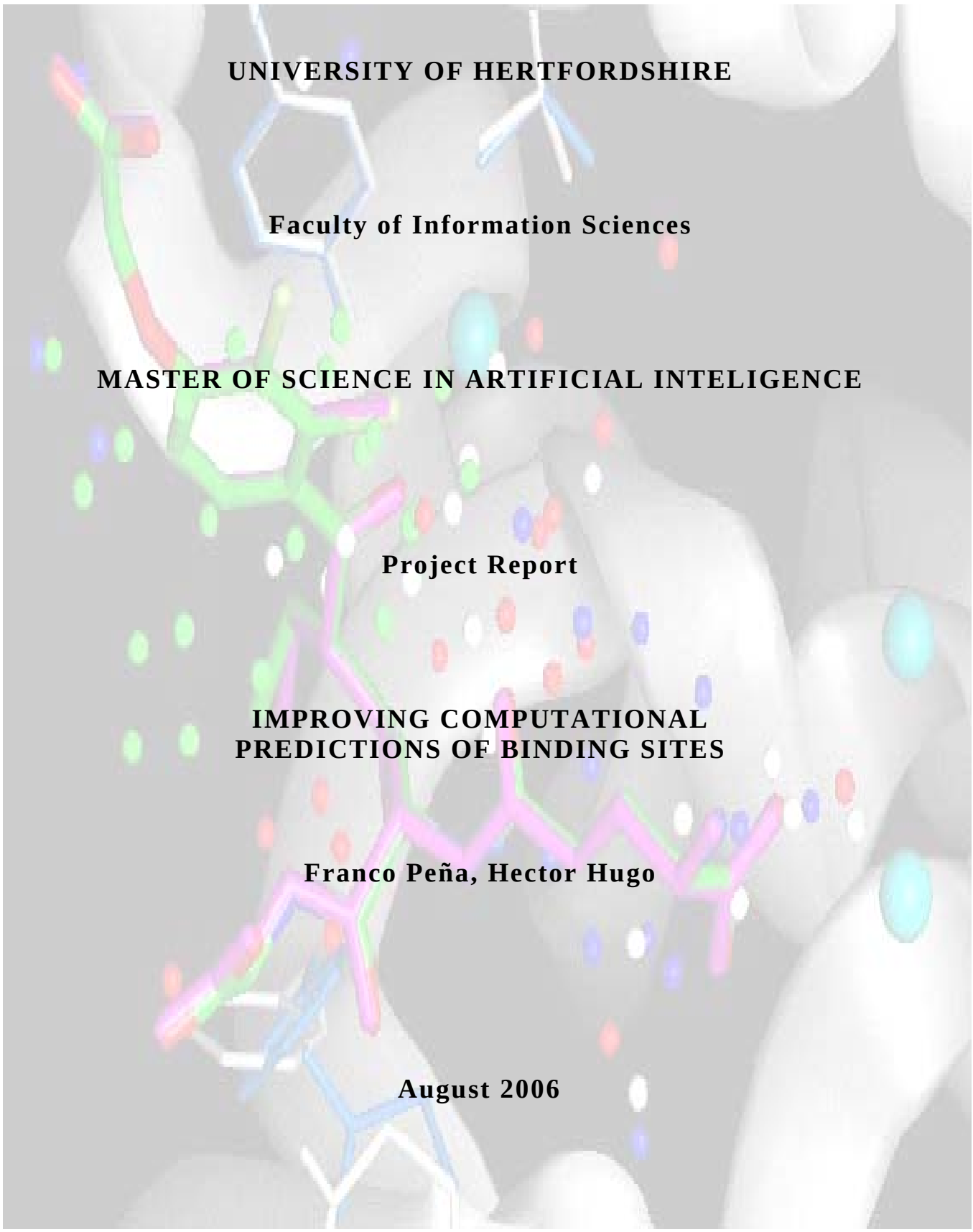


UNIVERSITY OF HERTFORDSHIRE

Faculty of Information Sciences

MASTER OF SCIENCE IN ARTIFICIAL INTELIGENCE

Project Report

IMPROVING COMPUTATIONAL PREDICTIONS OF BINDING SITES

Franco Peña, Hector Hugo

August 2006

ACKNOWLEDGES

Quiero dar las gracias a mi padre y a mi madre por darme la oportunidad de estudiar este año en la Universidad de Hertfordshire y consecuentemente realizar este proyecto.

También quiero agradecerle a mi supervisora Yi Sun, el haber gastado su tiempo cada semana en explicarme los detalles del proyecto durante todo el año.

I would like to thank my father and my mother for living me the chance to study this year in the University of Hertdfordshire an consequently conduct this Project.

I would also like to express my gratitude to my supervisor Yi Sun, because she was expending a long time every week to explain me details about the project during all the year.

IMPROVING COMPUTATIONAL PREDICTIONS OF BINDING SITES

INDEX:

i. Index 3
ii. Abstract 4
iii. Introduction 4
iv. Description of the task and data 4
v. Objective 7
vi. Metrics 7
vii. Sampling techniques: SMOTE (Synthetic Minority Over-sampling Technique) algorithm 8
  a. Over-sampling 8
  b. Under-sampling 8
viii. Cross-validation 8
ix. Generating the training and testing datasets. 9
x. SVM : Support Vector Machine 11
xi. Implementation 13
xii. Discursion 14
  a. Computational problems 14
    i. Time 14
    ii. Memory optimization 15
  b. Algorithms 16
  c. Programming language and SVM 16
  d. Smote algorithm 17
  e. Generating data 15
  f. The best model 18
  g. Conclusion 19
xiii. Results 20
xiv. Observations 25
xv. References 26
xvi. Index of tables and figures 29

Appendix:
a. Example of space linear separable (SVM). 30
b. Illustration of SVM kernel mapping. 31
c. Pseudo code of Smote algorithm. 33
d. Description of the modules. 34
e. More details about the results. 41
f. Implementation of the module in C (SVM-MATRIX.C). 43
g. Implementations of the modules in Matlab. 47
h. Figures for RBF kernel. 82
i. Figures for Sigmoid kernel. 140
j. Content of the CD. 156

IMPROVING COMPUTATIONAL PREDICTIONS OF BINDING SITES

Hector Franco Peña
University of Hertfordshire

Knowing where the cis-regulatory binding sites in the DNA are is one important priority for a multitude of research areas like: embryonic development, evolution, pharmacogenemics, cancer and many other transcriptional diseases, and it will become an important precursor for the reverse engineering of genome. That is the reason why there exists a multitude of computational strategies and algorithms to predict cis-regulatory binding sites, but currently all the algorithms have very low accuracy.
In this project I use a SVM to integrate the predictions of 12 different algorithms and compare the efficiency of the SVM with different size inputs; permitting contextual information of the outputs of this 12 algorithms for the neighbour positions.
The results show an improvement of the precision and a decrease in false positive rate between single data and windowed data (when have contextual information).

III. INTRODUCTION

There are a large number of different algorithms in current use to search for predicting binding sites (Sun et al 2005b; Bailey and Elkan 1994; Blanchette and Tompa 2003), but most of them have a high rate of false positive predictions.

I perform the same experiments as recently published (Robinson et al 2006; Sun et al 2005): integrating binding sites predictions of 12 different algorithms but using different windows size for the input (http://rulai.cshl.edu/scpd/)

IV. DESCRIPTION OF THE TASK AND THE DATA:

It is estimated that more than a half of human genome is cis-regulatory DNA (Markstein et al 2002): regulatory sequences that are located near the gene being regulated. A gene here is understood as a region of DNA that is transcribed, that is, read by the RNA polymerase to form an RNA. Regulatory proteins bind to the cis-regulatory DNA (binding sites) by chemical interactions (working like a key and lock).

Understanding these cis-regulatory elements has a tremendous importance for the research on important biological phenomena (Armone and Davidson 1997), and is of great interest for the pharmaceutical industry. It is believed that even a partial knowledge about the

regulatory elements will allow for a partial reconstruction of the genetic regulatory networks, and thus the understanding of the development of the cells, either normal (which implication for possible enhancement) or pathological (for example, resulting in cancer).

Predict a binding site position it is difficult in these because are a large number of very different proteins to lock different places, and the algorithm attempt to find the way to recognize some patters who identify all these places.

The data consists of 68910 possible binding positions (Sun et al 2005b) and the prediction by 12 different algorithms using different strategies. The information for each position is the known label and the predictions. The labels correspond to the positions in DNA: 1, for a binding site or 0, if a position does not belong to a binding site. The resolution for the binding site depends of the particul ar method among the 12 employed, a value greater than 0 signifies a binding site, below 0, a position in DNA that has not been identified as belonging to a binding site. This value can be a real value between -1 and 1 or a binary value (-1 or 1).

The 12 algorisms are (taken form Table 2 and 3 of Robinson et al 2006)

- scanning strategy algorithms:
  - Fuzznuc (www.hgmp.mrc.ac.uk/software/Emboss/)
  - Motif Scanner (Thijs, G. et al 2001)
  - Ahab (Rajewsky, N, et al 2002)
    - This strategy attempt to generate a model such as a position weight matrix for each binding site.
- for statical strategy algorithms
  - PARS (http:// sourceforge.net/projects/pars/)
  - Dream (over represented motifs) (Radvojac et al 2004)
  - Dream (under)
  - Verbumculs (Apostolico et al 2000)
    - This strategy attempts to find structures that make unlikely some sequences in the context of a binding site
- Co-regulatory strategy algorithms
  - Meme (Bailey and Elkan 1994)
  - AlingACE (Hughes et al 2000)
  - Sampler (Institute for Systems Biology) (http:// sourceforge.net/projects/netmotsa/)
    - These use iterative techniques (such as Expectation-Maximization algorithms)
    - Expression profiles are relied upon by co-regulatory algorithms using hypothesis that genes are clustered
- Evolutionary strategy or Phylogenetic algorithms
  - SeqComp (Brown et al. 2002)
  - Footprinter (Blanchette and Tompa 2003)
    - These methods compare the sequences from different species to find similarities

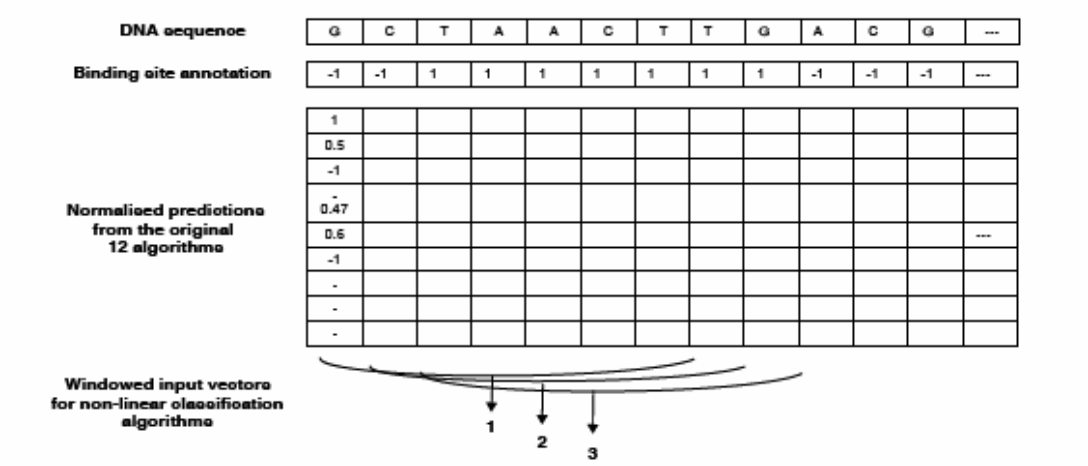


Figure 1 Window dataset
It show how is generated the windows data form the 12 algorithms

From this data about possible binding positions, the "widow data set" has been constructed.
For the "window data set" each input has the results for the 12 algorithms for (n-1)/2 neighbours positions where n is size of window. For example for window size equal to 3 each input has (3-1)/2 neighbours positions.

The data has been divided in two groups a training set (2/3 of the whole data) and a testing set (remaining 1/3)

In the training set, the inconsistent data and the repeated inputs have been removed, as was done previously (Sun et al 2005; Sun et al 2005b). This removal results in the reduction of 70% of the positions in the data set contend, but for a data set with a n= 7, only 5% of the columns are removed (Sun et al 2005b). These suggest that using a window can allow to resolve the problem of inconsistent data. However, previous results show (Sun et al 2005b) that the accuracy for the windowed data was worse than the accuracy for the single data when is predicted the binding sites using SLN (single layer network), SVM (support vector machine) and C4.5-Rules

Other important point is that the dataset is imbalanced, only 7% of the columns are binding positions (Japkowicz 2003). To balance the data, I will use the same techniques as employed before (Sun et al 2005b): under-sampling for the no-binding site class and over-sampling for the binding site class.

