Monograph

Title: Gene Expression Tree Classifier System for MAIGES

Tutor: Prof. Dr. Eduardo Jordão Neves

Author: Xieli Zhaofu

Table of Contents

PART I

1. Introduction

DNA microarray and its related technologies are playing a crucial role in the modern molecular biology and medicine science researches. Normally, a DNA microarray consists of thousands and millions microscopic spots which can be used to measure gene expression by measuring the amount of mRNA[1] present. And, in a microarray involved experiment, one of the challenges to researchers is to find out new tools and methods to process efficiently the obtained gene related information.

This monograph discusses and represents some fundamental mathematical tools for the decision-making process in a pattern recognition system. And, as part of my T.C.C[2], a gene data tree classifier system which can be integrated into MAIGES[3] is also introduced.

The contents of this monograph are divided into several chapters and sections:

In Chapter 1, there is a brief introduction for pattern recognition system and its related concepts, which brings a background for this monograph. In the second part of the chapter, there is an introduction about the T.C.C's objective.

In Chapter 2 and Chapter 3, the realized studies and research topics, such as CART algorithm, decision tree etc., are summarized, as well as the structure of the created gene data tree classifier system.

In Chapter 4, there is the summary of the T.C.C.

In Chapter 5, the study and research reference resources are listed.

In Part II of this monograph, there are my opinions about how my B.C.C. course study related with my T.C.C. project.

1.1 Pattern Recognition System

Pattern classification, also called pattern recognition, is a task to classify data basing on priori known information or information extracted from a given data. It is being used in the diverse research and application areas, such as: image processing, bioinformatics, clinical medicine, etc.

A pattern recognition system could be run in two models: supervised learning or unsupervised learning. The details about these two models will be introduced in the chapter 2.

A typical pattern recognition system should include the following classification process steps:

[1] mRNA stands for *messenger ribonucleic acid*.
[2] T.C.C – Trabalho de Conclusão de Curso
[3] MAIGES - Mathematical Analysis of Interacting Gene Expression Systems, URL: http://www.maiges.org/

- Collection: corresponds to the process for gathering raw data. Sometimes, it is also being called "Sensing" when the raw data need to be gathered by sensors.
- Normalization: corresponds to the process for normalizing the raw data to be processed.
- Feature Extraction: corresponds to the process for extracting numeric or symbolic information from the normalized data.
- Classification: corresponds to the process for categorizing the data according to the extracted features, and generating output data.
- Comparing: is the process for comparing the output data with the desired output.
- Adjusting: is the process to adjust classifier parameters by using the comparing results.
- Classifier design completion.

In this project, the gene expression tree classifier system acts like a modified supervised learning pattern recognition system. The system uses priori known gene expression data as train data, and generates classifiers according to the features of data and the chosen classification algorithm.

1.2 Objective

The objective of this T.C.C. can be divided into the following steps:

- To study and research the related topics.
- To create a gene expression tree classifier system by using the knowledge obtained from the previous step.

At the step 1, as a completely newcomer to bioinformatics, my main goal was to study the Bioconductor and Microarray related topics as well as the related bioinformatics and statistical knowledge.

At the step2, the goal was to create a robust and portable classifier system which could be integrated into MAIGES and the R statistic computation environment. The created classifier system should provide the system users the information about classifiers and classifiers' selection process as well as the classifiers' performance information. Also, the system should be able to receive parameters from the users, and to generate classifiers according to the specified requirements. The information about the classifier system[4] is stated in Chapter 3.

[4] *Classys* is the name of the created classifier system

2. Concepts

Before writing the classifier system, some basic concepts needed to be studied. In this chapter, I will try to summarize some of the important concepts that I studied at step 1.

