MACHINE LEARNING FOR ROBOTICS I - 86928

Lab 1: Naive Bayes classifier

- Task 1: Data preprocessing
- Task 2: Build a naive Bayes classifier
- Task 3: Improve the classifier with Laplace (additive) smoothing
- Think further

Task 1: Data preprocessing

Here you have to download data and prepare them for being used with a Matlab program.

Download the Weather data set and the Weather data description.

Remark: For use with Matlab it is convenient to convert all attribute values into integers>=1. So they can be used to index matrices. You can do so by either using a plain text editor, or reading the data into a spreadsheet table and then saving it back (remember to "save as" a plain text file or at most a .csv file, not a spreadsheet file).

Remark: The data set is so small that you can do the calculations by hand to compare with your code, and check if it is correct.

Remark: In statistical jargon, nominal or categorical variables don't have "values" but "levels".

Task 2: Build a naive Bayes classifier

Here you have to create a program (a Matlab function for instance) that takes the following parameters:

1. a set of data, as a *n* x (*d*+1) matrix, to be used as the training set
2. another set of data, as a *m* x *c* matrix, to be used as the test set

The program should:

1. Check that the number *c* of columns of the second matrix is at least the number *d* of columns of the first matrix - 1
2. Check that no entry in any of the two data sets is <1
3. Train a Naive Bayes classifier on the training set (first input argument), using its last column as the target
4. Classify the test set according to the inferred rule, and return the classification obtained
5. If the test set has a column number d+1, use this as a target, compute and return the error rate obtained (number of errors / *m*)

Write a script that, **without the user's intervention**:

1. loads the weather data set (already converted in numeric form)
2. splits it into a training set with 10 randomly chosen patterns and a test set with the remaining 4 patterns
3. calls the naive Bayes classifier and reads the results
4. prints the results: classification for each pattern of the test set, corresponding target if present, error rate if computed.

Hint: From the slides, you see that probability = frequency = number/totalnumber.

What may not be immediately clear from the formulas (because it is implicit in the definition of a probability mass function) is that for each variable you have to compute a probability for each possible value. This is the probability mass function that is used in the formulae. It gives the probability for each value (conditioned to the class, i.e. a likelihood; but this does not matter for the calculations).

Explicitly in pseudocode, what you have to compute (and store in a matrix or in several matrices) is:

```
for each class c
  for each variable x
    for each possible value v for variable x
        the number of instances of class c that have variable x == value v...
        ...divided by the number of instances of class c
```

and this will be our estimate of the likelihood *P*(*x*=*v* | *c*)

Remarks: The classifier should be programmed in such a way to be suitable for the following situations:

- other data sets of different cardinalities (*n*, *m*) and dimensionality (*d*)
- test sets not including a target
- test sets including attribute values that **were not in the training set**.

In the last case the program should issue an error and discard the corresponding pattern, because if you use that value to index a matrix it will likely go out of bounds (e.g., if you computed probabilities for values 1, 2 and 3 and you get value 4 in the test).

Hint: Use the slides (ML 2) to implement the classifier. Note that all attributes are categorical, but some are non-binary having more than 2 levels.

Hint: If you are not strong in programming, don't fear: it is simpler than it seems. Programs, as well as their specifications and requirements, are easier to write than to read.

In any case it is advisable to start with a core that does just part of what is required, for instance just load a data set and compute probabilities; and then add functionalities gradually in more iterations. (This even has a name: agile software development, a type of continual improvement process.) A good practice in this multiple iteration approach is to save each version of the program so as to be able to go back to a working one if you do something wrong.

Task 3: Improve the classifier with Laplace (additive) smoothing

You will observe that, with such a small data set, some combinations that appear in the test set were not encountered in the training set, so their probability is assumed to be zero. When you multiply many terms, just one zero will make the overall result be zero.

In general, it may be the case that some values of some attribute do not appear in the training set, however **you know** the number of levels in advance.

An example is binary attributes (e.g., true/false, present/absent...). You know that attribute x can be either true or false, but in your training set you only have observations with x = false. **This does not mean that the probability of x = true is zero!!!**

To deal with this case, you should introduce a kind of prior information, which is called **additive smoothing or Laplace smoothing**.

Laplace smoothing -- Suppose you have random variable *x* taking values in { 1, 2, ..., *v* } (*v* possible discrete values). What is the probability of observing a specific value *i* ?

We perform *N* experiments and value *i* occurs n_i times. Then:

- Without smoothing (simple frequency) the probability of observing value *i* is given by $P(x=i) = \frac{n_i}{N}$
- With Laplace smoothing, the probability of observing value *i* is given by $P(x=i) = \frac{n_i + a}{N + av}$ where *a* is a parameter (see below)

You have to change your code in 2 ways:

1) In the data preparation step, add the information about the number of levels. This means that for each data column you should add the number of possible different values for that column.

In the case of the weather data, this can be a list (or vector) [3, 3, 2, 2]. You can add this list as a first row in all the data sets (training and test), to be interpreted with this special meaning.

2) When you compute probabilities, you introduce Laplace smoothing in your formulas by adding some terms that take into account your prior belief. Since **you don't know anything**, your prior belief is that all values are **equally probable**. In this case, Laplace smoothing gives probability = 1/2 (for binary variables) or more generally probability = 1/v (for variables with v possible values). This is why the number of possible values must be known (modification 1).

In formulae, you should replace:

```
P(attribute x = value y | class z) =
   (number of observations in class z where attribute x has value y) / (number of observations in class z)
```

with the following:

```
P(attribute x = value y | class z) =
   ((number of observations in class z where attribute x has value y) +a) / ((number of observations in class z) +av)
```

where:

- v is the number of values of attribute x, as above;
- you can use a = 1. As a further refinement, you can experiment with a > 1 (which means "I trust my prior belief more than the data") or with a < 1 (which means "I trust my prior belief less than the data")

Think further

- If you have large data sets, you may incur in numerical errors while multiplying a lot of frequencies together. A solution is to work with log probabilities, by transforming logarithmically all probabilities and turning all multiplications into sums.
- How would you proceed if the input variables were continuous? (Hint: variable values are used to compute probabilities by counting, but theory tells us that probabilities may be obtained by probability mass functions directly if they are known analytically.)
- You can experiment with the continuous variable case using the Iris data set (with its description). The four features can be made binary by (1) computing the average of each and (2) replacing each individual value with True or 1 if it is above the mean and False or 0 if it is below. You can use the median in place of the average.

Submission status

Attempt number	This is attempt 1.
Submission status	Submitted for grading
Grading status	Not graded
Due date	Thursday, 20 October 2022, 1:40 PM
Time remaining	84 days 14 hours
Last modified	Thursday, 7 July 2022, 1:05 PM
File submissions	Machinelearnig_Lab1_S4845876.zip 7 July 2022, 1:05 PM