V. OBJECTIVE:

The objective of this project is to integrate 12 various methods to predict biding sites and obtain a meta-algorithm with better prediction accuracy than these 12 methods. And find the better value of the parameters to obtain the highest Precision with the lowest false positive rate.
For integrate this 12 methods I use a SVM: Support Vector Machine: is a learning classification method described in the section ten
And for help the machine is also integrated not only the 12 predictions for one point also some contextual information about the predictions for the neighbours positions, called windowed data, where the size of the windows indicate how many neighbours are imputed in the support vector machine.

VI. METRICS

Knowing the predictions and the labels of some data it is possible to construct the so-called confusion matrix, where TP is the number of true positives, TN is true negative, FP is false positive and FN is false negatives as shown in table 1 (Sun et al 2005b; Robinson et al 2006; Bussemaker and Siggia 2001; Tompa et al 2005).
Using these values one can calculate the Recall, F-score, FP_rate, Precision and Accuracy as follows:

Table 1: Performance metrics:

$$confusion\_matrix = \begin{pmatrix} TN & FP \\ FN & TP \end{pmatrix}$$

$$\text{Re}\, call = TP/(TP + FN).$$

$$F - socore = \frac{2 * \text{Re}\, call * \text{Pr}\, ecision}{\text{Re}\, call + \text{Pr}\, ecision}$$

$$Fp\_rate = \frac{FP}{FP + TN}$$

$$\text{Pr}\, ecision = \frac{TP}{TP + FP}$$

$$Accuracy = \frac{TP + TN}{TP + FN + TN + FP}$$

$$nCC = \frac{TP * TN - FN * FP}{\sqrt{(TP + FN) * (TN + FP) * (TP + FP) * (TN + FN)}}$$

VII. SAMPLING TECHNIQUES: SMOTE (Synthetic Minority Over-sampling Technique) ALGORITHM:

The dataset is imbalanced and is not appropriated for learning algorithms (Chawla et al 2002); because the algorithm (SVM) can learn that every input corresponds to the majority class (this can produce a good accuracy but not a good work)
For producing a new balanced dataset using over-sampling with the minority class and under-sampling with the majority class.

VII.A: Over-sampling:

This technique does not improve the minority class recognition, it only increases the minority class. The effects is identify similar regions of the minority class, this method has been proved successful in handwritten character recognition (Ha and Bunke 1997).
For each example of the minority class, for the k nearest neighbours (using Euclidean distance) taking the vector difference between the sample and the neighbour, this vector is multiplied by a random number between 0 and 1 plus the sample is added to it like a synthetic example.

The algorithm to calculate the distance is VDM, (Value Difference Metric). The implementation is in the appendix G

VII.B: Under-sampling

The majority class is reduced by randomly removing selection it helps to make more exactly the balance between the majority and minority class.

To see the pseudo code and the implementation code of SMOTE algorithm see the appendix.

VIII: CROSS VALIDATION.

This procedure can help to find “good values” for the parameters c: cost and g: gamma (Chang and Lin 2001).

The first step is to divide the training set into V subsets with the same size (in this case 10 subsets).
The second step is to train the SVM with V-1 subsets (in these case 9) and test with the last one subset. (This step is repeated V times for all the subsets). It prevents the overfitting problem.

The model that give the best group of subsets (tested in the subset not used in training) is used to predict the testing data. So this model generated with less information than the normal procedure.

IX. GENERATING THE TRAINING AND TESTING DATASETS:

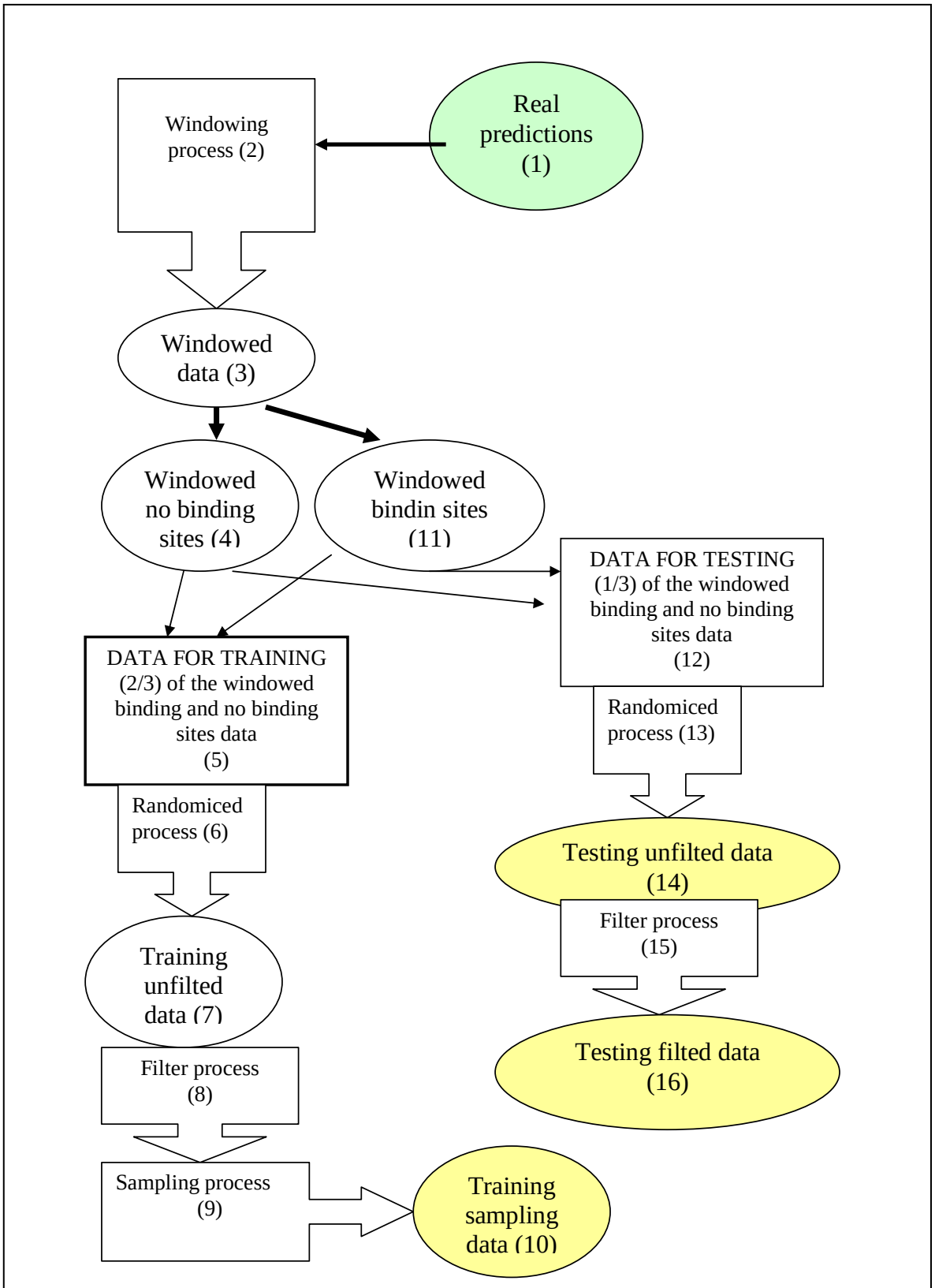


Figure 2 flux diagram of generating data process

In figure 2 it can be seen that following sub-process between the file: “real predictions” and the files “Training sampling data (18)”, “Testing filter data (12)” and “testing unfitted data (11)”.

(1) Is a matrix with the information of the prediction of 12 algorithms and the label “the real prediction of the biologist workers”.
(2) Windows process: this process generates a new matrix, where each line has the prediction of the 12 algorithms for it-self position and the prediction of the 12 algorithms for (window-1) neighbours positions. The new matrix have 12*windows size +1 columns.
(3) (9) (14) is needed to randomize the order of the rows in the matrix to produce a better training data (usually the neighbours positions have the same information)
(5) (6) In this step the data is spitted in two groups one for binding sites and another for no-binding sites, the aim is make a training data and testing data using the same proportion of binding sites/no-binding sites.
(8) (13) 1/3 of the information is used for testing the classifier support vector machine, and 2/3 of the information is use for training.
(10) The testing unfitted data is directly used for testing, but only need be translated and saved in a file in SVM format.
(11) (16) these steps remove redundant and inconsistent information.
(12) Testing filter data, this is the last step in testing data. Processing this also inform about in consistent data.
(15) Training un-filter data is not used for training because is imbalanced.
(17) Sampling process is need it for generate a balanced data. In the minority class (binding sites) use over-sampling it means produce a 500% more synthetic information, and under-sampling in the majority class reduce the data since to make it balanced. 44.4% are no-binding sites, and 55.6% are binding sites.
(18) Training samplings data: this file is enough to train the support vector machine properly.

X. SVM: Support Vector Machine:

Support Vector Machine (SVM) is a supervised learning method is an alternative for neural networks. A large number of methods has demonstrated that it have better accuracy than traditional classification methods like neural networks (Burges 1998).

The first step that the SVM is to change and increase the input space as: $R^N \xrightarrow{\phi} R^M$ where: M>N

" R^N " Means: the inputs space has N dimensions

" R^M " Means: the new mapped space has more dimensions than N.

" ϕ " is the kernel function.

Each input X has a new representation z: $z=\phi(x)$

$z \in R^M, x \in R^N$

$Z=\phi(X)$

Table 2 The most common kernels: (Ayat et al 2002)

Kernels	**Formula K(X,Y) =**
Linear	x*y
Sigmoid	tanh(a*x*y+b)
polynomial	(1+x*y)^d
RBF: Radial Basic Function †	exp(-g||x-y||^2)
exponential RBF‡	exp(-g||x-y||)
Where a,b,d are kernel parameters.	

The idea of this transformation is to make a new space where the inputs (vectors) are linearly separable, and make a hyper-plane which can separate classes with the maximum distance to the closet inputs data.

(Chang and Lin 2001) recommend to use the RBF kernel as the first option of the kernels listed in the table 2 because have less "numerical difficulties" than the others.
Lineal kernel is a especial case of RBF kernel (Keerthi and Lin 2003)
And sigmoid kernel can also behaves like RBF kernel for same parameters (Lin and Lin 2003), this kernel is not valid under some parameters (Vapnik 1995).

† This is the kernel that I use in this project.
‡ The source code of SVM that I use does not have implemented this kernel.

The reason why the inconsistent data is removed (the input vectors which are classified in more than one group at the same time) is that there is no any kernel that can separate the same input into a two different classes

In the appendix A are some examples of spaces linear separable and examples about the kernels transformations.

SVM do with this new space is to find a hyper-plane which split the space in two classes. In this case are only two classes, so it use a single linear discriminant function (Bishop 1995) learning algorithm like perceptron (Cristianini and Shawe 2000).

XI. IMPLEMENTATION:

To accomplish the objective of this work, it is necessary to first integrate the results form 12 different base algorithms for binding site prediction. These algorithms are described in the section 2.

Then, I used the windowed inputs where a fixed number of consecutive results were used as an input vector (contextual information).
I used “Support Vector Machines” to predict (if a particular column belongs to a binding site or not) inputting to the SVM the outputs of the 12 algorithms for predicting binding sites and in some cases inputting the outputs of the 12 algorithms also for the neighbours positions of the DNA

The parameters that are optimised are c: cost and g: gamma.
The parameter c: are tested for the values between 2^-3 and 2^11.
The parameter g: are tested for the values between 2^-11 and 2^3.
Both are increased adding 2 at the exponent.

I used one implementation of SVM coded in C++ developed by (Chang and Lin 2001). This vas downloaded from “http :// csie .ntu .edu .tw/ ~cjlin /libsvm” The SVM model is generated with “svmtrain.exe” and the metrics of classification are calculated from the confusion matrix that gives “svm-matrix.exe” this is a modification from the code “svmpredict.c” The new version only return the confusion matrix, the source code of this file in the appendix (“svm-matrix.c”), it was compiled in GCC environment in Windows.

The data is form the file “realPredictionsMatrixOpt.dat” produced by (Marks Robinson), it have 14 columns of numbers, 12 outputs of algorithms for search binding sites, one column with only -1 the fourth and the last one is the label 1 or -1 and 67782 rows of examples.

These real predictions are used for generate the training and testing data.
To do this I implemented and reused some Matlab code taken from (Yi Sun).
Are the descriptions of the Matlab functions that I use in this project in the appendix D, and the full code in the appendix F and G.

The Matlab notation is:
Function [<list of outputs>] = <name of the function> (<list of inputs>).