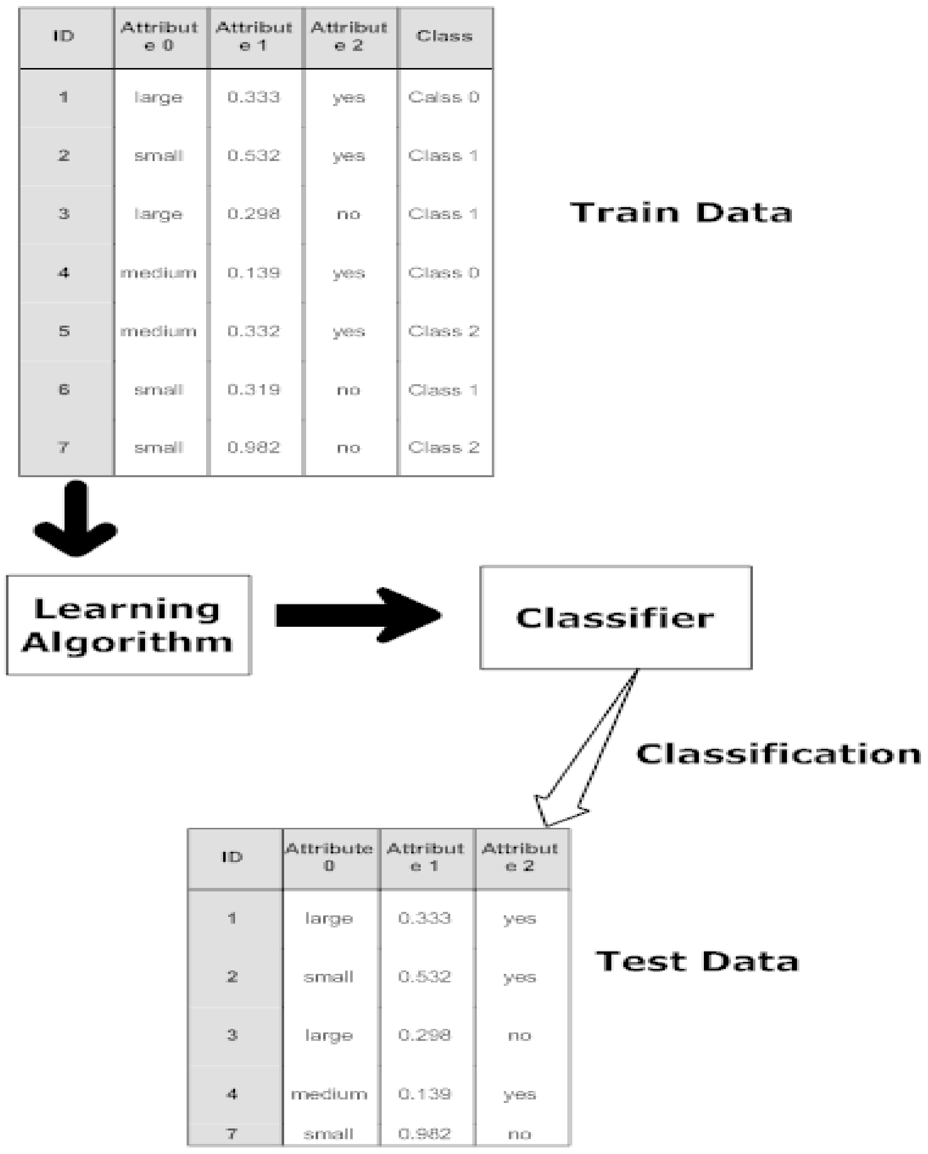
2.1 Classification

In the context of this project, classification is a way to categorize a given data sample according to the corresponding data features.

In other words, if we had a set of K known classes L = {*c1, c2, c3, …., ck*}, and a set of data X = {x1,x2, …., xm}. Classification should be a function F, by which F(X) =ci Є L, 1 ≤ i ≤ k.

The following diagram illustrates an imagined classification task: first, we have a given set of samples with IDs {1, 2 … 7}, named *"Train Data"*. For each sample, there are 3 attributes, named features, and one class label. And also, we have a set of samples with unknown class labels, named *"Test Data"*. The task is to "learn" the relations between the known *Train Data* features and their corresponding class labels, and uses the learning results to identify the unknown class labels of *Test Data*. As a precondition, we must sure that *Test Data* and *Train Data* belong to the same set of classes (i.e. they have at least some relations with each other).

In this diagram example, we can see that one machine learning algorithm was applied on Train Data to generate the corresponding classifier. And later, the generated classifier and the unknown Test Data features were used to find out the class labels.



ID	Attribute 0	Attribute 1	Attribute 2	Class
1	large	0.333	yes	Calss 0
2	small	0.532	yes	Class 1
3	large	0.298	no	Class 1
4	medium	0.139	yes	Class 0
5	medium	0.332	yes	Class 2
6	small	0.319	no	Class 1
7	small	0.982	no	Class 2

ID	Attribute 0	Attribute 1	Attribute 2
1	large	0.333	yes
2	small	0.532	yes
3	large	0.298	no
4	medium	0.139	yes
7	small	0.982	no

Figure 1: an imagined classification case

But in the microarray related research areas, things are not simple as the above example. One of the classification challenges is to choose a learning algorithm which could process efficiently from thousands and millions of gene features and generate an adequate classifier. Currently there are two principal classification techniques being used: one is unsupervised learning and the other one is supervised learning.

2.2 Supervised Learning

Supervised learning is a technique for learning the relations between data features and the corresponded data labels. It is called “supervised” because of the presence of the known data label variables to guide the learning process. The diagram showed in the section *2.1 Classification* is actually a supervised learning classification case. In order to solve a supervised learning problem, at least three requisites are needed: train data, the corresponding class labels and learning algorithms.

Supervised learning is widely used in the diverse research and application areas. One of the possible use cases is a handwritten recognition system. In this case, the train data is a set of handwritten characters' images, and the class labels are the corresponding characters. The task of a learning algorithm is to summarize the rules which could predict the identity of each known handwritten image quickly and accurately. And later, we could use the summarized rules to predict the identities of unknown handwritten images.

