Naive Bayes Classifier

Machine Learning 2021/22: 1^{st}**Assignment**

Matteo Maragliano
EMARO-European Master on Advanced Robotics
DIBRIS - Dip. Informatica, Bioingegneria, Robotica, Ing. dei Sistemi
Università degli Studi di Genova

***Abstract*—The goal of the assignment is to develop a Naive Bayes Classifier to determine possible weather conditions for playing tennis. The first step to be done is the so called Data pre-processing. We are given a data set with a certain number of observations, in particular our data set contains fourteen rows with four columns, for each row, each one representing an attribute; moreover, there is a fifth column in which is stored the decision, *yes* or *not* in our case, for playing tennis. Our first task consists in converting the given data set into a numeric one so that it can be read by the program used, MATLAB.**
We are also given a file containing a brief description of the goal of the project and the characteristics of the data set in order to be able to interpret it.
Once the data set is available for the work we started writing the code.
The project contains two different files: the first implemented is a function which receives as inputs the training data set and the test set and returns as output the target, an array containing the results and the error rate, a number representing the error computed and committed due to the decision taken; the second file is a script in which we use the function, here we load the data set and decide the percentage of rows being the training set and the test set: a common percentage is 70% of the total for the training set while the remaining 30% for the test set, both are taken randomly.
The second task concerns the building of the Naive Bayes Classifier: the task is divided in two phases, one of training and one of testing. All of this can be done by passing a previous check of the matrices dimensions.
We calculated the probabilities for the different combinations found in the test set and then we fill a cell array with the results. At the end, the third task to be implemented is to improve the classifier with Laplace additive smoothing in order to calculate the probability of observing a value knowing the possible outcomes from other observations.
The last result we obtained is the error rate, calculated as an error among all the possibilities in order to see how far we are in the different conditions processed.

I. Introduction

NAIVE Bayes relies on an assumption that is rarely valid in practical learning problems: that the attributes used for deriving a prediction are independent of each other, given the predicted value. It has been shown that, for classification problems where the predicted value is categorical, the independence assumption is less restrictive than might be expected. For several practical classification tasks, naive Bayes produces significantly lower error rates than more sophisticated learning schemes, such as ones that learn univariate decision trees.

This project has the goal of building a Naive Bayes Classifier starting from an already known data set. A Naive Bayes Classifier is based on the Bayes's theorem, so it works on conditional probabilities: the probability that something will happen, given that something else has already occurred. We can calculate the conditional probability of a certain event using its prior knowledge.

Naive Bayes Classifier predicts membership probabilities for each class such as the probability that a given record or data point belongs to a particular class. The class with the highest probability is considered as the most likely one.
The assumption taken by the Naive Bayes Classifier is that all features are unrelated to each other, so the presence or absence of a particular feature does not affect the presence or absence of another one feature.
We have different classes according to the data set given for the experiment and we want to make a prediction, so we want to know if a certain event will happen or not according to the data.
Since we have a data set we divide it randomly, according to the 70%-30% factor, it is often used in Machine Learning: we take the first percentage as the training set and the other part as the test set. The training set represents all the observations we have and provides us a general vision of the probabilities. Thank to the training set we can elaborate our decision seeing the test set, so according on what we learned in the first phase of the procedure we can estimate a possible output for a certain combination of features taken from the test set.
At the end of the implementation of the Nave Bayes Classifier we have also implemented a Laplace additive smoothing in order to perform better results on our decisions.

II. Data Processing

The first task we improved is the so called *Data Processing*. We are given a data set in a written format, as follows:

#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

Fig. 1. Data set given

Since we cannot elaborate literal data we converted them into numerical, according to a legend of only positive numbers: it is not important which value is attributed to each word but it has to be coherent. We chose our legend as follows:

#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		

Fig. 2. Legend to convert the data set into a numerical one

Codifying the data set with the respect legend we obtained a numerical matrix of positive integer numbers:

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

Fig. 3. Numerical matrix obtained converting the given data set

Once finished this step we obtained a matrix and so we were able to work on it with the MATLAB sotfware.

III. Build a naive Bayes classifier

The second task to be solved was the building of the classifier. First of all, before starting with the different phases, we had to check the correct dimensions of both the sets formed, the training set and the test set.
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 first check concerned that the number *c* of columns of the test set matrix is at least the number *d* of columns of the training set matrix decreased by *1*.
Once this first check returned a positive result there were two other tests to be performed:

- all the matrices have to have non negative elements: if only one of the values is negative the program returns an error message;
- all the values contained in the test set need to be already contained in the training set: if there was a value which does not respect this rule, the entire row of the test set matrix has to be removed.