XII. DISSCUSION

XII.A. Computational Problems

XII.A.1 Time:

Table 3 size of the files,

	Rows	columns	memory size in doubles (4 Bytes)	size in SVM format in Kb	
Real predictions	67,782	14	948,948	-	
testing unfilter w = 1	22594	13	293722	2053	
testing unfilter w = 3	22594	37	835978	6292	
testing unfilter w = 5	22593	61	1378173	10531	
testing unfilter w = 7	22592	85	1920320	14770	
testing unfilter w = 9	22592	109	2462528	19208	
testing unfilter w = 11	22591	133	3004603	23977	
testing unfilter w = 13	22590	157	3546630	28745	
					% remove
testing filter w = 1	7750	13	100750	727	65.70%
testing filter w = 3	14675	37	542975	4146	35.05%
testing filter w = 5	18461	61	1126121	8671	18.29%
testing filter w = 7	20386	85	1732810	13385	9.76%
testing filter w = 9	21391	109	2331619	18231	5.32%
testing filter w = 11	21891	133	2911503	23268	3.10%
testing filter w = 13	22168	157	3480376	28233	1.87%
samplings w = 1	3427	13	44551	328	
samplings w = 3	7671	37	283827	2133	
samplings w = 5	9594	61	585234	4411	
samplings w = 7	10418	85	885530	6677	
samplings w = 9	10728	109	1169352	8918	
samplings w = 11	10882	133	1447306	11276	
samplings w = 13	10965	157	1721505	13600	

the table describes the size of the files that are used to perform all the experiments with SVM. The procedure to produce the files described in the figure 2.
For testing filter also gives information about how many percentage has been remove when was filtered.

Ther is a direct proportion between the size of the files the SVM use to train and test and the time to trained and make the predictions. When the size of the windows is increased SVM needs more time for the work. I

executed the program in one laptop Pentium M at 1.73GHz with 1 GB of ram memory SVM to make a model (train) and make the predictions one hundred times needs around one day. The time that it takes also depend on the parameters used: see the appendix to know more details about the computational time cost.

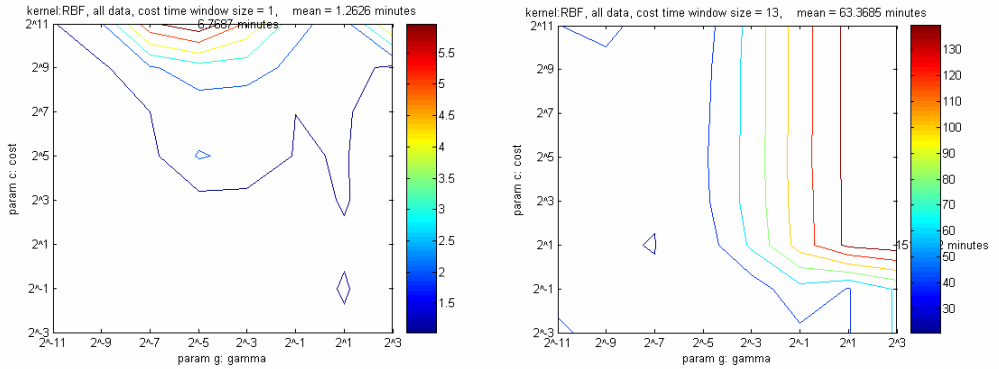


Figure 3: Comparison of the time cost between the experiments for windows size 13 and single data.
The cost (time in seconds) of process each one experiment (training SVM and testing the machine with three different files). The processing cost depends strongly on the value of the parameters. For single run all the experiments need it 80 minutes. For windows data 13 it was taking 4000 minutes; 50 times more than for single data. For windows size 13 the parameter gamma affects more the time than the parameter c.

XII.A.2: Memory optimization:

I use two different modules for make a consistent data:
Ffilted.m:
Fast filter, is based in the algorithm quick sort $\Theta n * \log(n)$, where n is the number of rows or examples.

Filted.m:
Slow filter, is based in lineal searching $\Theta n\text{^}2$,

The module ffilted works well for windows size smaller than size eleven, for size eleven it throws a memory error. So the slow module is needed to filter windowed data larger than nine (for w = 11 are 133 columns) because it needs less memory than ffilted.m
Anyway for windows size equal to 15 Matlab throws memory errors in the part of SMOTE algorithm.

XII.B: algorithms:

There is a large number of different algorithms in current use to predict binding sites (Sun et al 2005b; Bailey and Elkan 1994; Blanchette and Tompa 2003) and it is difficult to know which of them is the most suitable for this project. The reason to choose these 12 is because I want to improve the previous work (Robinson et al 2006; Sun et al 2005) who choose this 12 algorithms but by using the strategy of windowed data instead of single data.
These algorithms have a different nature (different strategies) and very different performance, some of them have a high rate of false positive predictions others classify near all the inputs like a no binding site. This diversity causes me to use all these 12 algorithms.

XII.C: Programming language and SVM

The project was developed using two different programming languages, Matlab and C++, the reason to use Matlab is because some functions that I used were developed in previous work and it permit develop the program in less time. But Matlab is not as rapid as C++ and use more memory to execute a similar program. This is an important problem as described in the chapter of computational problems.
SVM is in C++ and all the others parts of the program are in Matlab are also free versions of SVM for Matlab in internet (for example: http://asi.insa-rouen.fr/~arakotom/toolbox/index.html). But the reason of use this one (Chang and Lin 2001) is write in C++ and is expected to do the same work in less time; what it is executing SVM is the step where the computer spend the most of the time.

SVM is used because it was demonstrated that it have better accuracy than traditional classifying methods like neural networks (Burges 1998). And in SVM RBF was used kernel because (Chang and Lin 2001) recommend because have less “numerical difficulties” than the others.

XII.D: SMOTE algorithm:

The over-sampling algorithm: it produces a new synthetic data, produce five times more of synthetic examples than the original data. These parameter can have value between 1 and 12 make a balanced data set, because there are 13.76 times more examples of no-binding sites than binding sites in the original file of real data predictions. Knowing this limit I think five was a good decision for produce de data. Anyway optimizing this parameter can be a new task for future work.
Over-sampling: this algorithm reduce removing randomly examples of the majority class for training, it reduce the majority class to 80% of the minority class, for this project it remove more than half of the no-binding sites examples. It is expected that if the training data have more examples of binding sites the machine will be make more false positive errors than if the dataset has more examples of no-binding sites (but it can produce more false negatives errors), this is an to interesting point because in this project is a very important issue to find the parameters that reduce the false positive, because this increases the precision and reduce the false positive rate, but on other hand the objective of the work is even not to find a SVM model which classify all the examples like no binding site, the objective is find a better way to predict binding sites.

XII.E: generating data:

In the process of generating data (see figure 2) the windowed data is divided in two groups one for training and other for testing, and later is filtered in both groups. For windows size equal to one or tree where the process of filtering the training data remove 65% and 35% (see table 3), means that the original file have “too much redundant data”, so it is possible find a large number of inputs that are exactly the same in the training data and in the testing data, and is possible too that one input in the training data have opposite label in the testing data.
The first thing (having the same examples in the training data and testing data) can produce a high accuracy, but the results show that the accuracy when the SVM model is tested with the training data is far high than then is tested with the testing data.
These phenomena can be reduced when the windows size became larger, for example for windows size 13 less than 2% data is removed.
Other way to affront these phenomena can be result if the first step is filter and secondly splitting, but in this case it can be more difficult produce un-filter testing data.

XII.F: The best model

Which is the best model to predict binding sites?
This is the biggest problem to resolve in this work and it is not resolute. The best model and this is completely clear is the model that does not produce any false positive and false negative in the classification, but this model do not exist, and need some algorithm to select which is the best model.

It is difficult find a good mathematical function to identify with is the best model for predicting binding sites using only the confusion matrix. In this work I decided to use “precision * (1-fasle positive rate)”, This function gives very strong importance to reduce the false positives, and try to find the model that do not produce false positives.

Is it easy to select which is the best model if there is only one comparable scalar value. Usually this value is the parameter accuracy, but for this work the dataset is very unbalanced to make the parameter accuracy a bad indicator. So it need it a new parameters for select the best model.
It is difficult to compare two models and say which one is the best in the case of these two models have different characteristics, and each one being better than the other for some feature. Usually when one model makes less false positive errors it makes more false negative errors and so on.
The problem to select which is the best model is not resolute because is not clear what is better if one model that do not make too many false positives but do not detect a large part of binding sites or other algorithm that can detect a most of the binding sites but produce more false positives than the first one algorithm.
For this work the first one (who attempt to not produce false positive) is better, but is not clear in numbers how much better is this feature than the other.

XII.F. Conclusion:

All the algorithms to predict binding sites are far from not making errors.

The graphs show a little improvement of the classification (ratio) when the windows size is increased but I do not find any model which improves all the features. No clear improvement is generating when the parameters c: cost and g: gamma are changed for SVM. Not even by implementing the techniques of cross-validation and windowing data. However the parameter f-measure shows an important improvement when used with windowing data.

This means that is it easy find an improvement for one characteristic but not for all, and for each feature are some of the 12 algorithms that has been use like an input for the SVM how have a better performance than the SVM model.

The new models using windowing data which are not better than the 12 primary algorithms but are not worse even but have different features.

XIII. RESULTS:

Table 4: Metrics predictions of the 12 algorithms used:

Algoritm	1	2	3	4	5	6
Precision	0.2027	0.1046	0.0836	0.0807	0.0970	0.1024
fp_rate	0.1013	0.1832	0.3391	0.3324	0.1892	0.2490
Recall	0.3610	0.2946	0.4261	0.4017	0.2800	0.3908
f_measure	0.2596	0.1544	0.1398	0.1344	0.1441	0.1622
Accuracy	0.8606	0.7815	0.6450	0.6496	0.7748	0.7266
Ncc	0.1990	0.0712	0.0460	0.0369	0.0575	0.0814
precision* (1-fp_rate)	0.1822	0.0854	0.0553	0.0539	0.0786	0.0769

Algoritm	7	8	9	10	11	12
Precision	0.1443	0.78130000	0.0000	0.0687	0.0696	0.1101
fp_rate	0.1637	0.00011077	0.0000	0.3972	0.0181	0.1472
Recall	0.3800	0.00540000	0.0000	0.4033	0.0111	0.2505
f_measure	0.2091	0.01080000	0.0000	0.1174	0.0192	0.1529
Accuracy	0.8054	0.93250000	0.9322	0.5893	0.9230	0.8121
Ncc	0.1419	0.06170000	-0.0018	0.0031	0.0000	0.0719
precision* (1-fp_rate)	0.1207	0.78121346	0.0000	0.0414	0.0683	0.0939

The windows sizes that I use are: 1, 3, 5, 7, 9, 11 and 13.
The training data consist in 44.4% approximate are examples of no binding sites and 55.6% approximate are examples of binding sites.

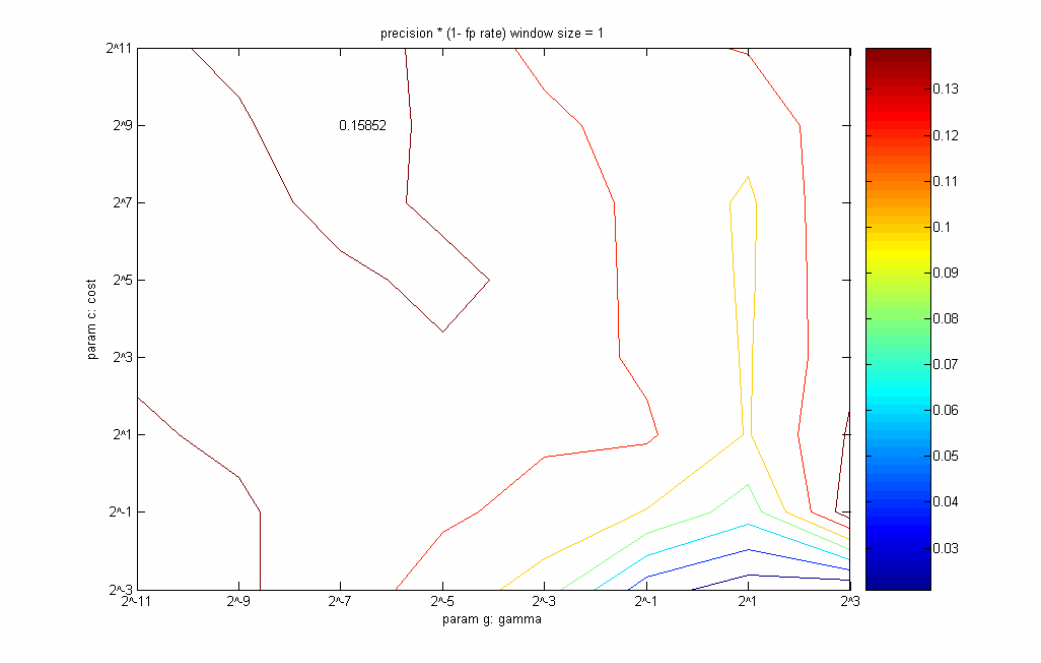


Figure 4
This graph is made it making with 64 different value parameters for the SVM learning and prediction values for single data.
It corresponds with the ratio: precision * (1- fp_rate).
The number 0.15852 corresponds with the maxim value for this ratio, which corresponds for parameter c = 2^9 and g = 2^-7
fp_rate = 0.17504
precision = 0.19216