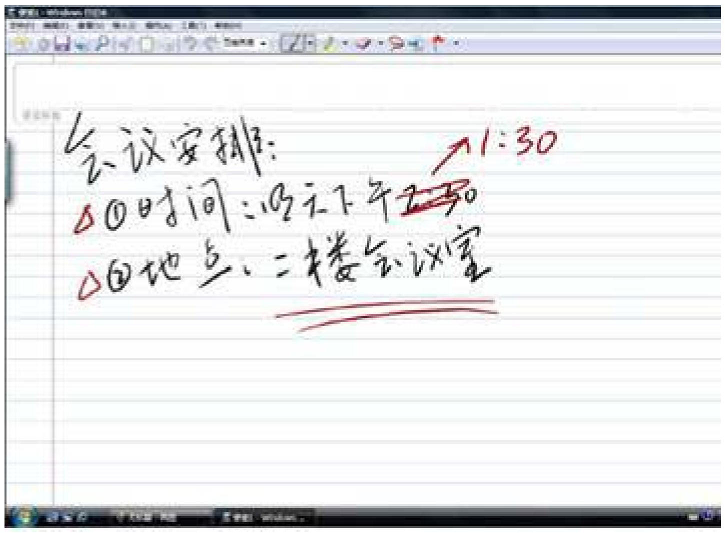

会议安排:
△①时间:明天下午~~2:30~~ 1:30
△②地点:二楼会议室

Figure 2: Microsoft Windows Vista has a recognition system which could identify Chinese handwritten characters and a search engine which could process queries from the handwritten contents.

Another supervised learning example is a weather forecast system. A weather forecast system normally applies one or more learning algorithms on some history weather observations to find out the relations between the observed data and the weather. And then, the obtained relations are used in the future weather forecast.

There are a lot of supervised learning algorithms and techniques, such as: decision tree learning, case-based reasoning, Gaussian process regression, learning automata etc. But in the context of our classifier system, the decision tree learning algorithms will be stated with details in Section 2.5 Classifier and Classification Algorithm.

2.3 Unsupervised Learning

Unsupervised learning is a technique to seek to summarize and explore the features of a given data set. The essential difference between supervised learning and unsupervised learning is that unsupervised learning involves statistical process analysis from only unlabeled train data. One of the unsupervised learning cases is "Clustering".

In a clustering problem, the task is to partition a raw data into subsets, named clusters. The data in the same cluster should have some common traits to different data of other clusters. Some notable application fields of clustering analysis are data mining, image processing, artificial intelligence etc.

Since unsupervised learning is not the focus of our classifier system, the technique details will not be discussed here.

2.4 Gene Expression Data

Gene expression data is a way to express DNA microarray experiments' results. A gene expression data collects the expression values with each column representing an experiment, and each row representing an individual gene.

In our gene expression tree classifier system, the system treats a gene expression data as a matrix with N rows and M columns. Each row represents a sample with M gene features. And we will use this convention in the chapter 3 when the classifier system is introduced.

2.5 Classifier and Classification Algorithm

In the context of a supervised learning process, a classifier is a function which is used to classify an object into one of the known classes basing on the related criteria. And a classification algorithm is used to generate classifiers from a set of known train set.

CART, the abbreviation of "*Classification And Regression Tree*", is a popular tree-based classification and regression algorithm.

Normally, a CART algorithm runs in 3 principal steps:

The first step is to choose splitting attributes. At each node of tree, attributes of train data are evaluated in order to find out the basis for separating the classes of the train samples. One of the typical evaluating functions is "Gini Index".

"Gini Index" is usually used to measure inequality of distribution, for example: inequality of income distribution or inequality of wealth distribution etc. The famous graphical representation of "Gini Index"[5] can be found in almost all articles and books which introduce "Gini Index".

The calculation formula is:

$$G = 1 - 2\int_0^1 L(X)dX.$$

Where L(X) represents the Lorenz curve[6].

The second step is to determine splitting criteria. Generally, a good splitting criteria is the one which produces "the purest" nodes in the smallest decision tree. But for different research experiment requirements, the notion of "the purest node" could be varied. In our classifier system, the purest node means a node with the lowest

[5] The URL of the image on Wikipedia:http://en.wikipedia.org/wiki/File:Economics_Gini_coefficient.svg
[6] Lorenz curve represents graphically the cumulative distribution function of a probability distribution.

misclassification ratios (i.e. the ratio between the predicted labels and the actual labels.)

And the third step is to determine when stopping split. The stopping criterion varies a lot according to the different experiments' goals. Sometimes, researchers should want that the splitting stops after all tree nodes being pure (i.e. perfectly classified). But sometimes, it is not viable to classify all tree nodes because of that the tree size could increase quickly. So some adapted criteria should be used instead of perfection.

