Instituto Superior Técnico

Departamento de Engenharia Electrotécnica e de Computadores

Machine Learning

4^{th} Lab Assignment

Shift Thursday Group number 5

Number 80966 Name Francisco Azevedo

Number 81356 Name Duarte Dias

Naive Bayes classifiers

1 Naive Bayes Classifier

Naive Bayes classifiers normally are rather simple, and are very effective in many practical situations. Describe in your own words how the Bayes classifier works. Be precise. Use equations when appropriate.

The Bayes classifier is a solution to a classification problem where you have x features and want to know which class ω_i they belong. It is based on the calculation of the a posteriori distribution of the classes ω_i by using the Bayes law. From a train dataset we can obtain P(ω_i) and P(x|ω_i) and P(x). By applying Bayes law we obtain:

$$P(\omega_i|x) = \frac{p(x|\omega_i)P(\omega_i)}{p(x)} \tag{1}$$

By calculating this probability, we can predict what class the element belongs to based on its features. If we associate a loss function L(y,ω_i) to penalize a difference between the target (y) and predicted class (ŷ =w) with the probability P(y|x) we obtain the cost function:

$$c_\omega(x) = \sum_{y \in \Omega} L(y, \omega) P(y|x) \tag{2}$$

This is an optimal classifier in the sense that it minimizes the risk of a general loss function. This is very easy to implement for a problem that only has 1 feature but it becomes quite difficult when we have many features since it is harder to calculate the conditional distribution P(x |ω_i). This is where the Naïve Bayes classifier comes in, it simplifies the problem by assuming that features are independent from each other which allows us to do simplify the previous calculation:

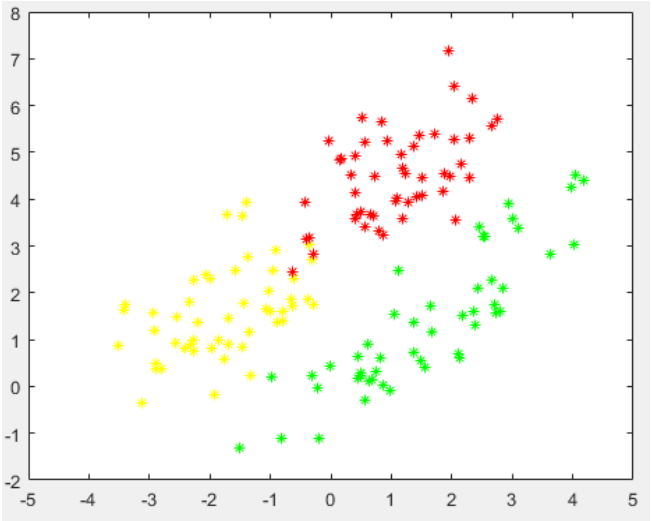
$$p(x_1, \ldots, x_p|\omega_k) = \prod_{i=1}^{p} p(x_i|x_1, \ldots, x_{i-1}, \omega_k) = \prod_{i=1}^{p} p(x_i|\omega_k) \tag{3}$$

2 A simple example

In this part of the assignment, you'll make a naive Bayes classifier for a very simple set of data. The input data are two-dimensional, and belong to one of three classes. To load the data, load the file `data1.mat`. The data have already been split into training data (variables `xtrain` and `ytrain`) and test data (variables `xtest` and `ytest`).

1. Visualize a scatter plot of the training and test data, using different colors, or symbols, for the different classes. Don't forget to use equal scales for both axes, so that the

scatter plot is not distorted. Draw a sketch of the scatter plot of the training data.



Green -> Class1
Yellow -> Class2
Red -> Class 3

2. Make a Matlab script that creates a naive Bayes classifier based on the training data, and that finds the classifications that that classifier gives to the test data. The script should plot the classifications of the test data as a function of the test pattern index, and should print the percentage of errors that the classifier makes on the test data. Write your own code, do not use any Matlab ready made function for Naive Bayes classfication.