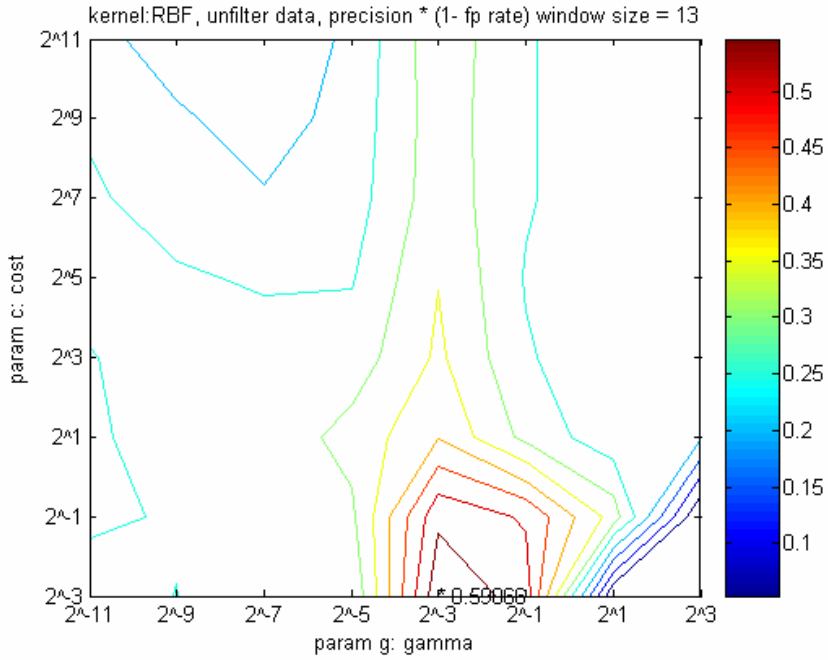
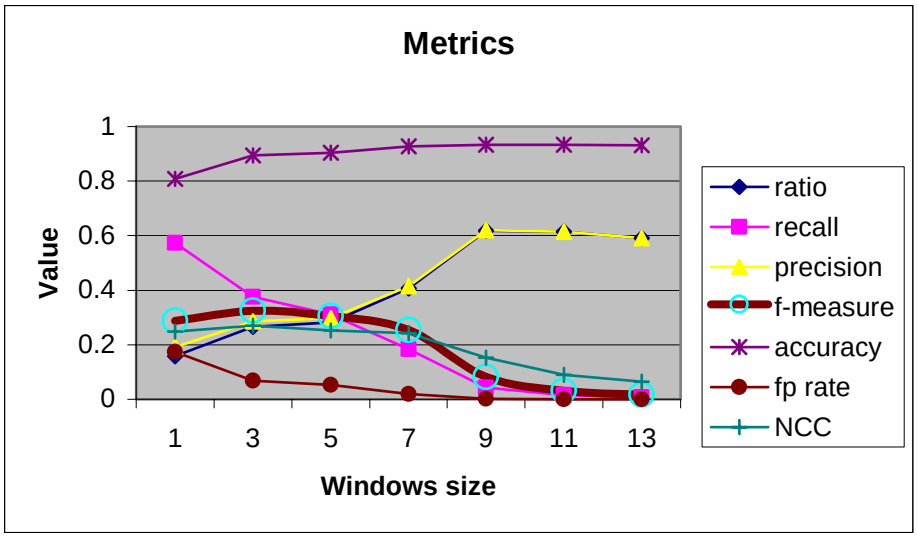
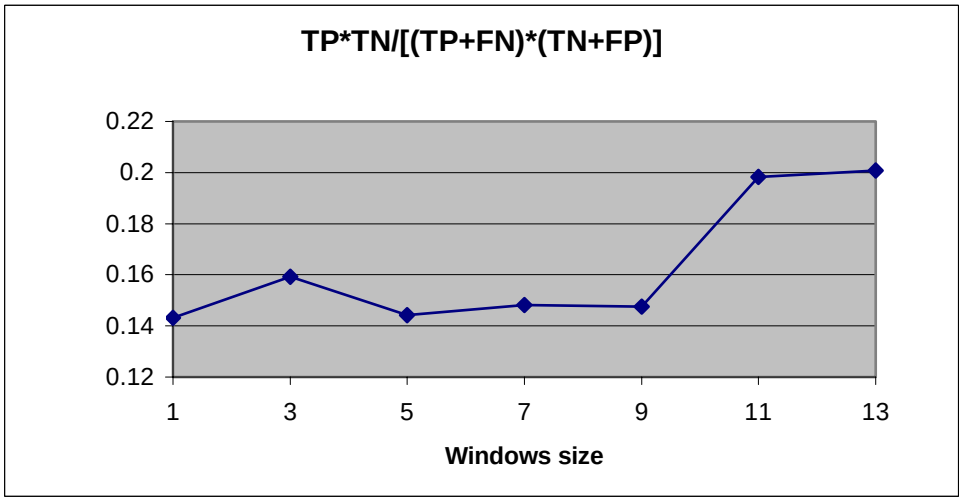


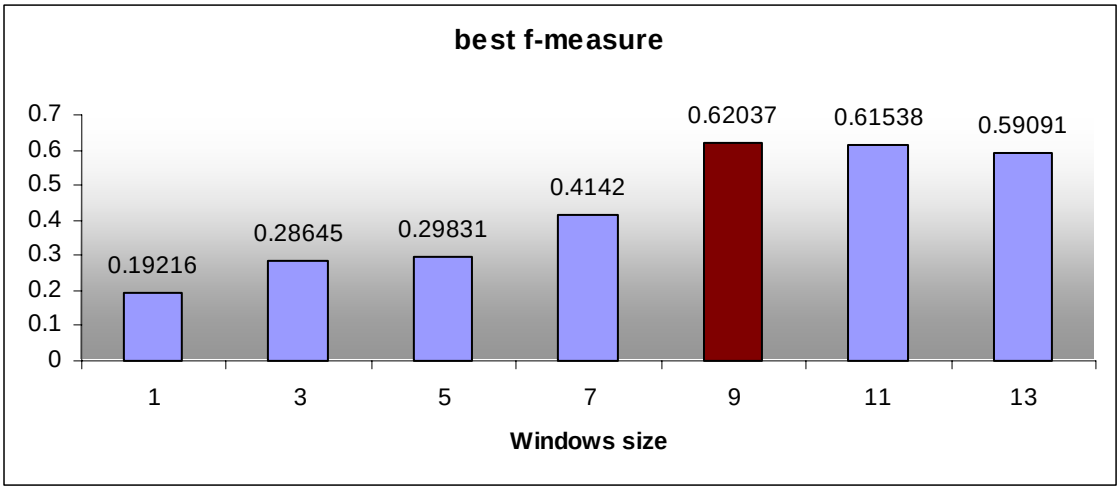
Figure 5
This graph is made it making with 64 different value parameters for SVM learning and prediction values for windowed data windows size 13.
It corresponds with the ratio: precision * (1- fp_rate).
The number 0.059 corresponds with the maxim value for this ratio, which corresponds for parameter c = 2^-3 and g = 2^-3
fp_rate = 0.00042735
precision = 0.59091



Graph 6
This graph shows the evolution of the parameters for the best (the highest ratio: precision*(1-fp rate)) model for each windows size.
The highest ratio is for windows size 9. (value = 0.619)
The better f-measure in these optimization for windows size 3.



Graph 7 TP*TN/[(TP+FN)*(TN+FP)]
This graph represents the mean value for all 64 different experiments for the same windows size.
It shows the proportion of positives that SVM recognise multiply by the proportion of negatives that the machine recognise.
The highest values are for the windows size equal to 11 and 13. Between 1 and 9 the values are similar (but they increased the false positive rate).



Graph 8 evolution of f-measure

This graph represents the best f-measure for each windows size.
All the experiments using windows have a better f-measure than the experiments of single data.
The best (with have the best f-measure) algorithm of the 12 used to train SVM has 0.2596 f-measure. So integrate the predictions using windowed data causes an improvement of the best f-measure, and in these case the best f-measure is for windows size 9 for the parameters cost = 2^-3 and gamma = 2^-1.

Using cross-validation:

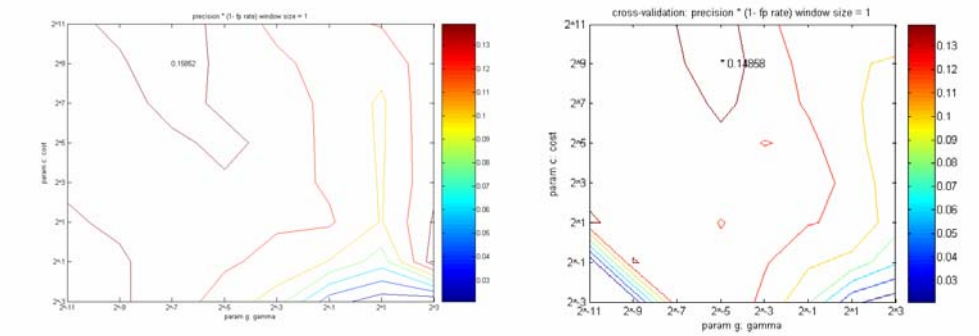


Figure 9 Comparison between using cross-validation (right) and with out cross-validation (left).

This graph is made it making with 64 different value parameters for SVM learning and prediction values for single data.
It corresponds with the ratio: precision * (1- fp_rate).
For single data is not observed any improvement for use cross-validation.
If fact the maxim value rate is decreased 0.15852 to 0.14858.
For cross-validation the parameters:
fp_rate = 0.18653
precision = 0.18265
and for with out cross-validation:
fp_rate = 0.17504
precision = 0.19216

In general cross-validation do not generate any improvement, are more examples about it in the appendix H.

XIV. OBSERVATIONS:

1. For the experiments between 1 and 9for different windows size is detected a perturbation in the map for the value gamma equal to two.
    a. It affects with low precision and high false positive rate when the machine is testing.
    b. It affects with high precision and low false positive rate when the machine is testing with the same data that has been use to trainer.
    c. It gives the best results when the machine is testing with the same data that has been use to trainer for gamma equal to two and the maximum value for cost that has been attempted (2^11).
    d. This effect is not observer when is use cross-validation technique.
    e. It increases the computational time.
2. The graphs for testing with unfilter and filter data.
    a. Gives the same maxim values for the same parameters for windows size equal to tree.
    b. The picture looks the same for windows size equal to five.
    c. I do not check this property using cross-validation.
3. The ratio precision * (1 – false positive rate) is increased as the windows size is increased.
4. I do not observe any improvement for use cross-validation.
5. For windows size larger than 9 when SVM is tested with the training set sampling data are models who have 100% accuracy
6. For windows size 13 the maxim value for precision and minimum false positive rate for un-filer filter and samplings data correspond for the parameters cost = 2^-3 and gamma = 2^-3.
7. Some of the maximums of the metrics parameters are in the value cost = 2^-3,thise value is the treasure of the search grid of SVM, maybe is possible find a better results if is permitted know the results for parameter cost lower than 2^-3

XV. REFERENCES:

(Apostolico et al 2000) Apostolico, A., et al., Efficient detection of unusual words. J Comput Biol, 2000. 7(1-2): p. 71-94.

(Armone and Davidson 1997) m.i. Armone and E. H. Davidson, 'The hardwrig of development: Organization and function of genomic regulatory systems", development 124, 1851-1864, 1997.

(Ayat et al 2002) Ayat N. E, Cheriet M. and Suen C. Y. Optimization of the SVM kernels using an empirical error minimization scheme. In Proc of the international Workshop on Pattern Recognition with Support Vector Machine, pages 354-369, Niagara Falls-Canada, August 2002.

(Bailey and Elkan 1994) Bailey, T.L. and C. Elkan, Fitting a mixture model by expectation maximization to discover motifs in biopolymers. Proc Int Conf Intell Syst Mol Biol, 1994. 2: p. 28-36.

(Bishop 1995) Bishop C. M, Neural Networks for Pattern Recognition, Oxford University Press, 1995

(Blanchette and Tompa 2003) Blanchette, M. and M. Tompa, FootPrinter: A program designed for phylogenetic footprinting. Nucleic Acids Res, 2003. 31(13): p. 3840-2.

(Brown et al. 2002) Brown, C.T., et al., New computational approaches for analysis of cisregulatory networks. Dev Biol, 2002. 246(1): p. 86-102.