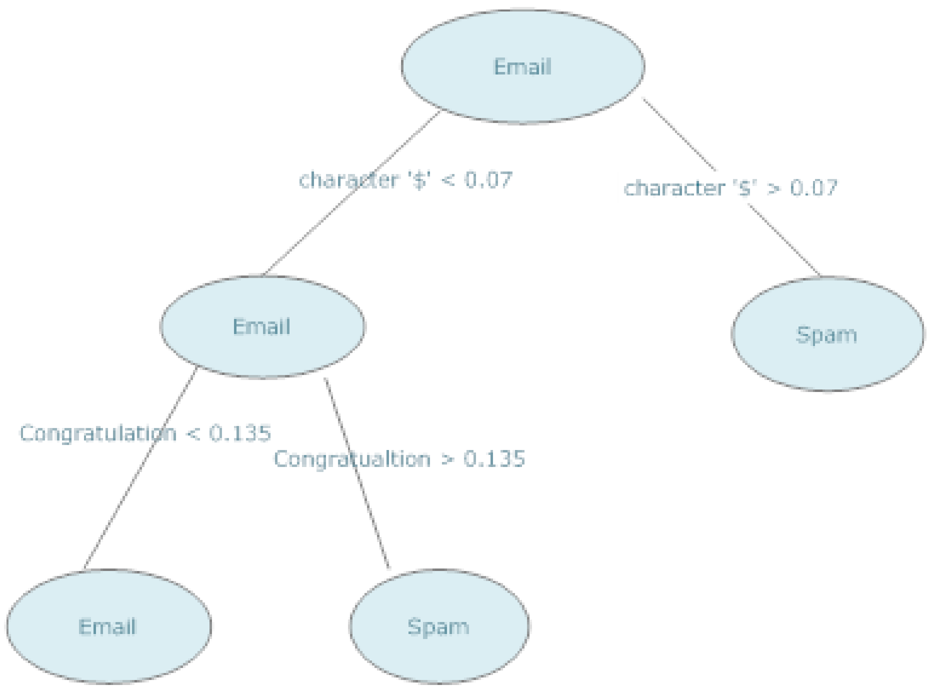


Figure 3: The diagram shows a typical CART tree of a simplified Email/Spam verification system

In order to explicate how a CART tree can be used, we can see a very simplified Email/Spam verification system example. The CART tree of this system is like Figure 3 showed above.

In this decision tree, the splitting attributes are the character '$' and the word "congratulation". The splitting criteria are: i) whether the appearance of the character '$' is less than 7% of email message words and ii) whether the appearance of the word "congratulation" is less than 13.5% of message words. Since there are only two questions (i.e. splitting criteria), the stop splitting criteria is to get all questions asked and answered. According to the splitting criteria, when a new message arrives, the verification system will first check if the percentage of appearance of the character '$' is less than 7% or not, if not the email is a spam, if so, the second question will be asked, whether the percentage of appearance of the word "congratulation" is less than 13.5%, if so the message is an email, otherwise it is a spam.

3. Classifier System – Classys

At the second step of my T.C.C., my main concern was how to build a robust and portable classifier system by using all the study results of the step 1 and my computation knowledge.

3.1 Picking Classification Algorithms

There are several approaches and algorithms used to perform predictive analysis, such as parametric and non-parametric techniques, regression and machine learning techniques etc.

Each algorithm has its own different approach and complexity, and also for different testing data, the same algorithm could represent completely different performances. Instead of picking a fixed classification algorithm for the classifier system, I decided to make a flexible system which could accept a set of candidate classification algorithms from the system users. At the same time, the system should also provide some default supported and tested classification algorithms to the users who don't have any specified classification algorithms in mind.

The tested classification algorithms by the classifier system are:

- Recursive Partitioning and regression trees, which is provided by the R package *rpart.*
- Classification and Regression trees, which is provided by the R package *tree*.
- Linear Discriminant analysis, which is provided by the R package *MASS*.
- Quadratic Discriminant analysis, which is also provided by the R package *MASS*.

The system users also have two ways to run a specified classification algorithm on the system: 1) by providing an algorithm from an xml file 2) or by calling system provided functions.

In the case of an algorithm is provided by an xml file, the xml file must follow a pre-defined schema as showing in the below:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<list>
      <algorithm>
            <is.ignored>FALSE</is.ignored>
            <name>my.tree</name>
            <package>tree</package>
            <function>
                  <name>tree</name>
                  <pass.arguments>TRUE</pass.arguments>
            </function>
            <train>
                  <parameter.name>data</parameter.name>
                  <combine.with.labels>
                        TRUE
                  </combine.with.labels>
                  <make.data.frame>TRUE</make.data.frame>
            </train>
            <labels>
```

```
                    <parameter.name>formula</parameter.name>
                    <formula>
                          <left.hand>labels</left.hand>
                          <right.hand>.</right.hand>
                    </formula>
              </labels>
              </algorithm>

