Instituto Superior Técnico

**Departamento de Engenharia Electrotécnica e de Computadores**

# Machine Learning

$4^{th}$ Lab Assignment

Shift Wed - 14h Group number 1

Number 78486 Name Luís Bernardo de Brito Mendes Rei

Number 78761 Name João Miguel Limpo de Lacerda Pestana Girão

**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 theorem works on conditional probability - the probability that something will happen, given that something else has already occurred. Suppose we seek to estimate the class of an observation given a vector of features. Denoting class C and the vector of features $(F_1, \ldots, F_k)$, given a probability model underlying the data (that is, given the joint distribution of $(C, F_1, \ldots, F_k)$), the Bayes classification function chooses a class by maximizing the probability of the class given the observed features P(C = c | $F_1 = f_1, ..., F_k = f_k$).

In practice the quantity $P(C = c|F_1 = f_1, ..., F_k = f_k)$ is very difficult to compute. We can make it a bit easier by applying Bayes' theorem and ignoring the resulting denominator, which is a constant. Then we have

$$P(C = c)P(F_1 = f_1, ..., F_k = f_k|C = c),$$

but this is often still intractable: lots of observations are required to estimate these conditional distributions, and this gets worse as **k** increases (high dimensionality). And what if a particular combination of features is never observed in the data?

The Naive Bayes classifier predicts membership probabilities for each class such as the probability that a given record or data point belongs to a particular class. The class with the highest probability is considered as the most likely class (maximum a posteriori). Naive Bayes classifier assumes that all the features are unrelated to each other

$$P(F_1 = f_1, ..., F_k = f_k|C = c) \simeq \prod_{i=1}^{k} P(F_i = f_i|C = c).$$

Presence or absence of a feature does not influence the presence or absence of any other feature. Substituting this into the Bayes classifier yields the naive Bayes classifier

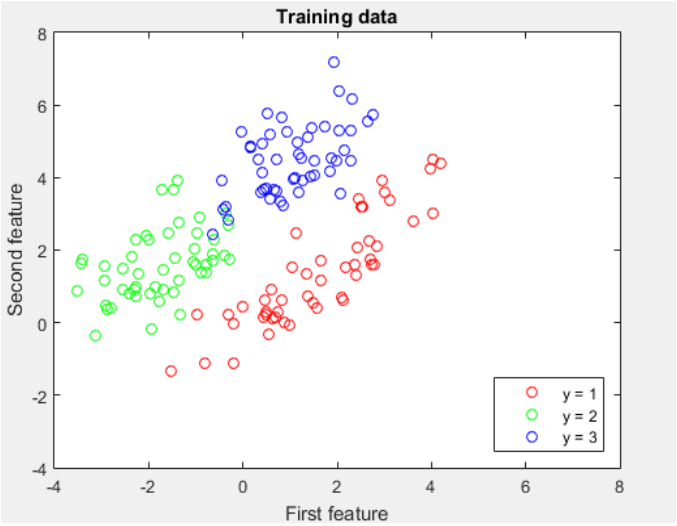
$$argmax_C \quad P(C = c) \prod_{i=1}^{k} P(F_i = f_i|C = c).$$

# 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.




In figure 1 and 2 in annex, we observe the scatter plot of the test data and the scatter plot of the training and test data combined, respectively.

In the test set, the distribution of the class points resembles narrower elipses than those featured in the training set, and the points are such that the gaussian function determined using the training set fits the distribution almost perfectly.

This means that either the test set had few outliers, or the training set was very good, both of which do not tend to happen in the real world.

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: 4.6667% (7 in 150)

4. Comment on your results.

| Class | First | Second | Third |
|---|---|---|---|
| Error | 10% (5 in 50) | 4% (2 in 50) | 0% (0 in 50) |

* In table 1 and 2 in annex you will find tables comparing the bayes and naive bayes classifier performances.

After fitting the Gaussian functions to the class distribution, we can observe (figures 3 to 5 in annex) three distinct regions of classification that correspond to the different classes. The regions for classes 2 and 3 have a high density over a smaller area around their respective means, while the densest region for class 1 spans over a wider area (approximately two times the area of the other two) but with much lower peak value and probability density function (pdf) value variation. As a result, the points that are in the overlapped sections (with relevant pdf values) of the different Gaussians will be classified as belonging to the one with the biggest pdf values, making the classification error-prone to values closer to the border of said classes.

Applying a Bayes classifier without the naive approximation provides a test set error rate of mere 1.3333% (2 in 150).