   Give the listing of your script in a separate file. The script should have enough comments to allow the reader to understand how it works (normally, this will correspond to less than one comment per line). You don't need to make a very general script: you can make as simple a script as you wish, as long as it does what is requested.

   Suggestion: You will need to estimate probability densities of certain sets of data, to solve this item. For the estimation of each density, use a Gaussian distribution. Estimate its mean and variance, respectively, as the mean and variance of the corresponding set of data (for the variance estimates, divide by N, and not by $N-1$, where N is the number of data points). The estimator that you'll obtain in this way is the maximum-likelihood estimator of the Gaussian distribution.

3. Indicate the percentage of errors that you obtained in the test set.

   Test set error rate: 2,67%

4. Comment on your results.

   The Naive Bayes classifier yields a low test set error rate. Given that the data set is small (10^2 magnitude) this result could further be improved with larger sets of data. We can see through the plot of the data classified by us that there are a few points miss-classified. This occurs on the edge of the clusters belonging to class 2 and 3 which is understandable.

The classification problem that you have just solved is very small, and was specially prepared to illustrate the basic working of naive Bayes classifiers. You should be aware,

however, that the real-life situations in which these classifiers are normally most useful are rather different from this one: they are situations in which the data to be classified have a large number of features and each feature gives some information on which is the correct class. Normally, for each individual feature, there is a significant probability of giving a wrong indication. However, with a large number of features, the probability of many of them being simultaneously wrong is very low, and, because of that, the naive Bayes classifier gives a reliable classification. The second part of this assignment addresses such a situation.

3 Language recognizer

One of the applications in which naive Bayes classifiers give good results and are relatively simple to implement, is language recognition. In the second part of this assignment, you will make some of the code of a naive Bayes language recognizer, and you will then test the recognizer. The training data are provided to you. Most of the code of the recognizer is also provided, but the parts that specifically concern the classifier's computations are missing. You will be asked to provide them. After that, you will be asked to test the recognizer.

3.1 Software and data

The Matlab code for the recognizer is given in the file `languagerecognizer.m`. This code is incomplete, and should be completed by you as indicated ahead. The code consists of two parts, which are clearly identified by comments:

- The *first part* reads the trigram counts of the training data for the various languages, from files that are supplied. The names of these files are of the form `xx_trigram_count_filtered.tsv`, where `xx` is a two-character code identifying the language that the file refers to (`pt` for Portuguese, `es` for Spanish, `fr` for French, and `en` for English).

  The aforementioned files contain the data of one trigram per line: each line contains the trigram, followed by the number of times that that trigram occurred in the corresponding language's training data. Before counting the trigrams in the training data, all upper case characters were converted to lower case. The set of characters that was considered was {abcdefghijklmnopqrstuvwxyzáéíóúàèìòùâêîôûäëïöüãõñ .,:;!¡?¿-'} (note that there is a blank character in the set). Trigrams containing characters outside that set were discarded. You may want to look into the trigram count files to have an idea of what are their contents, or to check the numbers of occurrences of some specific trigrams.

  After executing the *first part* of the code, the following variables are available:

  - `languages`: Cell array that stores the two-character codes for the languages. For

example, `languages{4}` contains the string `'en'`. Note that the argument is between braces, not parentheses.

- `total_counts`: Array that contains the total number of trigrams that occurred in the training data, for each language. For example, `total_counts(4)` contains the total number of trigrams that occurred in the training data for English. Trigrams that occurred repeatedly are counted multiple times.

The *first part* of the code is complete: you shouldn't add any code to it.

- The *second part* of the code consists, basically, of a loop that repeatedly asks for a line of input text and then classifies it. Each iteration of the loop performs the following operations:

  - Ask for a line of input text and read it.
  - Check whether the input text contains only the word `quit`. If so, exit the loop (this will end the program).
  - Convert all the input text to lower case.
  - Perform a loop on the languages. Within this loop, perform a loop on all the trigrams of the input text.
  - Print the scores of the various languages, the recognized language and the classification margin.