(Burges 1998) Burges C. A tutorial on support vector machines for pattern recognition. Data Mining and Knowledge, Discovery, vol 2. no. 2. 1998. p 121-167

(Bussemaker and Siggia 2001) Bussemaker, H. J. H. Li, and E.D. Siggia, Regulatory element detection using correlation with expression. Nat Gent, 2001. 27(2): p. 167-71.

(Chawla et al 2002) Chawla, N.V. et al, SMOTE,: Syntetic minority over-sampling Technique. Journal of Artificial Intelligence Research, 2002, 16: p. 321-357.

(Chang and Lin 2001) Chih-Chung Chang and Chih-Jen Lin LIBSVM : a library for support vector machines, 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm

(Cristianini and Shawe 2000) Cristianini N. and Shawe-Taylor J. An Introduction to Support Vector Machines and other kernel-based learning methods. Cambridge University Press 2000.

(Ha and Bunke 1997) Ha, T.M. and Bunke, H. “Off-line, Handwritten Numeral Recognition by Perturbation Method”, Pattern Analysis and Machine Intelligence, vol 19/5,1997 pp. 535-539

(Hughes et al 2000) Hughes, J.D., et al., Computational identification of cis-regulatory elements associated with groups of functionally related genes in Saccharomyces cerevisiae. J Mol Biol, 2000. 296(5): p. 1205-14.

(Japkowicz 2003) N. Japkowicz, “Class imbalances: Are we focusing on the right issure?” Workshop on learning from imbalanced datasets, II, CML, Washington DC, 2003.

(Keerthi and Lin 2003) S.S. Keerthi and C.-J. Lin. Asymptotic behaviors of support vector machines with Gaussian kernel. Neural Computation 2003. 15 (7), 1667–1689.

(Lin and Lin 2003) H.-T. Lin and C.-J. Lin. A study on sigmoid kernels for SVM and the training of non-PSD kernels by SMO-type methods. Technical report, Department of Computer Science, National Taiwan University. 2003

(Thijs et al 2001) Thijs, G. et al, A higher-order background model improves the detection of promoter regulatory elements by Gibbs sampling. Bioinformatics, 2001, 17(12): p 1113-22.

(Markstein et al 2002) M. Markstein, A. Stathopoulos, V. Markstein, P. Markstein, N. Harafuji, D. Keys, B. Lee, P. Richardson, D. Rokshar and M. Levine, “Decoding Noncoding Regulatory DNAs in Metazoan Genomes”, proceeding of 1st IEEE Computer Society BioinformaticsConference (CSB 2002), Stanford, CA, USA, 14-16, August, 2002.

(Radvojac et al 2004) Radvojac, P. et al, Classification and knowledge discovery in protein databases. J Biomed Inform, 2004. 37(4): p 24-39

(Rajewsky et al 2002) Rajewsky, N, et al, Computational detection of genomic cis-regulatory modules applied to body patterning in the early Drosphila embryo. BMC Bioinformatics, 2002 3(1), p. 30.

(Robinson et al 2006) M.Robinson, Y. Sun, R. Boekhorst, P. Kaye, R. Adams, N. Davey. Improving computational predictions of cis-regulatory binding sites. Pacific symposium on biocomputing 2006, World Scientific Publishing Co. Pte. Ltd

(Tompa et al 2005) Tompa, M. et al, Assessing computational tools for the discovery of transcription factor binding sites. Nat Biotechnol, 2005, 23(1): p. 137-44

(Schölkopf and Smola 2002) B. Scholköpf and A. J. Smola, Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond, The MIT Press, 2002.

(Sun et al 2005) Y. Sun, et al Integrating binding site predictions using non-linear classification methods. In Machine Learning Workshop. 2005. Sheffield: LNAI.

(Sun et al 2005b) Y. Sun, M. Robinson, R. Adams, P. Kaye, A.G. Rust, N. Davey Using Real-valued meta classifiers to integrate binding site predictions. 2005. IJCNN '05. Proceedings. 2005 IEEE International Joint Conference, Neural Networks

(Sun et al 2005c) Y. Sun, M. Robinson, R. Adams, P. Kayes, A. G. Rust and N.Davey, “Integrating binding site predictions using meta classification methods”, Proceedings ICANNGA05, 2005.

(Vapnik 1995). V. Vapnik. The Nature of Statistical Learning Theory. New York, NY: Springer-Verlag. 1995

http://asi.insa-rouen.fr/~arakotom/toolbox/index.html

http://www.hgmp.mrc.ac.uk/Software/EMBOSS/.

http://family.caltech.edu/SeqComp/index.html.

http://rulai.cshl.edu/scpd/

http://www.bch.msu.edu/~kuhn/projects/screening/gst.jpg

XVI. INDEX OF TABLES AND FIGURES

TABLES:

1. Performance Metrics 7
2. The most common kernels 11
3. Size of the files (datasets) 14
4. metrics of the predictions of the 12 algorithms used 20

FIGURES:

1. Windows dataset 6
2. Flux diagram of generating data process 9
3. Comparison of the time cost between the experiments for windows size 13 and single data 15
4. ratio: precision*(1-fp_rate) for single data 21
5. ratio: precision*(1-fp_rate) for windows size 13 21
6. Evolution of the metrics parameters for the best ratio precision*(1-fp_rate) for each windows size 22
7. TP*TN/[(TP+FN)*(TN+FP)] mean values 22
8. Evolution of f-measure 23
9. Comparison about using cross-validation and with out cross-validation 23

APPENDIX A : EXAMPLE OF SPACE LINEAR SEPARABLE (SVM)

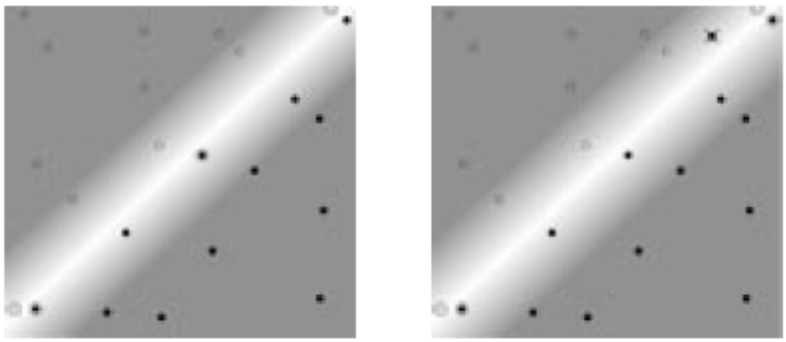


Figure 1 : black points and grey points represents two different classes. The left picture represents a linearly separable case, and the right show non-separable case. The white line represents the hyper-plane. (Burges 1998)

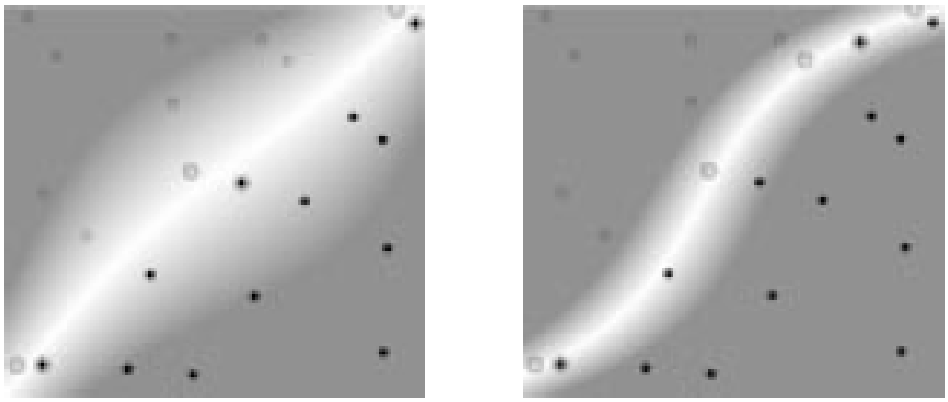


Figure 2 : black points and grey points represents two different classes. The wait line represents the hyper-plane. This is a SVM solution for separate in two groups using a polynomial kernel. (Burges 1998)

APPENDIX B: Illustration of SVM kernel mapping.

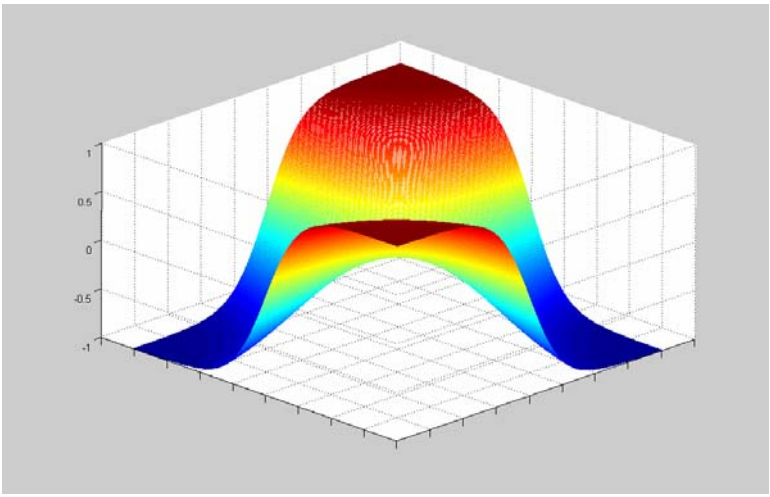


Figure 1 : Sigmoid kernel mapping 2D into 3D x and y goes between -2 and 2 Formula: K (X,Y) = tanh(a*x*y+b); for a = 1 and b = 0;

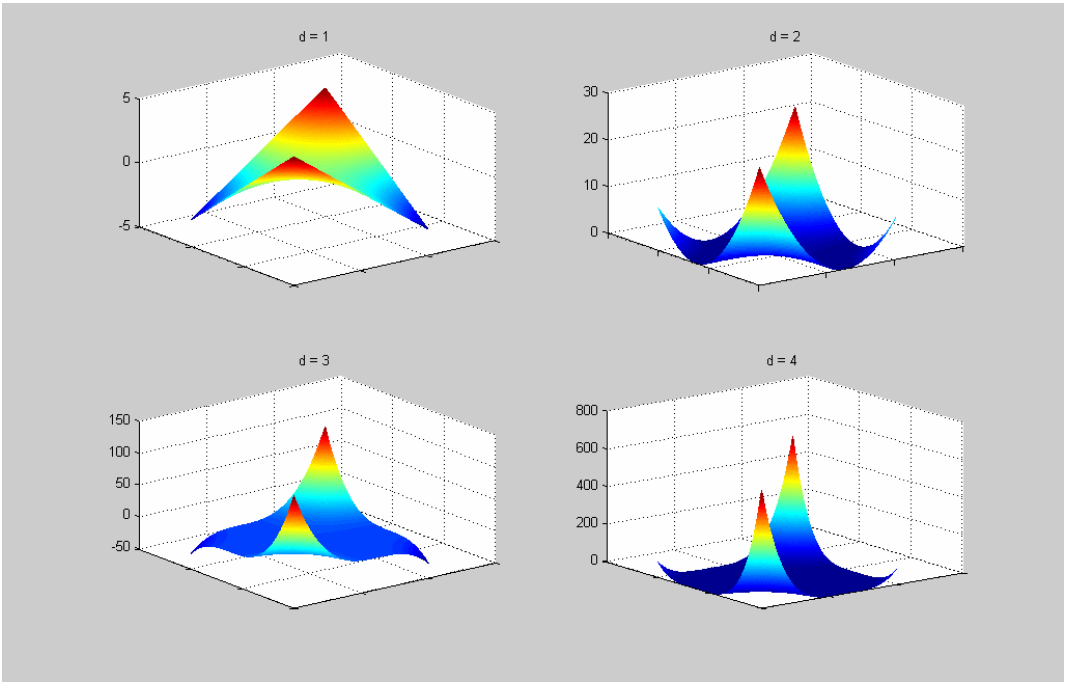


Figure 2 : polynomial kernel mapping 2D into 3D x and y goes between -2 and 2 Formula: K (X,Y) = (1+x*y)^d; for d = 1 , 2, 3, and 4

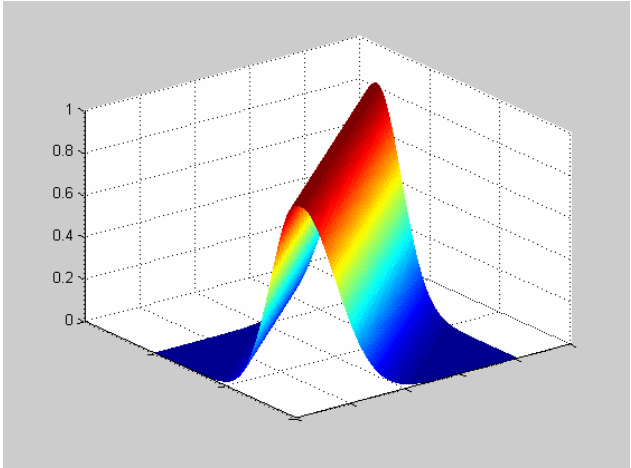


Figure 3 : BRF kernel mapping 2D into 3D x and y goes between -2 and 2
Formula: K (X,Y) = exp(-(g*abs(x-y)^2)) ; for g = 1