        </list>
```

In this example, the tree function from the “tree” package is provided to the system. The system will read the package name which is “tree”, and load the package if necessary, and then read the function name which is coincidently “tree” and the function parameters. With these information, the system passes the parameters to the algorithm function and generates the corresponding classifier.

The other way is by calling the system functions: csys.make.algorithm(…) and csys.add.algorithm(…). The details are stated in the section *3.2 Classifier System Architecture*

3.2 Classifier System Architecture

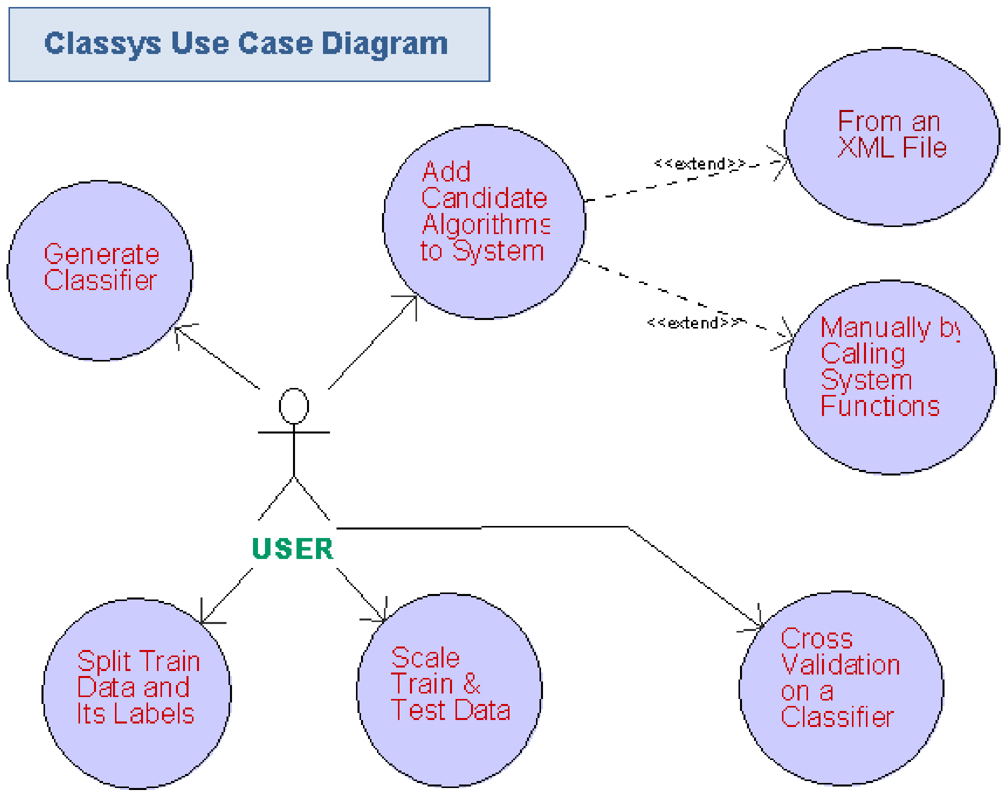

Classsys Use Case Diagram
Generate Classifier
Add Candidate Algorithms to System
From an XML File
<<extend>>
<<extend>>
Manually by Calling System Functions
USER
Split Train Data and Its Labels
Scale Train & Test Data
Cross Validation on a Classifier

Figure 4: *Classys* Use Case Diagram

As shown in the system use case diagram, the classifier system, named “Classys”, can be considered as a join of 5 use cases:

a. To add candidate algorithms to system:

(a) From an XML file: users can add a set of algorithms from a local or web based XML file by calling: csys.load.algorithm(file.name). The XML file format must be the same showed in the section 3.1.

(b) Manually: by calling two functions csys.make.algorithm(…) and csys.add.algorithm(…). When a user calls csys.make.algorithm(…), the following parameters are needed:

(i) name – the algorithm name, a user defined name to identify algorithm.

(ii) package – the package name.

(iii) func.name – the algorithm function name.

(iv) train.name – the train data parameter name, which should be used when algorithm function is being invoked.

(v) labels.name – the labels parameter name, which should be used when algorithm function is being invoked.

(vi) pass.argument – whether system function parameters will be passed to algorithm function. Its value must be either "TRUE" or "FALSE".

(vii) cbind – whether the train data is combined with its labels. This option sometimes is required by some classification algorithm. Its value must be either "TRUE" or "FALSE".

(viii) data.frame – whether the train data is in a data.frame[7] format. Its value must be either "TRUE" or "FALSE".

(ix) labels.formula.left –The symbol locates at left part, if the labels need to be passed to algorithm function in a formula[8] format.

(x) labels.formula.right - The symbol locates at right part, if the labels need to be passed to algorithm function in a formula format.

b. To generate classifier from a set of candidate algorithms: by calling csys.get.classifier(train, labels, algorithm.name, supported.algorithms, …)

(a) train – train data. It could be a matrix or a data.frame

(b) labels – train data labels, It should be a vector of string values.

(c) algorithm.name – the candidate algorithms' names. It should be a vector of string values

(d) supported.algorithms – all supported algorithms in the data.frame format. Normally it is obtained by calling csys.add.algorithm(…) or csys.algorithm.load(…)

c. To make Cross-Validation on a classifier: by calling csys.cv(train, labels, classifier, iterance, …)

(a) train – train data. Matrix or data.frame

(b) labels – train data labels.

[7] data.frame is an object format in R environment. For more details about it, please see the R related manuals which can be found here: http://cran.r-project.org/manuals.html

[8] A formula is a string format with two parts divided by "~", for example: "labels ~ ." is an acceptable formula. Please see the corresponding algorithm function manuals to know the acceptable formulas.

(c) classifier – classifier generated by csys.get.classifier(…)
(d) iterance – the number of iterance of cross validation on the given classifier.

d. To split data: by calling csys.split(data, labels, ratio, mask)
  (a) data – data to be split
  (b) labels – data labels to be split
  (c) ratio – splitting ratio
  (d) mask – user specified splitting index

e. To scale data: by calling csys.scale(train, test, scale.method)
  (a) train – train data
  (b) test – test data
  (c) scale.method – the scale method must be one of three supported methods: “max-min”, which scales the max. value to 1 and the min. value to 0; “mean-sd”, which scales mean to 0 and mean+sd to 1; “median-mad”, which scales median to 0 and median+mad to 1.

3.3 Notable Characteristics of *Classys*

Some notable characteristics of Classys are:

- Modular system design brings more use cases to the system users.
- The system users can use specified candidate algorithms by providing a formatted xml file which contains algorithm related information
- All system process results are disposable to the users
- Test demos and documentations permit new users to learn to use the system quickly

3.4 System Testing[9]

The data used in the system test is a table of gastro-genes. The table has the dimension of 100x20 which contains 100 genes of 4 different labels: Neso, Aeso, Aest and Nest.

The system test[10] was divided into 2 separated steps. At first step, the system supported classification algorithms were run on the testing data to generate the respective classifier of each algorithm. And then the results were used to compare the performance of each algorithm. At second step, we ran the generated classifiers on the testing data without labels to see the accuracy of each classifier.

The system tests were run on a PC which has CPU of Intel Q6600 2.4 GHz and

SDRAM memory of 3.0 GB, and the following is the testing result.

[9] The testing data can be found in */data/gene.txt* from the source/binary zipped package.
[10] The testing script can be found in /tests/system.test.R from the source/binary zipped package.

3.4.1 Step 1: Generating classifiers

The classifier generated by using the CART algorithm:

