Machine learning Assignment1

Chiara Saporetti

October 2020

Abstract The purpose of this assignment is to implement a Naive Bayes Classifier capable of working on data sets of different dimensions and internal characteristics. It also tries to solve some inaccuracies of the basic implementation by adding a Laplacian smoothing to the probabilistic model. Results are tested on a given data set.

1 Introduction

The Naive Bayes classifier is a probabilistic classifier based on the Bayes equation and on the the naive assumption that all input variables are independent from each other. The a posteriori probability will then be:

$$P(c|X) \propto \prod P(xi|c)P(c) \tag{1}$$

The code to implement the classifier is written in the MATLAB environment [2] and divided in functions. It is tested on a given "weather" data set. The latter is composed of 14 examples, 4 features, 2 possible classes and at maximum 3 different outcomes for each feature.

2 Task 1: Preprocessing the data set

The first step of the work was to preprocess the given data set by eliminating the names of the features and coding each possible instance of the features with a numerical value. This was done on Microsoft Word through the substitute command.

3 Task 2: Coding

The second step was to prepare code to implement the classifier. The code is structured in a main function that calls multiple functions to compute important data and that subsequently prints the results.

3.1 Main.m

The main loads the data set and saves information about the number of examples/observations (rows of the data set), the number of features (columns of the data set), the number of levels of each feature (possible values of the observations) and the number of classes (possible outcomes of the observations). It also controls that no value in the data set is below 1. It then shuffles the data set rows and randomly separates them into a training subset and a test subset, by previously asking the user how many lines to use for the training part.
It subsequently calls two functions that respectively compute the model of our training set, i.e. the $P(X|c)$ and $P(c)$, and the smoothed model of our training set, that reaches the same goal by although optimizing the logic.
After defining the model of our set, the main calls a function that tests the probabilities computed by the previous one by considering the test set. If the results of the test set are available, the main calls a function that evaluates the accuracy of the predictions. Otherwise, the main just shows the predictions and the test set. Hereafter are presented the functions.

3.2 NaiveModel.m

Input:

- my_set: data set of attributes
- my_res: results of data set
- CLASSES: number of possible results
- LEVELS: levels of each attribute

Output:

- p_class: $P(c)$
- p_feature_class: $P(X|c)$

This function counts the number of the different class instances in the result column of the data set (N_c) and the number of instances of each feature level for class (N_fc). Then the function computes the ratio between N_c over the total number of examples and the the ratio between N_fc over the total number of examples of that class. In formulas:

$$p\_class(c) = \frac{N\_c(c)}{N\_examples} \tag{2}$$

$$p\_feature\_class(f, l, c) = \frac{N\_fc(f, l, c)}{N\_c(c)} \tag{3}$$

where p_feat_class is the likelihood matrix (with dimensions features x levels x classes).

Finally the function checks if the likelihoods of all levels of a feature sum up to 1. The problem with this function is that, if a specific level of a feature doesn't appear in the training set, its likelihood appears to be zero. In order to solve this inaccuracy the formula is slightly modified in Task 3 (see chapter 4.1)

3.3 NaiveClassifier.m

Input:

- test_set: data set of attributes
- p_c: P(c)
- p_feature_class: $P(X|c)$
- N_TEST: number of examples of the test set
- CLASSES: number of possible results

Output:

- p_class_feature: $P(c|X)$

This function implements the Naive Bayes classifier. [1] It computes the $P(c|X)$ for every line of the test, by applying the Bayes equation:

$$P(c|X) = \frac{\prod P(xi|c)P(c)}{P(X)} \tag{4}$$

However, the denominator of the Bayes equation is omitted in the code. The reason is that, since it is the same for all classes and since we only need to compare the values of P for each class, we can just compare the numerators of the fraction. The resulting equation is:

$$P(c|X) \propto \prod P(xi|c)P(c) \tag{5}$$

The problem of applying this kind of equation is that, if probabilities are very low, i.e. tend to zero, the total product goes quickly to zero and may not be right. An attempt to solve this problem may be found at chapter 5.1.

3.4 EvaluateAccuracy.m

Input:

- prob_c: $P(c|X)$
- test_res: results of the test set

Output:

- accuracy: how many guesses were right

This function evaluates the percentage of correct predictions in the results with reference to the real values, and can thus be used only when the results are available. In order to do so, for each example the function takes the the index of the maximum value of $P(c)$ and compares it to the real result. Then it computes the ratio of correct guesses over the total number of tests and gives the result in percentage.

4 Task 3

As previously anticipated, this task optimizes the function described in chapter 3.2. The result is hereby described.