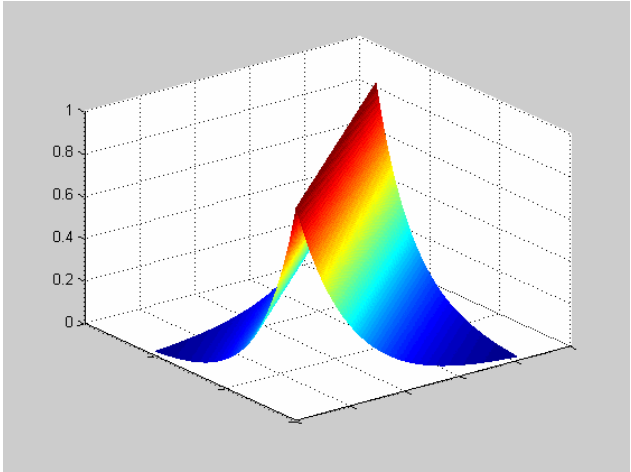


Figure 4 : exponential RBF kernel mapping 2D into 3D
x and y goes between -2 and 2
Formula: K (X,Y) = `exp(-(g*abs(x-y)))`; g = 1

APPENDIX C: PSEUDO CODE OF SMOTE ALGORITHM

Algorithm *SMOTE*(T, N, k)
Input: Number of minority class samples T; Amount of SMOTE $N\%$; Number of nearest neighbors k
Output: $(N/100) * T$ synthetic minority class samples

```
1.  (* If N is less than 100%, randomize the minority class samples as
    only a random percent of them will be SMOTEd. *)
2.  if N < 100
3.     then Randomize the T minority class samples
4.          T = (N/100) * T
5.          N = 100
6.  endif
7.  N = (int)(N/100) (* The amount of SMOTE is assumed to be in
    integral multiples of 100. *)
8.  k = Number of nearest neighbors
9.  numattrs = Number of attributes
10. Sample[ ][ ]: array for original minority class samples
11. newindex: keeps a count of number of synthetic samples generated,
    initialized to 0
12. Synthetic[ ][ ]: array for synthetic samples
    (* Compute k nearest neighbors for each minority class sample only. *)
13. for i ← 1 to T
14.         Compute k nearest neighbors for i, and save the indices in
            the nnarray
15.         Populate(N, i, nnarray)
16. endfor

    Populate(N, i, nnarray) (* Function to generate the synthetic sam-
    ples. *)
17. while N ≠ 0
18.         Choose a random number between 1 and k, call it nn. This
            step chooses one of the k nearest neighbors of i.
19.         for attr ← 1 to numattrs
20.                 Compute: dif = Sample[nnarray[nn]][attr]−Sample[i][
21.                 Compute: gap = random number between 0 and 1
22.                 Synthetic[newindex][attr] = Sample[i][attr] + gap *
                    dif
23.         endfor
24.         newindex++
25.         N = N − 1
26. endwhile
27. return (* End of Populate. *)
    End of Pseudo-Code.
```

from (Chawla et al 2002)

APPENDIX D : DESCRIPTION OF THE MODULES

---

function [recall, precision, f_measure, accuracy, fp_rate, nCC] = metrics(C);
This function works for 2 class confusion matrix
This function returns the metrics parameters for one confusion matrix.
Input:
Confusion matrix [TN, FP; FN TN];
Output:
recall
precision
f_measure
accuracy
fp_rate
nCC

---

function n2 = dist2_var(x, c, Var)
takes two matrices of vectors and calculates the squared Euclidean distance between them. Is required for make over sampling (I do not implemented this module)
DIST2 Calculates squared distance between two sets of points.
Description
D = DIST2(X, C) takes two matrices of vectors and calculates the squared Euclidean distance between them. Both matrices must be of the same column dimension. If X has M rows and N columns, and C has
L rows and N columns, then the result has M rows and L columns. The I, Jth entry is the squared distance from the Ith row of X to the Jth row of C.

Copyright (c) Ian T Nabney (1996-2001)

---

function lin = line2lines(line_)
This function translate one sentence to a set of sentence:
for example X = 'Dear Ann\n I ...'
P = line2lines(X)
P(1).s
ans =
Dear Ann
Input:
Line: type: string
Output:

Lines: set of strings.

---

function X = quitpoint(Y)
This function helps to translate a file in SVM format to a file in Maltab format:
The input Y have the following format:
<label> <index> ":" <value>.
This function scans the input string and when finds the symbol ":" replace the semicolon and the last two characters (who correspond with the index) for tree space characters.
Later the new format is: (output)
<label> <value1> <value2>...

---

function [NewMatrix] = ffilted(Matrix);

This function filter the matrix: remove repeated vectors and remove inconsistent vectors: vectors how have for the same input different label. This algorithm use quick short for develop the task

Label mast been at the end at the last column in the matrix.

---

function New = filted(Matrix);

This function filter the matrix: remove repeated vectors and remove inconsistent vectors: vectors how have for the same input different label. This algorithm use do not use quick short for develop the task and works slowly than "ffilted", but use less memory, that is the reason why is need it because for the biggest windows size (11 and 13) "ffilted" throws memory errors.

Label mast been at the first column in the matrix (the opposite as "ffilted").

Script: hacer_foto
This script helps to make graphs to show the results.

---

function hd = hvdm(data, centre, Var_real, D_vdm)

This is a function to calculate the Heterogeneous value difference metric distance

---

script: make6
these script give a combination of parameters to svm to make the experiments and stored information about confusion matrix and metrics of the experiment, the parameters that has been used: parameter cost, gamma and with file has been use to test (testing un-filter, testing filter or training file, and the windows size), and also information about how much time has been used to develop the task.
It use RBF kernel for all the experiments
parameters used:
param_c = [2^-3,2^-1,2^1,2^3,2^5,2^7,2^9,2^11]
param_g = [2^-11,2^-9,2^-7,2^-5,2^-3,2^-1,2^1,2^3]
param_w = [1:2:21]

---

function confusion = make_confusion_matrix(prediction,label);

This function compare the label and the prediction of one algorithm and return the confusion matrix.

---

function
[results_k_layer,results_in_testing_filted,results_in_testing_unfilted] =
make_crossvalidation3
(Name_File,testing_file_unfilted,testing_file_filted,crossvalidation_laye
rs,Paramiters)
Paramiters:
Name of data file
Name of the testing file unfilter
Name of the testing file filter
Value for crosvalidation, number of subsets
list of paramiters for SVM

Return:
function
[results_k_layer,results_in_testing_filted,results_in_testing_unfilted] =
make_crossvalidation make_crossvalidation3

(Name_File,testing_file_unfilted,testing_file_filted,crossvalidation_laye
rs,Paramiters)

---

script: make lineal
these script give a combination of parameters to svm to make the experiments and stored information about confusion matrix and metrics of the experiment, the parameters that has been used: parameter cost, gamma and with file has been use to test (testing un-filter, testing filter or training file, and the windows size), and also information about how much time has been used to develop the task.
It use lineal kernel for all the experiments
parameters used:
param_c = 2.^(-5:11);
param_w = [1:2:13]

---

script: make_polinomial
these script give a combination of parameters to svm to make the experiments and stored information about confusion matrix and metrics of the experiment, the parameters that has been used: parameter cost, gamma and with file has been use to test (testing un-filter, testing filter or training file, and the windows size), and also information about how much time has been used to develop the task.
It use polynomial kernel for all the experiments (degree = 3)
parameters used:
param_c = [2^-3,2^-1,2^1,2^3,2^5,2^7,2^9,2^11]
param_g = [2^-11,2^-9,2^-7,2^-5,2^-3,2^-1,2^1,2^3]
param_w = [1:2:13]

---

script: make_sigmoidal
these script give a combination of parameters to svm to make the experiments and stored information about confusion matrix and metrics of the experiment, the parameters that has been used: parameter cost, gamma and with file has been use to test (testing un-filter, testing filter or training file, and the windows size), and also information about how much time has been used to develop the task.
It use sigmoidal kernel for all the experiments
parameters used:
param_c = [2^-3,2^-1,2^1,2^3,2^5,2^7,2^9,2^11]
param_g = [2^-11,2^-9,2^-7,2^-5,2^-3,2^-1,2^1,2^3]
param_w = [1:2:13]

---

function make_windows_data_new (Windows,Name_File)
This function produce the training and testing data for a certain windows size and file.
Inputs:
Windows: size of the windows.
Name_file: Original file (real predictions)
Outputs: do not have, because the function saves the results into a file.

Given the size of the windows and the name of the data file, generate a new data file with the windowed data: this means if the data file have 12 inputs + 1 output = 13 columns for windows size 3 the new file will be have 12*3 + 1 output = 37 columns and 2 rows less than the original file because the first line and the last one only have one neighbour so are remove it.

---

script: mappcross6
these script give a combination of parameters to svm to make the experiments using cross-validation with 10 subsets and stored information about confusion matrix and metrics of the experiment, the parameters that has been used: parameter cost, gamma and with file has been use to test (testing un-filter, testing filter or training file, and the windows size), and also information about how much time has been used to develop the task.
It use RBF kernel for all the experiments
param_c = [2^-3,2^-1,2^1,2^3,2^5,2^7,2^9,2^11]
param_g = [2^-11,2^-9,2^-7,2^-5,2^-3,2^-1,2^1,2^3]
param_w = [1:2:3]
crossvalidation_layers = 10

---

script runwindows.
is for generate windows data

---

function save_in_svm_format(Matrix,File)
The SVM software need a different format of the information, need the label in the first column and for each input the number of the column and double point: all in ASCII code. (Chang and Lin 2001)
<label> <index>:<value1> <index2>:<value2>...
...
...
<label> is the target value of the training data. (is supported multi classification.
<index> mast be in ascendant order.
Input:
Matrix: label mast be in the first column.
File: <string> the name of the file.
Output: do not have

---

function results = seematrixmatrix (Matrix)
(Label,Output)
Paramiters:
vector of targets (or labels)
vector of Outputs (predictions)

Return: confusion matrix
'The variable results:
[recall; precision; f_measure; accuracy; fp_rate; nCC;']

---

function [NewMatrix] = sortmatrix(Matrix);
These function sort one matrix in alphabetical pre-order.
Input:
Matrix
Output:
NewMatrix.

---

function d = vdm(data, labels)
Value difference metric (VDM)
From paper <<Reduction Techniques for Instance-Based learning Algorithms>>, authors: D. R. Wilson and T. R. Martinez
Machine Learning, 38-3, pp. 257-286 (2000)

attr1: the column with attribute 1
x: indicating value 1
y: indicating value -1
labels: patterns' lables

---

function [LabelW, MatrixW] = Window_ing (TamWindows,Labels,MatrixSingle);

These function produce a un-filter windows data.
Input:
TamWindows: Windows size.
Labels: column with the labels.
MatrixSingle: Matrix with the inputs vectors.
Output
LabelW = Labels for the new windowed data.
MatrixW = New inputs vectors.

---

function [newdata, target] = oversampling_RB_window(data, K, N, width);

This is a modified version of SMOTE: the distances are computed using HVDM and the discrete values are randomly choosen from neighbours
_RB_ means this is a function for mixed data sets
This function is based on SMOTE algorithm

data an imbalanced ndata x ndim matrix;
K Number of nearest neighbours;
newdata new dataset
target labels of the new dataset
N amount of SMOTE, if N>K, then some samples are
width the size of a window
interpoting more than once.