```
> classifier.tree = csys.get.classifiers(matrix, labels,”my.tree”, algos)

> classifier.tree
```

```
node), split, n, deviance, yval, (yprob)

    * denotes terminal node

1) root 20 55.45 Aeso ( 0.25 0.25 0.25 0.25 )

  2) X2: -0.015799108791839,-0.0186490962801162,-0.0811881544863418,-
0.108431377453378,-0.119049586229764,-0.17210287312337,-0.184770683945603,-
0.224280269632315,-0.343604415012653,-0.376069941715975 10 13.86 Aest ( 0.00 0.50 0.00 0.50 )

    4) X2: -0.015799108791839,-0.0186490962801162,-0.0811881544863418,-
0.184770683945603,-0.224280269632315 5  0.00 Nest ( 0.00 0.00 0.00 1.00 ) *

    5) X2: -0.108431377453378,-0.119049586229764,-0.17210287312337,-
0.343604415012653,-0.376069941715975 5  0.00 Aest ( 0.00 1.00 0.00 0.00 ) *

  3) X2: -0.10838010411441,-0.12254754275087,-0.132672761434407,-
0.180246340067187,-0.181784664445725,-0.214674455052881,-0.219679548553482,-
0.270169505178192,0.0235475057372925,0.0288372186306159 10 13.86 Neso ( 0.50 0.00 0.50 0.00 )

    6) X2: -0.10838010411441,-0.180246340067187,-0.214674455052881,-
0.219679548553482,0.0235475057372925 5  0.00 Aeso ( 1.00 0.00 0.00 0.00 ) *

    7) X2: -0.12254754275087,-0.132672761434407,-0.181784664445725,-
0.270169505178192,0.0288372186306159 5  0.00 Neso ( 0.00 0.00 1.00 0.00 ) *
```

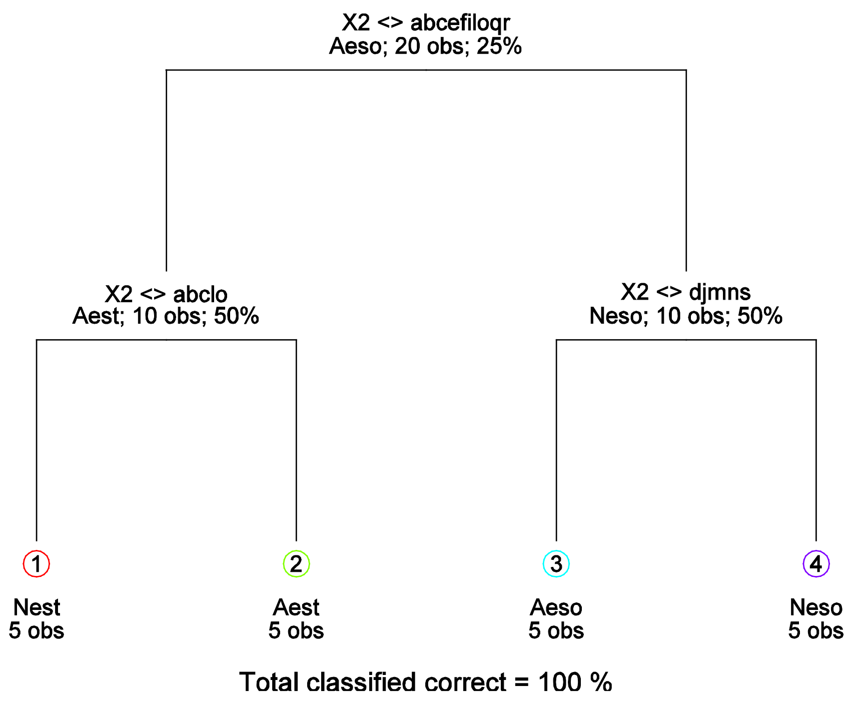


Figure 5 The hierarchical diagram of classifier.tree

The classifier generated by using the rpart algorithm:

```
>classifier.rpart = csys.get.classifiers(t.matrix, t.labels, "rpart", algos);

> classifier.rpart

$name

n= 20

node), split, n, loss, yval, (yprob)

      * denotes terminal node

1) root 20 15 Aeso (0.2500000 0.2500000 0.2500000 0.2500000)

 2) X1=Aeso,Aest 10  5 Aeso (0.5000000 0.5000000 0.0000000 0.0000000) *
```

*3) X1=Neso,Nest 10 5 Neso (0.0000000 0.0000000 0.5000000 0.5000000) **

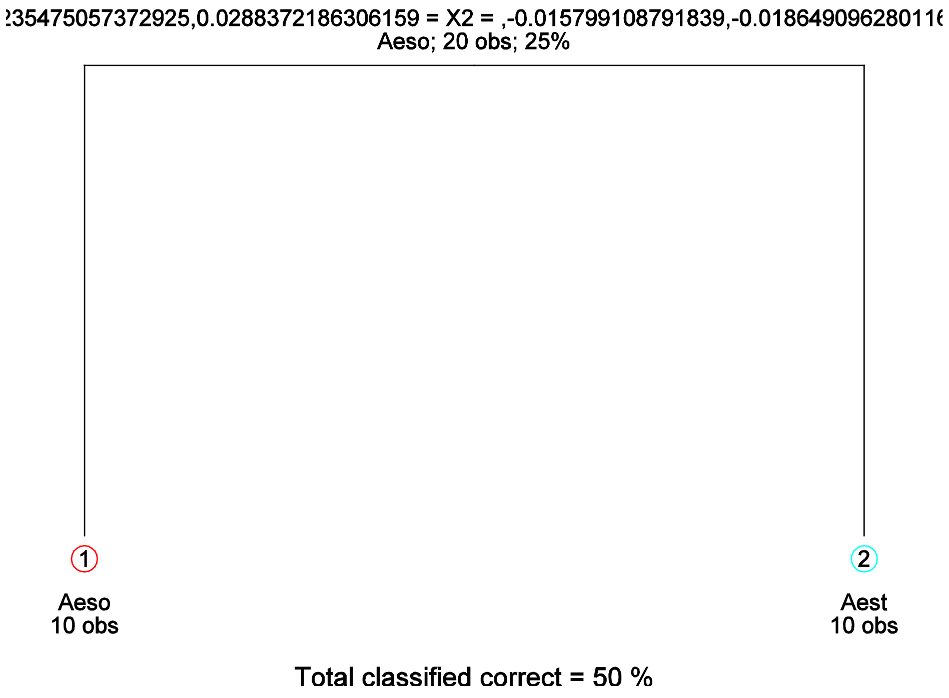

235475057372925,0.0288372186306159 = X2 = ,-0.015799108791839,-0.018649096280116
Aeso; 20 obs; 25%
1
Aeso
10 obs
2
Aest
10 obs
Total classified correct = 50 %

Figure 6 The hierarchical diagram of classifier.rpart

The classifier generated by using the linear discriminate analysis (“lda”) algorithm:

> classifier.lda = csys.get.classifiers(t.matrix, t.labels, "lda", algos);

> classifier.lda

… (Author’s note: since the classifier details result two “tedious” numeric matrixes, the matrix part is shielded here)

```
Proportion of trace:

   LD1       LD2       LD3

0.5542    0.2589    0.1869
```

Below is a tidy algorithms’ performances comparing table

Algorithm	Speed (in second)
Tree	86.42
Rpart	8.14
Lda	0.08

3.4.2 Step 2: Applying the classifiers

Applying the tree classifier on the gastro-genes:

```
> predict.tree(classifier.tree, newdata = data.frame(t.matrix), type = "class")

 [1] Neso Neso Neso Neso Neso Aeso Aeso Aeso Aeso Aeso Aest Aest Aest Aest Aest Nest Nest
Nest Nest Nest

Levels: Aeso Aest Neso Nest
```

Applyig the Rpart classifier on the gastro-genes:

```
> predict(classifier.rpart, newdata = data.frame(t.matrix), type = "class")

   V2   V3   V4   V5   V6   V7   V8   V9  V10  V11  V12  V13  V14  V15  V16  V17  V18  V19  V20
V21

 Aeso Aeso Aeso Aeso Aeso Aeso Aeso Aeso Aeso Aeso Aest Aest Aest Aest Aest Aest Aest Aest
Aest Aest

 Levels: Aeso Aest Neso Nest

 > predict(classifier.rpart,data.frame(t.matrix), type = "prob") ## Note: measure the
classification accuracy probability