  This description of the operations performed by the *second part* of the code may sound somewhat incomplete, because this part of the code actually is incomplete. You should complete it by adding code, as described below.

The places where you may need to add code are clearly marked, with comments, in the file `languagerecognizer.m`. Those places are identified, in the comments, as Code Sections 1, 2 and 3. You will need to use those identifications later on.

The code that is provided already contains all the loops that are needed, as well as a few more commands. You will need to add the code that performs the calculations for the recognizer itself, using the data produced by the *first part* of the program (described above), as well as some data that are computed by already existing code of the *second part* of the program. Take into account the following indications:

- The basic structure of the *second part* is as follows:

  - There is an outermost loop, which repeatedly asks for input text and then proceeds to classify it.
  - That loop contains a loop on the languages.

  - The loop on the languages contains a loop on all the trigrams of the input text. In the beginning of this loop, the trigram that is to be processed in the current iteration is placed in the variable `trigram`, and the number of occurrences of that trigram in the training data for the current language is placed in the variable `trigramcount`.

- In Code Section 3, the final results of the calculations that you perform should be placed in an array called `scores`, of size 4, with an element for each language. For example, `scores(4)` should contain the score for English. The scores should be computed so that a higher score corresponds to a language that is more likely to be the one in which the input text was written.

- The end of the *second part* of the code already contains the instructions that will find the language with the highest score and output the results. The program outputs the scores of the various languages, followed by the identification of the language that has the highest score, and by the *classification margin*, which is the difference between the two highest scores.

3.1.1 Practical assignment

1. Complete the code given in the file `languagerecognizer.m`. Transcribe here the code that you have added to the program. Clearly separate and identify Sections 1, 2 and 3 of the added code.

```
This is code Section 1:

scores(languageindex) = 0;

This is code Section 2:

scores(languageindex) = scores(languageindex) + trigramcount;

This is code section 3:

scores(languageindex) = scores(languageindex)/total_counts(languageindex);
```

2. Once you have completed the code and verified that the recognizer is operating properly, complete the table given below, by writing down the results that you obtained for the pieces of text that are given in the first column.

   The last piece of text is intended to check whether your recognizer is able to properly classify relatively long pieces of text. It is formed by the sentence "I go to the beach. " repeated ten times (in the table, the piece of text is abbreviated). Note that the given sentence has a blank space after the period, so that the repeated sentences are grammatically correct. You may use copy and paste operations to ease the input of this piece of text.

Text	Real language	Recognized language	Score	Classification margin
O curso dura cinco anos.	pt	pt	0.02326	0.00363
El mercado está muy lejos.	es	es	0.02853	0.00677
Eu vou à loja.	pt	fr	0.005844	0.00185
The word é is very short.	en	en	0.038797	0.02918
I go to the beach. ... I go to the beach.	en	en	0.59448	0.5053

3. Give a detailed comment on the results that you have obtained for each sentence.

The classifier is correct on the first two sentences which is to be expected because the sentences have words that differ significantly from the other languages. However the 2nd sentence margin is double of the 1st one.

The 3rd sentence is missclassified by a small margin. A first look yields a surprising result given that the respective translation in french is "Je vais au magasin". However this is not how our classifier judges sentences. The classifier pays attention to trigrams, in this case, both trigrams are likely present in either French or Portuguese. This exposes a major flaw in the classifier: due to only paying attention to trigrams it looses all context of words/sentences and is easily fooled as seen above.

The 4th sentence is classified as English, with a significant margin, despite having a non english trigram "é". This reveals that the classification model is robust to "noise", or small terms of different languages. This is only possible if the rest of the sentence is written in English.

The 5th sentence prooves that the classifier works well for larger sentences, resulting in a larger classification margin.