Instituto Superior Técnico

Departamento de Engenharia Electrotécnica e de Computadores

Machine Learning

4^{th} Lab Assignment

Shift *Sexta,14h* Group number *1*

Number *84053* Name *Francisco Raposo de Melo*

Number *89213* Name *Rodrigo Tavares Rego*

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 Naive Bayes classifier works. Be precise. Use equations when appropriate.

Classifying a new observation into the appropriate class can be done using Bayes Classifier, which minimizes the probability of misclassification, by computing an estimation of conditional probability.

Bayes Classifier: $\hat{Y} = \underset{W \in \Omega}{\operatorname{argmax}} \quad P(W|X)$

Given a feature vector x = [x1, ..., xp] containing many features, the computation of the estimation of the conditional distribution turns out to be difficult. To simplify the problem we can use the Naive Bayes Classifier.

The Naive Bayes Classifier assumes that the features are conditionally independent, which means that it is only required to compute the estimation of the conditional distributions of each feature (simpler problem). The appropriate class corresponds to the one with highest estimated conditional probability.

Naive Bayes Classifier:

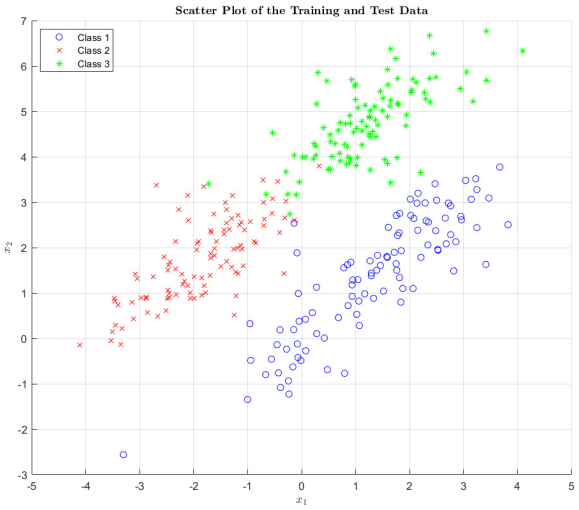
$$p(x_1, \ldots, x_p|\omega_k) = \prod_{i=1}^{p} p(x_i|x_1, \ldots, x_{i-1}, \omega_k) = \prod_{i=1}^{p} p(x_i|\omega_k)$$

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. Load the file `data1.mat` to get the data, which have already been split into training data (variables `xtrain` and `ytrain`) and test data (variables `xtest` and `ytest`).

1. Obtain 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.

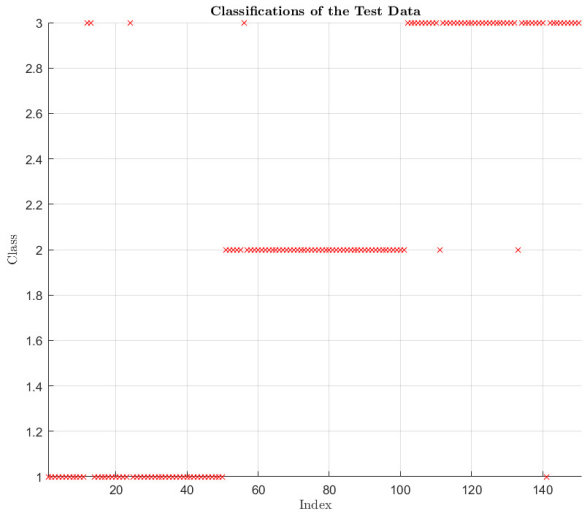


2. Make a Matlab script that creates a naive Bayes classifier based on the training data, and that finds the classifications that that classifier gives to the test data. The script should plot the classifications of the test data as a function of the test pattern index, and should print the percentage 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 classification.

   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. Plot the classifications of the test data as a function of the test pattern index.



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

   Test set error rate: ***5,33%***

5. Comment on your results.

Observing the figure in question 1 we can see that for all classes both features are positively correlated (cloud points are inclined), and that the variance of class 1 is higher in both dimensions, though it has the mean value set more apart. From this observation we can deduce that the independence assumption, used with Naive Bayes Classifier, between the estimation of the conditional distributions of both features, will imply a higher error (the features are in fact positively correlated).

As we can observe from the figure in question 3, there was a higher classification error in points from class 3 and 1, and class 3 and 2. This happens because class 1 has a higher variance in both dimensions, having more points mixed between the other cloud points, and because the mean values of class 2 and 3 are relatively close.

For those reasons the obtained error was expected. To control our conclusions we used Bayes Classifier in order to check how the correlated features with Naive Bayes Classifier impacts the value of the error, having obtained an error of 3,33% with Bayes Classifier, with no misclassification for class 1 with this test set (we also sent this code - Bayes_ Classifier.m).

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. Include comments.

```
%% SECTION 1
% Additive smoothing (Laplce Smoothing) requires that we add 1
% to all triagram combinations, since the set of characters
% used is of length 60, the sum of ones of all combinations
% equals 60^3 (used to update the total_count for each language)

v_total_counts(languageindex)=0;
v_total_counts(languageindex)=total_counts(languageindex)+60^3;

clear v_trigramcount;

%% SECTION 2
% We also update de trigram count, by adding 1, as a result
% of using additive smoothing

v_trigramcount(trigramindex) = log(trigramcount + 1);

%% SECTION 3
scores(languageindex) = sum(v_trigramcount)-(length(v_trigramcount) * ...
                                log(v_total_counts(languageindex)));

% We applied the log properties in the computation of the probability
% beforehand to avoid over and underflows.
```

The use of logarithmic scale is reasonable, because it conserves the relative order of points, since the log function is strictly monotone.

$$log[\hat{P}(X|C_j)] = log[\frac{1}{(Total\_C_j + 60^3)^N} \prod_{i=1}^{N}[(n_i, C_j) + 1]]$$

$$\sum_{i=1}^{N} log[(n_i, C_j) + 1] - [N \cdot log(Total\_C_j + 60^3)]$$

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*	*-161,9883*	*1,3628*
El mercado está muy lejos.	es	*es*	*-183,7106*	*18,5874*
Tu vais à loja.	pt	*fr*	*-114,8892*	*8,0609*
The word é is very short.	en	*en*	*-192,8834*	*3,3290*
I go to the beach. ... I go to the beach.	en	*en*	*-1313,7568*	*262,6609*

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

1st Sentence: *correctly classified, however the classification margin was small, meaning that the obtained highest score was close to the score for another language (es), because the text was small and some of the trigrams from this text occur also frequently in spanish ("cur" occurs 1,7 more times and "os." occurs 21,4 more times in es than in pt).*
2nd Sentence: *correctly classified with high classification margin, as a result of having a sufficiently high number of trigrams particular and frequent in spanish.*
3rd Sentence: *wrongly classifed, with a reasonable classification margin, due to the existence of a sufficiently high number of trigrams frequent not only in portuguese, but also in french and more frequent in the latter (for example, "tu " occurs 10,5 more times, "vai" occurs 16,3 more times and " à " occurs 23,7 more times in french than in portuguese).*

4th Sentence: *correctly classified, with a low classification margin, due to the existence of a very particular trigram to other languages that use more words with accents. The text is written in two mixed languages, but still was correctly classified.*
5th Sentence: *correctly classified, with a very high classification margin, due to the use of a long text written only in english. There is a high number of trigrams in the text frequently used in this language, lowering notoriously the probability of misclassification.*