(This module was implemented by Yi Sun)

APPENDIX E: MORE DETAILS ABOUT THE RESULTS

For RBF kernel:

Evolution of the matrix for the best (ratio: precision*(1-false_positive_rate)) model tested in un-filter data for different windows size:

Best w	1	3	5	7	9	11	13
ratio	0.1585	0.2661	0.2824	0.4064	0.6192	0.6149	0.5907
recall	0.5732	0.3752	0.3118	0.183	0.0438	0.0157	0.0085
precision	0.1922	0.2856	0.2983	0.4142	0.6204	0.6154	0.5909
f-measure	0.2878	0.3243	0.3049	0.2539	0.0818	0.0306	0.0168
accuracy	0.8079	0.8941	0.9037	0.9271	0.9334	0.9327	0.9324
fp rate	0.175	0.0682	0.0533	0.0188	0.0019	0.0007	0.0004
NCC	0.2492	0.2709	0.2533	0.2422	0.1524	0.0906	0.065

Evolution of the matrix for the mean value for all the models tested in un-filter data for each different windows size:

Best w	1	3	5	7	9	11	13
ratio	0.1185	0.165072	0.156333	0.1755	0.18947	0.2455	0.24632
recall	0.614	0.5414	0.5772	0.532	0.5349	0.4311	0.4281
precision	0.1592	0.2058	0.205	0.2226	0.241	0.2788	0.2789
f-measure	0.2498	0.281	0.2801	0.2847	0.2831	0.2785	0.2706
accuracy	0.7356	0.7845	0.7501	0.7711	0.7692	0.85	0.8523
fp rate	0.2556	0.1979	0.2374	0.2115	0.2138	0.1195	0.1168
NCC	0.2098	0.2362	0.2358	0.2384	0.2441	0.2502	NaN

Evolution of the matrix for the median value for all the models tested in un-filter data for each different windows size:

Best w	1	3	5	7	9	11	13
ratio	0.1279	0.189356	0.19564	0.2122	0.22125	0.248	0.24729
recall	0.5863	0.517	0.532	0.4588	0.4863	0.4693	0.4908
precision	0.1632	0.2153	0.224	0.2369	0.2454	0.271	0.2708
f-measure	0.2543	0.2952	0.2982	0.3067	0.3111	0.3293	0.3419
accuracy	0.7681	0.8548	0.8479	0.8678	0.8731	0.886	0.8852
fp rate	0.216	0.1205	0.1266	0.1044	0.0984	0.0847	0.0868
NCC	0.2112	0.2561	0.2544	0.2588	0.2644	0.2884	0.302

Differences in the parameters with the neighbours values of windows size:

	1-3	3-5	5-7	7-9	9-11	11-13
recall	0.07	-0.02	0.07	-0.03	0.02	-0.02
precision	-0.05	-0.01	-0.01	-0.01	-0.03	0.00
f-measure	-0.04	-0.00	-0.01	-0.00	-0.02	-0.01
accuracy	-0.09	0.01	-0.02	-0.01	-0.01	0.00
fp rate	0.10	-0.01	0.02	0.01	0.01	-0.00
NCC	-0.04	0.00	-0.00	-0.01	-0.02	-0.01

Media of percentage of good classification for each windows size:

Good classification	1	3	5	7	9	11	13
Positives	0.1486	0.1658	0.1501	0.1545	0.1538	0.2076	0.2102
Negatives	0.9637	0.9601	0.9613	0.9587	0.9588	0.9552	0.9551
Area	0.1432	0.159185	0.14429	0.1481	0.14746	0.1983	0.20076

Where positives is the same as recall and area is % of positives * % of negatives

APPENDIX F: SVM-MATRIX.C,

is a modification of svmpredict.c

```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include "svm.h"

char* line;
int max_line_len = 1024;
struct svm_node *x;
int max_nr_attr = 64;

struct svm_model* model;
int predict_probability=0;

void predict(FILE *input, FILE *output)
{
	int correct = 0;
	int total = 0;
	double error = 0;
	double sumv = 0, sumy = 0, sumvv = 0, sumyy = 0, sumvy = 0;
	double TP=0,TN=0,FP=0,FN=0, recall, precision;

	int svm_type=svm_get_svm_type(model);
	int nr_class=svm_get_nr_class(model);
	int *labels=(int *) malloc(nr_class*sizeof(int));
	double *prob_estimates=NULL;
	int j;

	if(predict_probability)
	{
		if (svm_type==NU_SVR || svm_type==EPSILON_SVR)
			printf("Prob. model for test data: target value = predicted
value + z,\nz: Laplace distribution e^(-
|z|/sigma)/(2sigma),sigma=%g\n",svm_get_svr_probability(model));
		else
		{
			svm_get_labels(model,labels);
			prob_estimates = (double *)
malloc(nr_class*sizeof(double));
			fprintf(output,"labels");
			for(j=0;j<nr_class;j++)
				fprintf(output," %d",labels[j]);
			fprintf(output,"\n");
		}
	}
	while(1)
	{
		int i = 0;
		int c;
		double target,v;

		if (fscanf(input,"%lf",&target)==EOF)
```

```
                break;

            while(1)
            {
                if(i>=max_nr_attr-1)        // need one more for index = -1
                {
                    max_nr_attr *= 2;
                    x = (struct svm_node *)
realloc(x,max_nr_attr*sizeof(struct svm_node));
                }

                do {
                    c = getc(input);
                    if(c=='\n' || c==EOF) goto out2;
                } while(isspace(c));
                ungetc(c,input);
                fscanf(input,"%d:%lf",&x[i].index,&x[i].value);
                ++i;
            }

out2:
            x[i++].index = -1;

            if (predict_probability && (svm_type==C_SVC ||
svm_type==NU_SVC))
            {
                v = svm_predict_probability(model,x,prob_estimates);
                fprintf(output,"%g ",v);
                for(j=0;j<nr_class;j++)
                    fprintf(output,"%g ",prob_estimates[j]);
                fprintf(output,"\n");
            }
            else
            {
                v = svm_predict(model,x);
                fprintf(output,"%g\n",v);
            }

            if(v == target)
                ++correct;
            if(v<0.5)
            {
                if (target<0.5)
                    TN++;
                else   FN++;
            }
            else{
                if (target>=0.5)
                    TP++;
                else   FP++;
            }
            //printf("predict %g \n",v);

            error += (v-target)*(v-target);
            sumv += v;
            sumy += target;
```

```
                sumvv += v*v;
                sumyy += target*target;
                sumvy += v*target;
                ++total;
        }
 /*
        printf("Accuracy = %g%% (%d/%d) (classification)\n",
                (double)correct/total*100,correct,total);
        printf("Mean squared error = %g (regression)\n",error/total);
        printf("Squared correlation coefficient = %g (regression)\n",
                (double)((total*sumvy-sumv*sumy)*(total*sumvy-sumv*sumy))/
                ((total*sumvv-sumv*sumv)*(total*sumyy-sumy*sumy))
                );

        recall = TP/(TP+FN);
        precision = TP /(TP + FP);
        printf("Recall = %g \n",recall);
        printf("Precision = %g \n", precision);
        printf("F-Score = %g \n", (2*recall*precision)/(recall+precision));
        printf("FP-rate = %g \n", FP/(FP+TN));
        printf("Confusion Matrix: \nTP: %g \nTN: %g \nFP: %g \nFN: %g
\n",TP,TN,FP,FN);
 */
    printf("%g %g  \n %g  %g", TN, FP, FN, TP);
        if(predict_probability)
        {
                free(prob_estimates);
                free(labels);
        }
}

void exit_with_help()
{
        printf(
        "Usage: svm-predict [options] test_file model_file output_file\n"
        "options:\n"
        "-b probability_estimates: whether to predict probability estimates, 0 or
1 (default 0); one-class SVM not supported yet\n"
        );
        exit(1);
}

int main(int argc, char **argv)
{
        FILE *input, *output;
        int i;

        // parse options
        for(i=1;i<argc;i++)
        {
                if(argv[i][0] != '-') break;
                ++i;
                switch(argv[i-1][1])
                {
                        case 'b':
                                predict_probability = atoi(argv[i]);
```

```
                        break;
                default:
                        fprintf(stderr,"unknown option\n");
                        exit_with_help();
                }
        }
        if(i>=argc)
                exit_with_help();

        input = fopen(argv[i],"r");
        if(input == NULL)
        {
                fprintf(stderr,"can't open input file %s\n",argv[i]);
                exit(1);
        }

        output = fopen(argv[i+2],"w");
        if(output == NULL)
        {
                fprintf(stderr,"can't open output file %s\n",argv[i+2]);
                exit(1);
        }

        if((model=svm_load_model(argv[i+1]))==0)
        {
                fprintf(stderr,"can't open model file %s\n",argv[i+1]);
                exit(1);
        }

        line = (char *) malloc(max_line_len*sizeof(char));
        x = (struct svm_node *) malloc(max_nr_attr*sizeof(struct svm_node));
        if(predict_probability)
                if(svm_check_probability_model(model)==0)
                {
                        fprintf(stderr,"Model does not support probabiliy
estimates\n");
                        exit(1);
                }
        predict(input,output);
        svm_destroy_model(model);
        free(line);
        free(x);
        fclose(input);
        fclose(output);
        return 0;
}
```

