calculated for all potential scenarios. Since we are dealing with discrete and categorical variables in this example, we will compute the general risk as follows:

$$R = \sum_{x \in X} R(y(x)|\mathbf{x}) P(\mathbf{x}) \tag{6}$$

where P(x) is the probability mass function of experimental observation and X is the set of all possible inputs (the input space).

We compute c blocks $g_j, j = 1...c$, called $discriminant\ functions$, that computes the total risk.

Since all the variables are independent, the Naive assumption allows us saying $P(x_1, ..., x_d|t_i) = P(x_1|t_i)P(x_2|t_i).....P(x_x|t_i)$ so the discriminant functions can be computed as:

$$g_i(\mathbf{x}) = P(t_i)[P(x_1|t_i)P(x_2|t_i)...P(x_d|t_i)] \tag{7}$$

$$g_i(\mathbf{x}) = P(t_i) \prod_{j=1}^{d} P(x_j|t_i) \tag{8}$$

In order to reduce risk, we will select the lowest value possible for each option. As a result, in the software, we will, if necessary, replace the value already existing in our array with one that incurs a lesser cost.

After obtaining all the findings, we also assessed the error rate to see how far the program's judgments departed from reality.

The error rate is simply computed by calculating the following formula:

$$Error = \frac{NumberOfTimesWrongDecision}{TotalDecisionTaken} \tag{9}$$

4. Improve the Classifier with Laplace smoothing

The classifier's Laplace smoothing needs to be added as the final step. This decision was made because there are a lot of zeros in the discriminant functions because there aren't much data in the small data set.

In order to prevent having too many zeros, we had to recalculate the probabilities for that work. In specifically, we have a variable x that assumes values in the range $1, 2, ..., v$, with v the potential discrete values.

The following is the Laplace probability of seeing a particular value i:

$$P(x = i) = \frac{n_i + a}{N + av} \tag{10}$$

where n_i is the times the value i occurs and a is a parameter properly chosen:

- a¿1 means that we trust the prior belief more than the data;
- a¡1 means that we trust the prior belief less than the data;

for our purpose we will use a=1.

Of course, we calculated the error rate for this second set of findings as well and then compared it to the one we had previously computed. The comparison demonstrated that employing Laplace smoothing can reduce error rates in contrast to not using them.

As an example have a look at the Figure 4:

```
Target without Laplace:
     2     2     2     2

Target with Laplace:
     1     1     1     1

Classification without Laplace:
         0    0.0098
         0    0.0211
         0    0.0059
         0    0.0117

Classification Laplace:
    0.1000    0.0422
    0.1500    0.0625
    0.1000    0.0281
    0.0500    0.0469

Error Rate without Laplace: 0.75
Error Rate with Laplace: 0.25
```

Figure 4. Results taken from a random experiment

The figure illustrates what we already said. The $Classification$ result shows the $discriminant\ functions$, and we can see that the zeros can be avoided by using Laplace smoothing. Additionally, the error computed by using Laplace smoothing is lower than before, producing a better decision-making outcome.

For example, we can see in the $target$ that the values calculated differ for the first element in the array, indicating that the decision made in that case was better.