Once both sets were ready for being used we started elaborating the data. First of all we computed the number of classes as the number of different values taken from the last column of the training set matrix, we will use this set in the first part.
Having the number of classes, in our case there were two classes represented by the choice *yes* or *not*, we calculated the probability for each variable and each class: we have four variables: Outlook, Temperature, Humidity and Windy and the two classes as already said.
We used the following formulas to calculate our probabilities:

$$P(x_k = true|t_i) = \frac{numberx_k = trueinclasst_i}{totobservationofclasst_i} = \frac{N_{true,t_i}}{N_{t_i}}$$

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

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

$$P(t_i) = \frac{numberofobservationinclasst_i}{numberofobservationsintehtrainingset} = \frac{N_{t_i}}{N}$$

In our MATLAB software we filled a cell array, appropriately initialised with the correct dimensions: it is a matrix in which every location *(i,j)* is an array (there is also the possibility to have different dimension for each array); then we put the calculated probability in each array location.
In our case the cell array was a *(2,4)* matrix in which, according to the different possible values of each attributes, the corresponding arrays have dimension *2* or *3*.

Once the probability cell array is filled properly we have to compute the output of our decision according on the data processed.
In order to make a decision we had to consider the probabilities calculated. We know that every decision made brings a cost and if it is big it means we took a wrong decision.
We measure this cost with a *loss function* $\lambda(y, \omega)$ where y is the decision taken and ω the state of nature. The role of the loss function is to minimise the risk, which will be a conditional risk because based on the observation made. The risk is calculated as follow:

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

and it is the conditional risk of a decision y_i when we have the experimental observation $\mathbf{x}$.

Of course the risk has to be calculated for all the possible cases so, since in this example we are with discrete and categorical variables, we will compute the general risk as:

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

where $P(\mathbf{x})$ 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 $P(x_1, ..., x_d|t_i) = P(x_1|t_i)P(x_2|t_i) \cdots P(x_x|t_i)$ so the discriminant functions can be computed as:

$$g_i(\mathbf{x}) = P(t_i)[P(x_1|t_i)P(x_2|t_i)\cdots P(x_d|t_i)] = P(t_i)\prod_{j=1}^{d} P(x_j|t_i)$$

In order to minimise the risk we will choose the minimum value for each decision; consequently, in the software, we will substitute if necessary the value already present in our array with one who takes a lower cost.

Once obtained all the results we also calculated the error rate in order to check how much the decision made by the program deviates from the reality.
The error rate is simply computed by calculating the following formula:

$$errorrate = \frac{numberoftimeswrongdecision}{totaldecisiontaken}$$

IV. Improve the classifier with Laplace smoothing

The last task to be performed is the addition of a Laplace smoothing in the classifier. This choice is determined by the presence of many zeros in the discriminant functions due to few data available in a small data set.
For that task we had to calculate again the probabilities, in order to avoid having so many zeros; in particular we have a variable *x* which assumes values in a range *1,2,...,v* with *v* the possible discrete values. The Laplace probability of observing a specific value *i* is:

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

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 also for this second set of results we computed, as before, the error rate and then we compare with the previous one calculated. The comparison pointed out that with Laplace smoothing the error rate can decrease in respect to not using it.

As an example have a look at the following figure:

```
Target no Laplace:
     1     2     1     2

Target with Laplace:
     2     2     1     2

Classification no Laplace:
          0    0.0035
     0.0111    0.0058
          0    0.0210
     0.0222    0.0058

Classification Laplace:
     0.0889    0.0210
     0.1185    0.0315
     0.0222    0.0735
     0.1778    0.0315

Error Rate no Laplace: 0.5
Error Rate Laplace: 0.25
```

Fig. 4. Results taken from a random experiment

The figure points out what we said before. The result of *Classification* represents the *discriminant functions* and we can see that with the use of the Laplace smoothing the zeros can be avoided; moreover the error computed with the use of Laplace smoothing is lower than before so it produces a better result in terms of decisions made: we can see in the *target* that he values calculated differs for the first element in the array, so it means that in that case the decision taken was better.