SUMMARY

1. Introduction

Nowadays genetic engineering is enabling us to work with thousands of genes. All of these genes affect to organism, like for instance the tendency to suffer certain diseases. In this sense, the development of automatic methods that help us determine whether they play an important role or not in the appearance of a certain illness would mean a great advance in the field. This is because we are being required to measure the expression levels of thousand of genes in a single experiment and use these results to compare those which are expected to be different.

In this process we have to confront such a big problem, the number of experiments that can be used for these purposes is very limited and genetic sequences are composed by thousands of elements. For this reason, finding an algorithm which could sort out the distinctive signatures of gene expression has become a priority.

2. Problem description

In this research, we are looking for an automatic method that points out the genes that are more relevant to the diagnosis of several kinds of cancer. To do so we will employ three different databases: *Carcinom*, *Lung* and *Glioma*. Due to all the datasets include a huge number of gene expression values (several thousands) but we can only test with a small number of experiments (hundreds of patients) it has become necessary to face, among other inconveniences, the curse of dimensionality. Recent studies show that in DNA sequence there are a few discriminatory and very useful genes but also many others that are unrelated to the experiment. Therefore, we need to find an effective classification algorithm that provides us smaller subsets of genes, which could be easier and more practical to work with.

At this point is important to look over the classification problem we are trying to handle. We have an input matrix, NxD, where N means the samples and D represents the n-dimensional feature space. To our case, the

samples are the genetic arrays composed of thousands of coefficients (what we called the second dimension). The issue is that, we have two kinds of samples, some of them are patients and the other are used as controls. We have to find a function that allows us to differentiate them. This function will be the classifier and our objective is to analyze its performance in order to determine what genes in the sequence help it to decide.

In conclusion it is critical to be able to select a small subset of genes in order to increase the cost effectiveness and the ease of verification of the importance of certain elements in the genetic sequence.

3. The classifiers

Although the huge dimensionality is regarded as a big problem, it also gives us certain advantages. This multidimensional situation could be understood as an indeterminate compatible system which let us find easily a decision function that separate the training data in a properly and efficient way. It means that there are a lot of ways to classify our training samples and all of them would be correct. In order to both sorting out the samples in function of the class they belong to and establishing a ranking of the genes in function of its relevance, we have chosen a linear classifier.

In spite of the multiple linear classifiers we could use to resolve our problem, we have made a selection of those we consider more useful to develop this experiment. We must find a classification method that not only classify but also establishes which genes become more conclusive to develop this process. We have to make a ranking of the genes so we need an algorithm that gives them different levels of relevance. To do this, we have only chosen classifiers whose performance is based on weights assignment.

Fisher linear discriminant

Fisher linear discriminant is a classification method that projects high-dimensional data onto a direction and performs classification in this one-dimensional space. The projection maximizes the distance between the means of the two classes while minimizing the variance within each class.

It can be considered as a specific choice of direction for projection of the data down to one direction rather than a discriminant. Thus, the projected data can subsequently be used to construct a discriminant.

The perceptron algorithm

Artificial neural networks are a machine learning technique based on neurobiological structures found in nature. The perceptron algorithm is the simplest form of neural network. It is composed by several input neurons, which receive the external data, and a single output neuron that shows the classification result. These external input data would be multiplied by weights that would be initially chosen randomly. This would mean that our problem would be conditioned by these initial values.

Although both of these classifiers could be used to solve our problem, we decided to start with the SVM method since it is considered the *state-of-art* classification algorithm and it is supposed to provide us a higher rate of success in a shorter time than the others.

Support vector machines (SVM)

As we explained above, we need to find a way to select the best subset of features in the array. Among various possible methods we have determine that the weights we obtain from this classifier can be used as feature ranking coefficients. These weights show us the real influence of each variable in the sequence so if our algorithm performs well, the inputs with the largest weights in absolute value will correspond to the most informative features.

The support vector machines base their performance on the concept of the margin, which is defined to be the smallest distance between the decision boundary and any of the samples. This decision boundary is chosen to be the one for which the margin is maximized. Those samples that serve to define the classification boundary are considered the support vectors. On the opposite, the samples that are placed further away from the margin will have a null weight value. Although SVMs handle non-linear decision

boundaries of arbitrary complexity, we limit ourselves to linear SVMs because of the nature of the datasets under study.