4.1 NaiveModelSmooth.m

Input:

- my_set: data set of attributes
- my_res: results of data set
- CLASSES: number of possible results
- LEVELS: levels of each attributes

Output:

- p_class: P(c)
- p_feature_class: $P(X|c)$

This function is very similar to NaiveModel.m but computes the $P(c|X)$ by applying Laplace smoothing:

$$p\_feat\_class(f, l, c) = \frac{N\_fc(f, l, c) + a}{N\_c(c) + av} \tag{6}$$

This equation tries to solve the problem of some values of a feature not appearing in the data set. It takes into account the idea that all values in a feature are equally probable. So it adds a term a=1 to the numerator, and a x v to the denominator, where v is the number of levels of that feature.

5 Probability with logarithm

As previously anticipated, this function tries to solve the problem highlighted in chapter 3.3. The result is hereby described.

5.1 NaiveClassifierLog.m

Input:

- test_set: data set of attributes
- p_c: P(c)
- p_feature_class: $P(X|c)$
- N_TEST: number of examples of the test set

Output:

- p_class_feature: $P(c|X)$

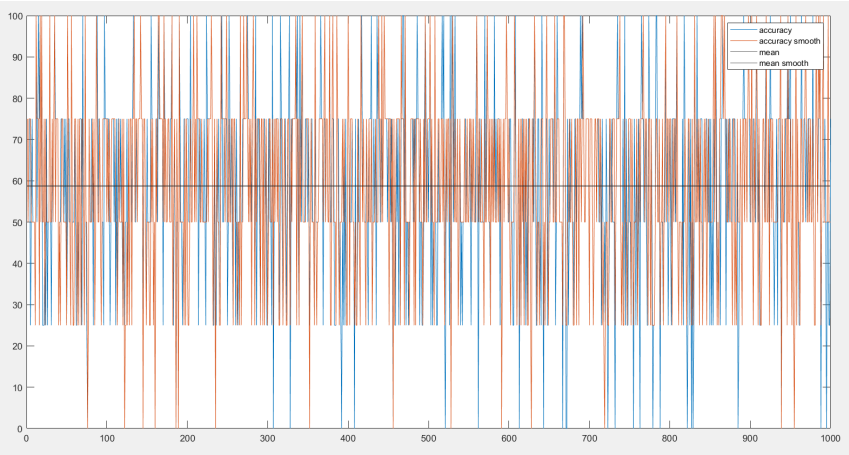
This function is very similar to NaiveClassifier.m but uses a sum of logarithms of the probabilities instead of a product of the probabilities:

$$P(c|X) = \sum log(P(X|c))log(P(c)) \tag{7}$$

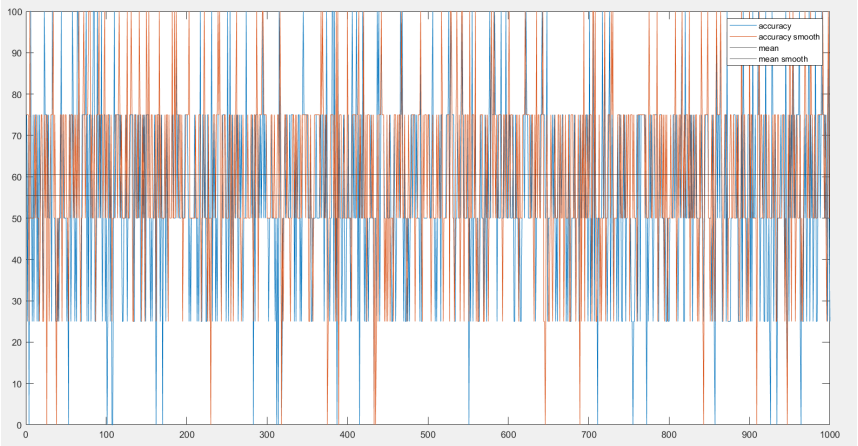
The logarithm keeps the relations between the probabilities while simplifying the calculus.

6 Results

The accuracy of the classifier depends on the data lines that were randomly chosen at the beginning. Since the data set is very small, in order to obtain more meaningful results, the code has been run 1000 times and the mean of the accuracy was calculated for each loop. The final results are:
mean = 59.4500 % and mean_smoothed = 58.9250 %
And the picture below shows the different values of accuracy for all the loops and the two means.



Moreover, the picture below shows the same data by setting all "1" values of the first feature to "2", in order to test the smoothing effect. The final results are:
mean = 55.5500 % and mean_smoothed = 60.5250 %



The smoothing effect is more evident from this second test, since the mean value is increased of about 8%. In general, however, a bigger data set would be needed in order to obtain better results.

References

[1] *https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c.*

[2] *Mathwork documentation.*