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.

```
% ************************************************************************************************************
% Section 1
possibletrigrams = numel(text)-2; % Number of possible trigrams in the input text

trigramprobability = zeros(1, possibletrigrams); % Probability of the trigram appearing given the language being checked

languageprobability(languageindex) = total_counts(languageindex)/sum(total_counts(:)); % Probability of current language
% ************************************************************************************************************


% ****************************************************************
% Section 2
trigramprobability(trigramindex) = log((trigramcount + 1)/(total_counts(languageindex)+ 60^3));% Update trigram probability
% ****************************************************************
```

```
% ********************************************************************************************************
% Section 3

scorekeeper = 1; % Initialize variable

% Compute probability of language given sentence
for index = 1:possibletrigrams      % For all probabilities do:
    scorekeeper = scorekeeper + trigramprobability(index);  % Add to old probability
end

scores(languageindex) = scorekeeper + log(languageprobability(languageindex)); % Final probability
% ********************************************************************************************************
```

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 | -162.73802 | 0.66067279 |
| El mercado está muy lejos. | es | es | -183.75821 | 19.289505 |
| Eu vou à loja. | pt | fr | -110.94491 | 3.6248262 |
| The word é is very short. | en | en | -196.64658 | 0.3155173 |
| I go to the beach. ... I go to the beach. | en | en | -1317.52 | 259.64745 |

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

First of all, let's have a look at how the score of a sentence is computed. Sentences are split into different sections, each with three characters starting with the first, second and third characters, followed by the second, third and fourth characters, and so on and so forth. Since we aren't looking at the words that are part of the phrase, we can't say that a piece of the text belongs to a certain language for sure. Hence we look at a piece of the sentence at a time and check the probability of it belonging to a certain language, then we multiply all the probabilities and select the one with bigger value (in this case we use the sum of the logarithm due to the size of the probabilities). In this exercise we assumed all the languages had the different prior probabilities and we calculated them as the number of trigrams from one language over the number of trigrams of all the languages. This produces good results if the language options are not similar or if there are some key trigrams that are classified as being part of a certain vocabulary. Note that the program needs a vast amount of examples to function properly, as it needs a good estimation of the trigrams' probabilities.

Now that we understand how this works we can understand the example phrases and their scores: The first one is correctly classified but the similarities with the Spanish language make it so that the difference in scores isn't that big. The second sentence has quite a big classification margin likely due to the key trigrams mentioned earlier, despite the closeness to the other Iberic dialect. The third phrase is the only misclassification on the set, and it happens to occur on the shortest sentence tested. The small panoply of trigrams leads to an uncertainty on the classification due to the lack of variety in the chosen letters. The prior probability of the sentence being french is much larger than all of the others (table 3 in annex), so for smaller sentences the tendency to classify one as french is bigger. The fourth sentence, despite not being misclassified, has a low classification margin, although some Portuguese trigrams do appear and weigh in considerably for the pt classification. Lastly, we are presented with the largest margin on the test set, which is a good example of a sentence that has by itself a fair certainty of the language it belongs to. This margin was amplified as the phrases that constitute it repeat, again a depiction of the amount of confidence in the original classification.

In sum, small sentences imply low variety of trigrams, which trigger small scores and classification margins. Bigger phrases tend to work better as the small differences accumulate and are propagated, allowing for much larger values.

# ANNEX

Table 1 – Test set error rate with naive bayes classifier

| Class | First | Second | Third |
|---|---|---|---|
| Error | 10% (5 in 50) | 4% (2 in 50) | 0% (0 in 50) |

Table 2 – Test set error rate with bayes classifier

| Class | First | Second | Third |
|---|---|---|---|
| Error | 0% (0 in 50) | 4% (2 in 50) | 0% (0 in 50) |

Table 3 - Prior probability of each language

| Language | PT | ES | FR | EN |
|---|---|---|---|---|
| Probability (%)* | 17.83% | 35.08% | 46.69% | 00.85% |