     Aeso Aest Neso Nest

 V2   0.5  0.0  0.5  0.0

 V3   0.5  0.0  0.5  0.0

 V4   0.5  0.0  0.5  0.0

 V5   0.5  0.0  0.5  0.0

 V6   0.5  0.0  0.5  0.0

 V7   0.5  0.0  0.5  0.0

 V8   0.5  0.0  0.5  0.0

 V9   0.5  0.0  0.5  0.0

 V10  0.5  0.0  0.5  0.0

 V11  0.5  0.0  0.5  0.0

 V12  0.0  0.5  0.0  0.5

 V13  0.0  0.5  0.0  0.5

 V14  0.0  0.5  0.0  0.5

 V15  0.0  0.5  0.0  0.5

 V16  0.0  0.5  0.0  0.5
```

```
V17  0.0  0.5  0.0  0.5

V18  0.0  0.5  0.0  0.5

V19  0.0  0.5  0.0  0.5

V20  0.0  0.5  0.0  0.5

V21  0.0  0.5  0.0  0.5
```

Applyig the lda classifier on the gastro-genes:

```
> lda.out = predict(classifier.lda, data.matrix(data.frame(t.matrix)));

> lda.out$class;

 [1] Neso Neso Neso Neso Neso Aeso Aeso Neso Aeso Nest Nest Aest Neso Aest Aest Aeso
Aest Nest Nest Nest

Levels: Aeso Aest Neso Nest
```

3.4.3 System testing conclusion

On this particular gastro-genes sample, we found that the “tree” algorithm generates the most accuracy classifier which results a classification of 100% accuracy, although it has the slowest speed to analyze the train data. On the other side, the classifier generated by rpart permits us to measure the classifier accuracy probability as shown above.

The noteworthiness point is that the classifiers’ accuracies of the different algorithms could vary a lot according to the natural of the applied data sample. For example, if we applied the cited algorithms on a smaller gene data “iris[11]” to generate classifiers, we could get the 100% of prediction accuracy from all the generated classifiers.

[11] The data could be found in the MASS package.

4. Conclusion

As my first experience with bioinformatics, this T.C.C. gave me an opportunity to study bioinformatics and statistical related topics and learn how computer science, such as algorithms and software engineering can be used in other science research areas.

Also, I realized the importance of software engineering, and how a well designed architecture can change completely the performance of software.

5. Bibliography

Duda, R.O., Hart, P.E. and Stork, D.G., Pattern Classification, Wiley, 2000

Esteves, G.H., MaigesPack, R, http://www.bioconductor.org/packages/2.3/bioc/html/maigesPack.html

Freeman, E., Bates, B. and Sierra, K., Head First Design Patterns, O'Reilly, 2004

Friedman, J., Hastie, T. and Tibshirani, R., The Elements of Statistical Learning; Data Mining, Inference, and Prediction, Springer, 2001

Hill, T. and Lewichi, P., Statistics: Methods and Applications, StatSoft, Inc., 2006

Jones, N.C. and Pevzner, P.A., An Introduction to Bioinformatics Algorithms, The MIT Press, 2004

Ripley, B.D. and Venables, W.N., Modern Applied Statiscs with S, Springer, 2002

rpart, Recursive partitioning and regression tree packages, R, http://cran.r-project.org/web/packages/rpart/index.html

The R Manuals, R, http://www.vps.fmvz.usp.br/CRAN/manuals.html

tree, Classification and regression trees, R, http://cran.r-project.org/web/packages/tree/index.html

Wikipedia, http://en.wikipedia.org

Zvelebil, M. and Baum, J., Understanding Bioinformatics, Garland Science, 2007

PART II

1. Challenges & Frustrations

Well, I would like to say that my B.C.C. life at IME-USP was filled full of challenges and also some frustrations.

I think the big challenge that I have met during these years is my communication difficulty, which not only caused my difficulty in my study but also caused other things like misunderstanding. These things certainly brought also the frustration feelings.

But at the other side, I am very grad to have luck and the opportunity to be able to study in IME-USP, one of the most respected institute of Brazil and the world, and being graduated from there.

2. Related Courses to TCC

Before listing the related courses, it is very important to remember all professors who ministered the courses, and their excellent quality.

MAC0122 – Princípios de Desenvolvimento de Algoritmos introduced me to the computer sciene study.

MAC0328 – Algoritmos em Grafos and MAC0338 – Análise de Algoritmos, these two courses constructed my algorithms' base, and also I learned to understand how to build efficient algorithms.

MAC0332 – Engenharia de Software and MAC0441 – Programação Orientada a Objetos, these two curses are fundamental to help me to design and construct the classifier system.

MAE0121 - Introdução a Probabilidade e a Estatística I , MAE0212 - Introdução a Probabilidade e a Estatística II and MAE0228 - Noções de Probabilidade e Processos Estocásticos. The topics studied from these courses are fundamental to help me to build a statistical classifier system. Probability notions are used to calculate the classifiers accuracy etc.

A lot of other BCC courses do also relate directly or indirectly to my TCC project, but the courses listed above are more significant.

3. Acknowledgement

I would like to use this space to acknowledge the contribution of my tutor Prof. Eduardo Jordão Neves, thank him for the project opportunity and his instruction. Also, I would like to thank all the professors who have taught me during these years.