Once the classification process has been finished we will obtain a weight vector. This output vector will help us to make a ranking of the features so we will be able to discard later those we consider the less relevant to our experiment.

4. RFE

As soon as we have checked the proper performance of the SVM classifier we must mention that the final objective of our research is not only making a classification but also sorting out the genes in the array in order to extract as much information as possible from the sequence to determine which positions are the most conclusive. It is important to remember that we count with a few genetic sequences but they are composed by thousands of elements. If we could select only the relevant variables to classify, we will obtain much more accuracy in the results.

To do that, we will use the RFE (recursive feature elimination) algorithm. This method establishes a criterion that allows us to extract those relevant elements from a reduced set of samples with a high number of dimensions. If we apply both SVM and RFE, we will base that criterion on the weight vector vector, calculated during the training process of the SVM. Once we have assigned the weights to all the genes, we will then sort the output vector and discard those that have a lower absolute value. Although this practice can affect the efficiency of the algorithm, it will provide us a global view from the interdependencies and correlations among genes and allow us to have all the results available in a shorter time.

Finally, we also have decided to include the calculation of the confusion matrix in this implementation. It is a square matrix which allows us to check if the mistakes are spread evenly among all classes or if, instead, there is any class with a higher error rate.

5. Robust RFE

In order to make sure of the real relevance of each gene in the array, we have decided to incorporate a modification in the RFE method; we will employ a sampling method called *bootstrap*. To do so we will repeat the algorithm thousand times. In each iteration we will build the training data by choosing only half of the samples randomly. Then we will store in which positions are eliminated the variables for every RFE we have tested. Finally we will make an average of them so we can determine which genes in the sequence the algorithm consider to be the most decisive. Once we have the vector in which all the dimensions are ordered, we will be able to start the classification process again. As we proceed in the previous implementation, we will only use the 200 first variables that the RFE algorithm gave a higher weight before in order to be able to compare later the performance of both algorithms.

6. Discussion of the results

Once we have obtained the results from the two algorithms for all the datasets it is time to make some comments.

On the one hand, we are going to talk about the confusion matrix. It allows us to check if the errors are uniformly distributed or if there are any class that accumulates a much higher error rate. In our problem, despite of having high success rates in both cases, the percentage of errors in the robust *RFE* is much more reduced than in the conventional *RFE*. We could be able to decrease it up to ten percentage points; even to achieve an error rate of almost twenty percent lower in one of the databases. This situation shows us that although the time we spend in executing the robust algorithm is a thousand times greater, we achieve such a big improvement by using it. This is because the error rate is the only measure we have to check if the process is working correctly. For this reason, having a lower number of mistakes indicates that the robust *RFE* chooses a better set of relevant features.

On the other hand, we have the gene selection. By implementing the robust *RFE* we obtained a new vector with the average relevance of each gene in the array. In this vector we observe new important variables that have been

discarded with the non-robust algorithm. Besides, we could check that the most conclusive variables are the same in both algorithms. Nevertheless, there are big differences among the dimensions which were considered as noise. This could be because in each iteration of the new algorithm it extracts some different variables so when we try to make an average the results diverge too much. Another reason is the strong correlation among certain genes. It causes the algorithm become unable to determine which gene is going to be more relevant or that the final result does not vary if we extract one or the other. However, making an identical ordination of the most conclusive genes helps us to verify that the algorithm has performed properly.

Besides, we could also see how the variance of the sequence size in the final classification process affects the results. Firstly, we established a comparison between both algorithms by choosing the 200 most relevant genes. But after that, we changed this value so we were able to check that selecting less than 100 variables increases the error rate more than the 30%. By contrast, if we use more than 300 or 400 we would have a lower number of samples incorrectly classified although not few enough to become a remarkable improvement of the error percentage. In the best case, by switching 200 samples to 300 we achieved an improvement about 3%. But if we changed then to 1000 we would only incremented it in a 5%. It meant that, although all the genes gave a piece of information, they were the most relevant that helped us clasify correctly.

To sum up, despite the fact that the robust *RFE* involves a higher computational cost the results we obtain are much better than when we used the normal *RFE*. Not only the accuracy of the classification increases but also the most determining genes of the sequence are confirmed with this implementation.