* The sum of all probabilities is bigger than 1 due to some roundings

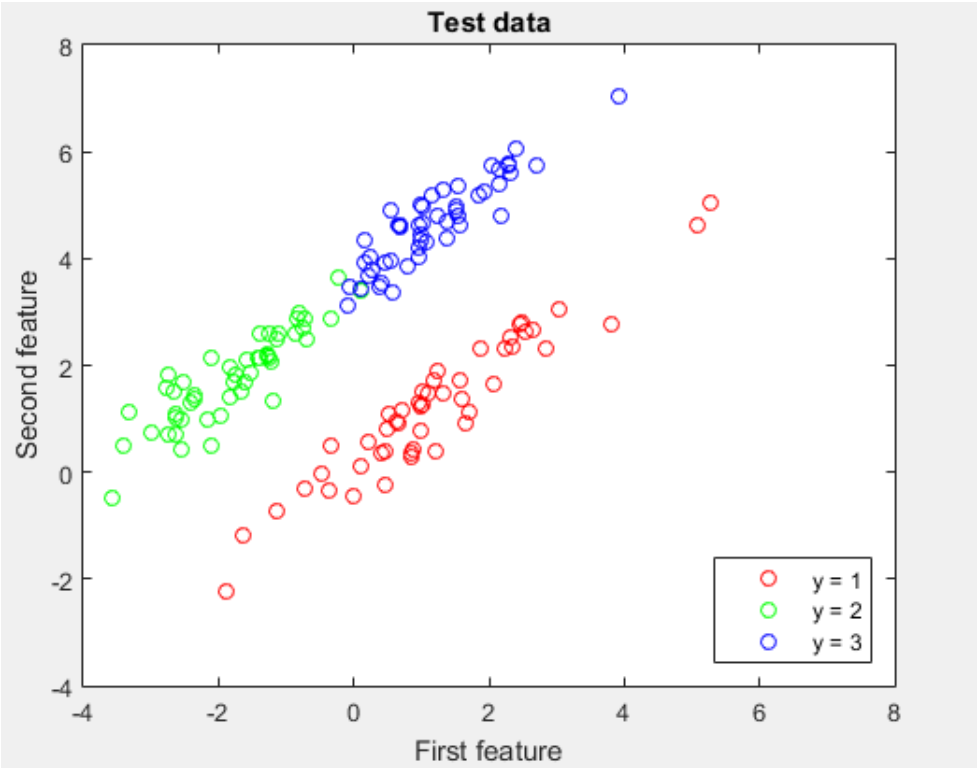



Figure 1 – Scatter plot of the test data.

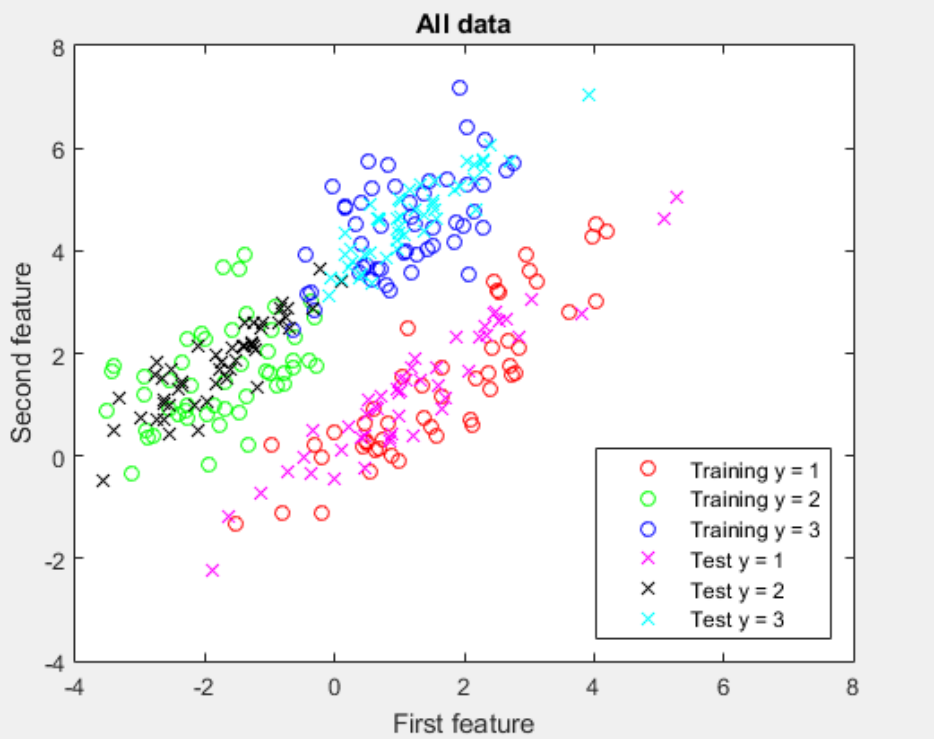



Figure 2 – Scatter plot of the test and training data combined.


Figure 3 – Scatter plot of the classified test features.

