Machine Learning Assignment 1 report
The Naive Bayes Classifier

Roberto Albanese 4234506
Università degli Studi di Genova
Robotics Enginireeng

October 22, 2020

***Abstract*—Naive Bayes is a powerful algorithms for classification based on Bayes' Theorem with an assumption of independence among predictors. I was asked to implement its behavior in a MATLAB environment. Further more I have improved it by the addiction of a Laplace smoothing.**

I. INTRODUCTION

1. *Classification*
   In statistics, classification is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known. The classifiers implement a classification algorithm, that maps input data to a category y.
   $$y = \arg\max_{H}[P(\mathbf{H}|\mathbf{X})] \tag{1}$$
   where:
   - $P(\mathbf{H}|\mathbf{X})$ is the a *posteriori probability* of hypothesis H after observing X
   - $\mathbf{H}$ indicates the full set of hypothesis
   - $\mathbf{X}$ indicates the full set of predicates for each observation
2. *Naive Bayes Classifier*
   A Naive Bayes classifier is a probabilistic machine learning model that's used for classification task. The crux of the classifier is based on the Bayes theorem.
   $$P(\mathbf{H}|\mathbf{X}) = \frac{P(\mathbf{X}|\mathbf{H})P(\mathbf{H})}{P(\mathbf{X})} \tag{2}$$
   where:
   - $P(\mathbf{X}|\mathbf{H})$ is the *likelihood probability* of observing X when H holds
   - $P(\mathbf{H})$ is the a *priori probability* that the hypotesis H is true before doing any observation
   - $P(\mathbf{X})$ is the is the *marginal probability* of the set of predictors X

Using Bayes theorem, I can find the probability of H happening, given that X has occurred. Here, X is the evidence and H is the hypothesis. The assumption made here is that the predictors/features are independent. That is that the presence of one particular feature does not affect the other. Hence it is called naive.
The *marginal probability* of X, the probability of observing X in any case, is computed as the following:

$$P(\mathbf{X}|\mathbf{H}) = \sum_{k}^{d} P(\mathbf{X}|H_k)P(H_k) \tag{3}$$

where:

- k indicates the classes (possible outcome of the decision)
- d indicates the attributes

In the case of the naive Bayes classifier I am making the naive assumption that all the features are independent; follows that the *likelihood probability* of observing X when H holds is computed as:

$$\begin{aligned} P(\mathbf{X}|\mathbf{H}) &= P(x_1, x_2, ..., x_n|\mathbf{H}) \\ &= P(x_1|\mathbf{H})P(x_2|H)...P(x_n|\mathbf{H}) \\ &= \prod_{1}^{d} P(x_d|\mathbf{H}) \end{aligned} \quad (4)$$

where:

- d indicates the attributes

The computation of the *priori probability* $P(\mathbf{H})$ is described in the *training phase* section of the document.

II. IMPLEMENTATION

1. *Data processing*
   For the purpose of this experiment, a data set is given in the following way:

   - Data Set Characteristics: Multivariate, supervised
   - Number of Instances (observations): 14
   - Attribute Characteristics: Categorical
   - Number of Attributes: 4
   - Associated Task: Classification
   - Number of classes: 2

   Data Description:

   - The first line is the column names
   - Columns 1-4 are the input features
   - Column 5 is the target class

   Attributes:

   - Outlook (overcast, rainy, sunny)(1,2,3)
   - Temperature (hot, cool, mild)(1,2,3)
   - Humidity (high, normal)(1,2)
   - Windy (TRUE, FALSE)(1,2) Classes:
   - Play(yes,no)(1,2)

   The data file is found within the zip folder under the name '*Data.txt*' and The first step of the project was to create a MATLAB function ('*BuildSets.m*') to upload data from a file and divide them into *training set* and *test set*. Data should be given in an integer form with different positive values for each level of each attribute. The *training set* will be composed of 10 random observation and a *target set* of data while the *test set* will be composed of the 4 remaining observation. Some additional check are implemented, as the file path existence check and the positive value check.

2. *Training phase*
   The second task was to implement a training phase ('*TrainBayesClassifier.m*') for the classifier using the *training set*. In this phase I compute *likelihood probability* using (4) and the *a priori probability* which will let us to compute the Bayes theorem.

   Since data in our problem are discrete, *likelihood probability* for each level of all the attributes and the *a priori probability* are computed in the following way:

   $$P(x_i = l|\omega_k) = \frac{\text{times that } x_i = l \text{ in class } \omega_k}{\text{observation of class } \omega_k} \quad (5)$$

   where:

   - $P(x_i = l|k)$ indicates the likelihood that the attribute x_i will be observed with the level l in the class ω_k
   - i indicates the attributes
   - l indicates the levels for each attribute
   - k indicates the classes

   $$P(\omega_k) = \frac{\text{observation of class } \omega_k}{\text{observation in the training set}} \quad (6)$$

   where:

   - k indicates the classes

3. *Testing phase*
   Once the Naive Bayes classifier has been trained

it can accept the *testing set* and classify all the observation by computing the Bayes theorem. Two *posteriori probability* are computed ('*BayesClassifier.m*') and the higher one is chosen to complete the classification algorithm.

4. *Validation*
The *testing set* is a file composed, as well as the *training set*, of as many rows as the observation and as many column as the attribute plus one more last prediction column for the class which can be used to validate the classifier. Once the classifier finish the classification an error is computed to check the validity in the following way:

$$error = \frac{\sum_{k=1}^{o} \text{diff}(\omega_k \neq \hat{\omega}_k)}{N} \tag{7}$$

where:

- o indicates the observations
- k indicates the classes
- $\hat{\omega}_k$ is the prediction class for observation o
- *diff()* is an operator which returns 1 if elements are different and 0 otherwise

5. *Laplace smoothing*
Some issues can occur using a small *training set* for the training phase since some combinations that appear in the *test set* could not be encountered in *the training set*, so their probability would be assumed to be zero. Multiplying many terms, just one zero will make the overall result be zero. To deal with this case ('*TrainBayesClassifierLaplaceSmoothing.m*') an additive smoothing prior information is implemented in the computation of the *likelihood probability* of each attribute editing (5) in the following way:

$$P(x_i = l|\omega_k) = \frac{n_{i,k} + a}{N_{i,k} + av} \tag{8}$$

where:

- $P(x_i = l|k)$ indicates the likelihood that the attribute x_i will be observed with the level l in the class ω_k
- i indicates the attributes
- l indicates the levels for each attribute
- k indicates the classes
- a indicates the trust in the data (higher the value, lower the trust)
- v indicates the number of levels of attribute x_i

This leads us to a more balanced formula which can compensate absent combination between attributes level and classes in the *the training set.*

Further more I have also implement a function ('*SetClassifierMode.m*') to let the user decide either to run the code with or without the Laplace smoothing.

III. CONCLUSION

To evaluate the results I have run the script 1000 times in order to test my functions with diferent training and test sets. The resultant average error rates are:

- For the classifier without the smoothing: 0.3640
- For the classifier with the smoothing: 0.4375

The Naive Bayes classifier is a good starting point for learning the basics of Machine Learning. Not only it's easy to implement, but also the naivety assumption is often approximately correct. Nevertheless, it still has some flaws that can be improved with simple adjustments, as it has been done in the third task, it is possible to overcome the eventuality of missing attribute values with data processing and Laplace smoothing. In the case of this assignment the code has shown that the classifier is less effective with a small set of data, but it can be improved with either a smoothing algorithm or a bigger set of data.