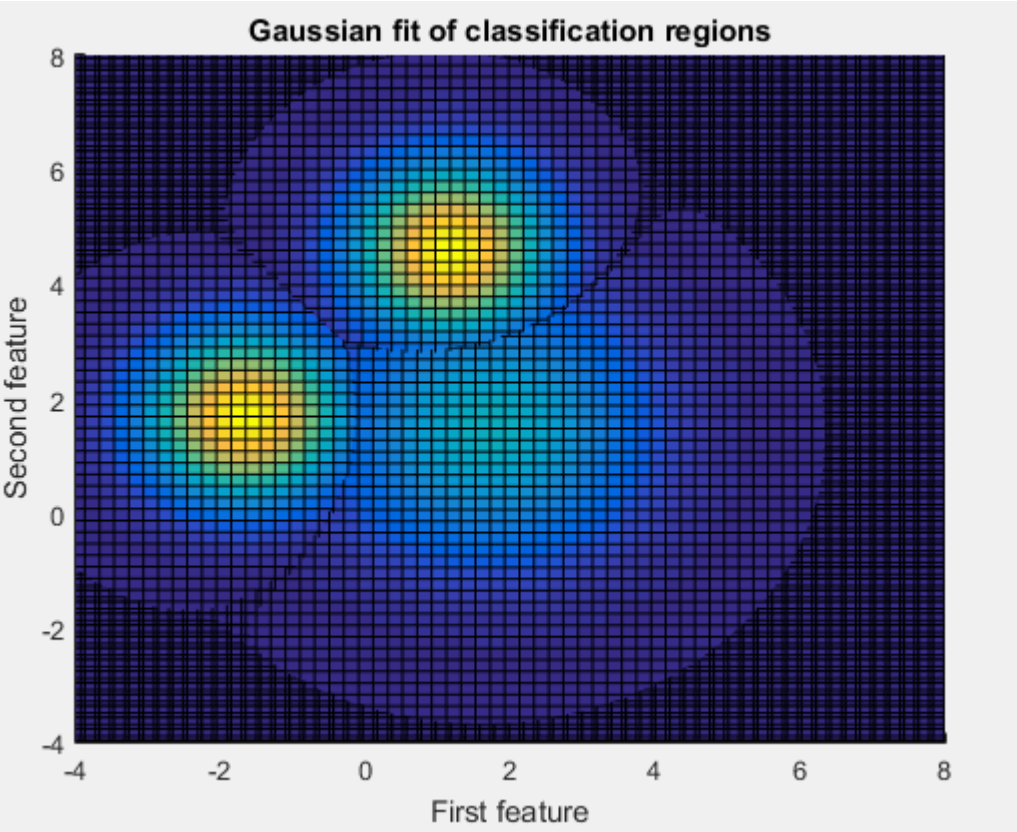



Figure 4 – 2D Gaussian fit of the classification regions.

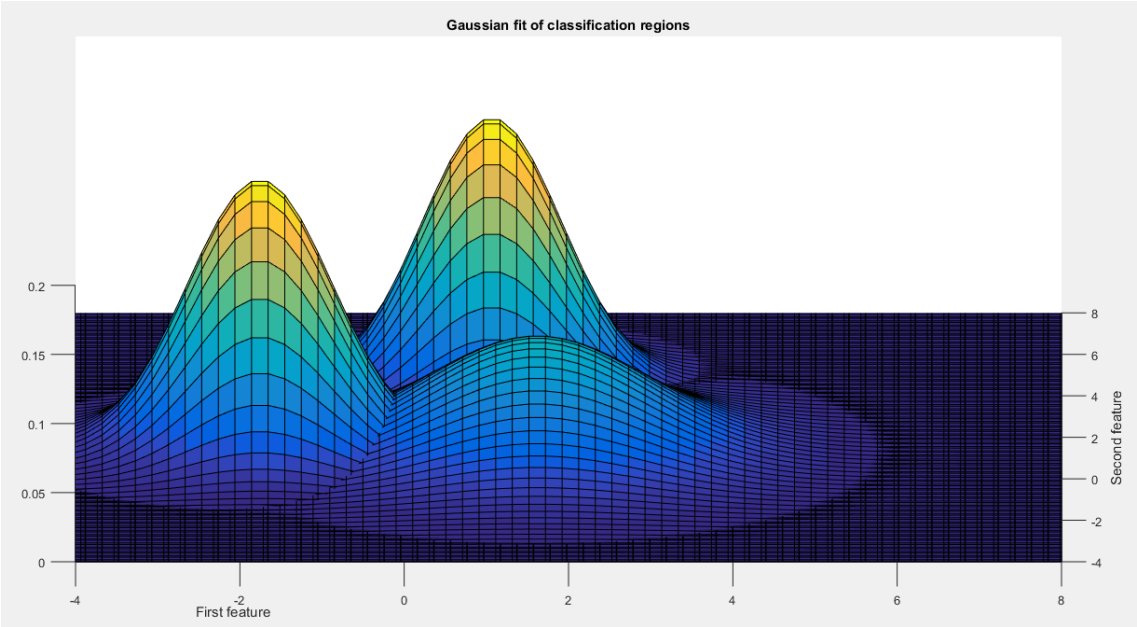



Figure 5 – 3D Gaussian fit of the classification regions.