APPENDIX G: IMPLEMENTATIONS OF THE MODULES IN MATLAB:

```
function n2 = dist2_var(x, c, Var)
%DIST2  Calculates squared distance between two sets of points.
%
%   Description
%   D = DIST2(X, C) takes two matrices of vectors and calculates the
%   squared Euclidean distance between them.  Both matrices must be of
%   the same column dimension.  If X has M rows and N columns, and C
      has
%   L rows and N columns, then the result has M rows and L columns.
      The
%   I, Jth entry is the  squared distance from the Ith row of X to the
%   Jth row of C.
%
%   See also
%   GMMACTIV, KMEANS, RBFFWD
%

%   Copyright (c) Ian T Nabney (1996-2001)

[ndata, dimx] = size(x);
[ncentres, dimc] = size(c);
if dimx ~= dimc
    error('Data dimension does not match dimension of centres')
end

n2 = (ones(ncentres, 1) * sum(((x.^2)./(repmat(16*Var, ndata,1)))',
      1))' + ...
  ones(ndata, 1) * sum(((c.^2)./(repmat(16*Var, ncentres,1)))',1) -
      ...
  2.*(x./(repmat(16*Var, ndata,1))*(c'));

% Rounding errors occasionally cause negative entries in n2
if any(any(n2<0))
  n2(n2<0) = 0;
end
```

```
function [NewMatrix] = ffilted(Matrix);
%
% This function filter the matrix: remove repeated vectors and remove
% inconsistent vectors: vectors how have for the same input different
      label.
% This algorithm use quick short for develop the task
%
% Label mast be at the end at the last column in the matrix.

columns = size(Matrix,2);
rows = size(Matrix,1);

M = [Matrix, (1:rows)'];

MM = sortmatrix(M);

pos = 1:rows;

pivote = MM(1,1:end-2);
```

```
cont = 1;

for i = pos
    if sum(MM(i,1:end-2) ~= pivote) == 0

        continue
    else

        labels = MM(cont:i-1,end-1);
        ss = unique(labels);
        val = size(ss,1);

        if val == 1
            MM(cont+1:i-1,end) = MM(cont+1:i-1,end)*0;
        elseif val == 2
            MM(cont:i-1,end) = MM(cont:i-1,end)*0;
        elseif val > 0
            error('error in ffilted');
        else
            warning('some group in ffilted void');
        end

        cont = i;
        pivote = MM(i,1:end-2);
    end
end

    labels = MM(cont:i,end-1);
    val = unique(labels);
    val = size(val,1);

    if val == 1
        MM(cont+1:i,end) = MM(cont+1:i,end)*0;
    elseif val == 2
        MM(cont:i,end) = MM(cont:i,end)*0;
    elseif val > 0
        error('error in ffilted');
    else
        warning('some group in ffilted void');
    end



posaux = MM(:,end);

ceritos = (posaux ~= 0);
a = find (ceritos);

pos = sort(posaux(a,:));


NewMatrix = Matrix(pos,:);
```

```
function New = filted(Matrix);
% This function filter the matrix: remove repeated vectors and
% remove inconsistent vectors: vectors how have for the same
% input different label.
%
% This algorithm use do not use quick short for develop the
% task and works slowly than “ffilted”, but use less memory,
% that is the reason why is need it because for the biggest
% windows size (11 and 13) “ffilted” throws memory errors.
%
% Label mast been at the first column in the matrix
% (the opposite as “ffilted”).


rows = size(Matrix,1);
cols = size(Matrix,2);

New = Matrix;

for i = 1:rows

    for j = i+1:rows-1
        i;
        j;
        if (sum(abs(Matrix(i,2:cols)-Matrix(j,2:cols)))==0)
            %if is the same input
            if real(Matrix(i,1)) == real(Matrix(j,1))
                % same tarject
                New(j,1) = New(j,1)+sqrt(-1);
            else
                New(j,1) = New(j,1)+sqrt(-1);
                New(i,1) = New(i,1)+sqrt(-1);
            end
        end
    end
end

disp('second phase');

%for i = 1:rows
%    if imag(New(i,1)) ~= 0
%        break
%    end
%end

%j = -1;
j = 0;
i = 1;
while (imag(New(i,1)) ~= 0)
    i = i +1;
    j = j -1;
end

for i = i:rows
    if imag(New(i,1)) ~= 0
        j = j-1;
    else

        New(i+j,:) = New(i,:);
    end
end
```

```
%rows+j;
New = New(1:rows+j+1,:);
```

```
% Script: hacer_foto
% These script helps to make graphs to show the results.

ventanita = 1;
disp(['window size = ' , num2str(ventanita)]);
%texto = 'kernel:RBF, unfilter data,'
%texto = 'kernel:RBF, filter data,'
texto = 'kernel:RBF, sampling data,'
%texto = 'kernel:RBF, all data,'
%ventanita = 1;
pivote = 8*8;
v = floor(ventanita/2);

pos = find(results_unfilted(end,:)==ventanita);

%pos = 1+v*pivote:pivote*(1+v);

%r = results_unfilted(:,pos);
%r = results_filted(:,pos);
r = results_samplings(:,pos);

%ventanita = 13;



time = time_need_it(:,pos);
q = [];

for i = 0:7
    %q = [ r(6,8*i+1:8*(i+1));q];
    q = [q; r(6,8*i+1:8*(i+1))];
end

q1 = [];

for i = 0:7
    %q1 = [ r(9,8*i+1:8*(i+1));q1];
    q1 = [ q1; r(9,8*i+1:8*(i+1))];
end


qf = [];

for i = 0:7
    %q1 = [ r(9,8*i+1:8*(i+1));q1];
    qf = [ qf; r(7,8*i+1:8*(i+1))];
end


%%%q(1,1) = 100
counter = 0;
q2 = q .* (1-q1);
q3 = -(q .* q1);
size(q2)
```

```
figure(1)
set(gcf, 'color', [1,1,1]);
contour(q2);
%pcolor(q2);
colorbar
%cross-validation:
title([texto, ' precision * (1- fp rate) window size = ',
      num2str(ventanita)]);
xlabel('param g: gamma');
ylabel('param c: cost');
text(find(sum(q2)==sum(sum(q2))),find(sum(q2')==sum(sum(q2))),'max');
set(gca,'XTickLabel',{'2^-11','2^-9','2^-7','2^-5','2^-3','2^-
      1','2^1','2^3'})
set(gca,'YTickLabel',{'2^-3','2^-
      1','2^1','2^3','2^5','2^7','2^9','2^11'})
tope = max(max(q2));
aux = (q2 == tope);
for x = 1:size(aux,1);
for y = 1:size(aux,2);
    counter = counter+1;
    if aux(x,y) == 1


        text(y,x,['* ',num2str(tope)])
        disp(['* ',num2str(tope)]);
        disp(['fp_rate   = ', num2str(r(9,counter))]);
        disp(['precision = ', num2str(r(6,counter))]);

    end
end
end

figure(2)
set(gcf, 'color', [1,1,1]);
contour(q);
colorbar
title([texto,  ' precision window size = ', num2str(ventanita)]);
xlabel('param g: gamma');
ylabel('param c: cost');
text(find(sum(q)==sum(sum(q))),find(sum(q')==sum(sum(q))),'max')
set(gca,'XTickLabel',{'2^-11','2^-9','2^-7','2^-5','2^-3','2^-
      1','2^1','2^3'})
set(gca,'YTickLabel',{'2^-3','2^-
      1','2^1','2^3','2^5','2^7','2^9','2^11'})
tope = max(max(q));
aux = (q == tope);
for x = 1:size(aux,1);
for y = 1:size(aux,2);
    if aux(x,y) == 1
    text(y,x,num2str(tope))
    end
end
end

figure(3)
set(gcf, 'color', [1,1,1]);
contour(q1);
colorbar
title([texto, ' fpRate window size = ', num2str(ventanita)]);
```

```
xlabel('param g: gamma');
ylabel('param c: cost');
text(find(sum(q1)==sum(sum(q1))),find(sum(q1')==sum(sum(q1))),'max')
set(gca,'XTickLabel',{'2^-11','2^-9','2^-7','2^-5','2^-3','2^-
      1','2^1','2^3'})
set(gca,'YTickLabel',{'2^-3','2^-
      1','2^1','2^3','2^5','2^7','2^9','2^11'})
tope = min(min(q1));
aux = (q1 == tope);
for x = 1:size(aux,1);
for y = 1:size(aux,2);
    if aux(x,y) == 1
    text(y,x,num2str(tope))
    end
end
end

figure(4)
set(gcf, 'color', [1,1,1]);
contour(q3);
colorbar
title([texto,' - (precision * fpRate )  window size = ',
      num2str(ventanita)]);
xlabel('param g: gamma');
ylabel('param c: cost');

set(gca,'XTickLabel',{'2^-11','2^-9','2^-7','2^-5','2^-3','2^-
      1','2^1','2^3'})
set(gca,'YTickLabel',{'2^-3','2^-
      1','2^1','2^3','2^5','2^7','2^9','2^11'})

tope = max(max(q3));
aux = (q3 == tope);
for x = 1:size(aux,1);
for y = 1:size(aux,2);
    if aux(x,y) == 1
    text(y,x,num2str(tope))
    end
end
end

t = [];
for i = 0:7
    t = [t; time(1,8*i+1:8*(i+1))];
end
figure(5)
set(gcf, 'color', [1,1,1]);
contour(t/60);
colorbar

xlabel('param g: gamma');
ylabel('param c: cost');

set(gca,'XTickLabel',{'2^-11','2^-9','2^-7','2^-5','2^-3','2^-
      1','2^1','2^3'})
set(gca,'YTickLabel',{'2^-3','2^-
      1','2^1','2^3','2^5','2^7','2^9','2^11'})

tope = max(max(t));
media = mean(mean(t))
```

```
title([texto, ' cost time window size = ', num2str(ventanita), ',
      mean = ' num2str(media/60) ' minutes']);
aux = (t == tope);
for x = 1:size(aux,1);
for y = 1:size(aux,2);
    if aux(x,y) == 1
    text(y,x,[num2str(tope/60),' minutes '])
    end
end
end


figure(6)
set(gcf, 'color', [1,1,1]);
contour(q);
colorbar
title([texto,  ' f-measure window size = ', num2str(ventanita)]);
xlabel('param g: gamma');
ylabel('param c: cost');
text(find(sum(q)==sum(sum(q))),find(sum(q')==sum(sum(q))),'max')
set(gca,'XTickLabel',{'2^-11','2^-9','2^-7','2^-5','2^-3','2^-
      1','2^1','2^3'})
set(gca,'YTickLabel',{'2^-3','2^-
      1','2^1','2^3','2^5','2^7','2^9','2^11'})
tope = max(max(q));
aux = (q == tope);
for x = 1:size(aux,1);
for y = 1:size(aux,2);
    if aux(x,y) == 1
    text(y,x,num2str(tope))
    end
end
end

figure(5)
figure(4)
figure(3)
figure(2)
figure(1)
figure(6)
```

```
function hd = hvdm(data, centre, Var_real, D_vdm)

%
% This is a function to calculate the Heterogeneous value difference
      metric
% distance
%

  % computing the Euclidean distance:
  n2 = dist2_var(centre(:, 6:12), data(:,6:12), Var_real);

  nvdm = zeros(1,length(data));
  d_v =[];
  for col = 1:2
      d_a = ones(1, length(data));
      if centre(:,col)~=0
         ind_1 = find(data(:, col)~=centre(:, col));
```

```
            d_a(ind_1) = D_vdm(col);
            ind_2 = find(data(:, col)==centre(:, col));
            d_a(ind_2)= 0;
            ind_3 = find(data(:,col)==0);
            d_a(ind_3) = 1;
        end
        sq_d_a = d_a.*d_a;
        d_v =[d_v; sq_d_a];
    end
    nvdm = sum(sq_d_a, 1);
    hd = sqrt(n2 + nvdm);
```

```
function lin = line2lines(line_)
% This function translate one sentence to a set of sentence:
% for example X = 'Dear Ann\n I ...'
% P = line2lines(X)
% P(1).s
% ans =
%   Dear Ann
% Input:
%   Line: type: string
% Output:
%   Lines: set of strings.

if (size(line_,1)>1)
    line_ = line_'
end
ini = 1;
final = 1;
count = 1;

while (final < size(line_,2))
    if(line_(final)==10)
        lin(count).s= line_(ini:final-2);
        ini = final+1;
        count = count +1;
    end

    final = final+1;
end
lin(count).s= line_(ini:final);
```

```
% these script give a combination of parameters to svm to make the
% experiments and stored information about confusion matrix and
      metrics
% of the experiment, the parameters that has been used: parameter
      cost,
% gamma and with file has been use to test (testing un-filter, testing
% filter or training file, and the windows size), and also information
% about how much time has been used to develop the task.
% It use RBF kernel for all the experiments
% parameters used:
% param_c = [2^-3,2^-1,2^1,2^3,2^5,2^7,2^9,2^11]
% param_g = [2^-11,2^-9,2^-7,2^-5,2^-3,2^-1,2^1,2^3]
% param_w = [1:2:21]



%script
param_c = [2^-3,2^-1,2^1,2^3,2^5,2^7,2^9,2^11]
param_g = [2^-11,2^-9,2^-7,2^-5,2^-3,2^-1,2^1,2^3]
param_w = [1]


readme = '[confusion(:,1);confusion(:,2);recall; precision; f_measure;
      accuracy; fp_rate; nCC; -c parameter; -g parameter; name of the
      data file]';

if ~exist('ultimo') || ultimo == 0

    ultimo = 0;
    results_unfilted = [];
    results_filted = [];
    results_samplings = [];
    time_need_it = [];
else
    disp('PROCES LOADED');
    ultimo
    pause(2);
end

time0 = clock;
moverse = ultimo;
pos = 1;

for  w_ = param_w
for  c_ = param_c
for  g_ = param_g
       %% For permiting continue.

        if moverse > 0
            moverse = moverse -1;
            continue
        end
        time1 = clock;
        ultimo = ultimo +1;

        aux_ = ['svmtrain -s 0 -b 1 -c ' , num2str(c_) , ' -t 2 -g ' ,
      num2str(g_) , ' -e 0.00001  svm_training_samplings_w' ,
      num2str(w_) , '.dat  x.model'];
        disp(aux_);
        %[A] = system(aux_);
```

```
  [A,B] = system(aux_);

  pause(2);

  if (A == 1) eval('stop_program_1'); end
  pause(2)

  seconds = etime(clock,time1);
  disp(['testing  ', num2str(seconds/60)]);
  aux_ = ['svm-matrix svm_windows_testing_unfilted_w' ,
num2str(w_),'.dat  x.model y.output'];
  %disp(aux_);
  [A,B]= system(aux_);
  if (A == 1) eval('stop_program_2');end
  confusion = str2num(B);
  [recall, precision, f_measure, accuracy, fp_rate, nCC] =
metrics(confusion);
  results_unfilted = [ results_unfilted ,
[confusion(:,1);confusion(:,2);recall; precision; f_measure;
accuracy; fp_rate; nCC; c_; g_; w_]] ;
  system('del y.output');

  aux_ = ['svm-matrix svm_windows_testing_filted_w' ,
num2str(w_),'.dat  x.model y.output'];
  %disp(aux_);
  [A,B]= system(aux_);
  if (A == 1) eval('stop_program_3');end
  confusion = str2num(B) ;
  [recall, precision, f_measure, accuracy, fp_rate, nCC] =
metrics(confusion);
  results_filted = [ results_filted ,
[confusion(:,1);confusion(:,2);recall; precision; f_measure;
accuracy; fp_rate; nCC; c_; g_; w_]] ;
  system('del y.output');

  aux_ = ['svm-matrix svm_training_samplings_w' ,
num2str(w_),'.dat  x.model y.output'];
  [A,B]= system(aux_);
  if (A == 1) eval('stop_program_4');end
  confusion = str2num(B);
  [recall, precision, f_measure, accuracy, fp_rate, nCC] =
metrics(confusion);
  results_samplings = [ results_samplings ,
[confusion(:,1);confusion(:,2);recall; precision; f_measure;
accuracy; fp_rate; nCC; c_; g_; w_]];



  seconds = etime(clock,time1);
  time_need_it = [time_need_it , seconds];

  disp (['step: ', num2str(ultimo) ,' w= ' ,num2str(w_), ' has
been needit ' num2str(seconds/60) ' minutes. Runing time = '
num2str(etime(clock,time0)/3600) ' hours ']);
  disp('.');


save('6results.mat','results_unfilted','results_filted','results
_samplings','time_need_it','ultimo','readme');
```

```
      save('6results_copy.mat','results_unfilted','results_filted','re
      sults_samplings','time_need_it','ultimo','readme');

        system('del y.output');
        system('del x.model');


end
end
readme
end
```

```
% script make6cross
% these script give a combination of parameters to svm using
% cross-validation k = 10, to make the experiments and stored
% information about confusion matrix and metrics of the experiment,
% the parameters that has been used: parameter cost, gamma only use
% unfilter and filter file to for testing. , and also information
      about
% how much time has been used to develop the task.
% It use RBF kernel for all the experiments


param_w = [1:2:21]

Paramiters = ' -s 0 -b 1 -c 2^-1 -t 2 -g 2^-7';



if ~exist('ultimo') || ultimo == 0
    ultimo = 0;
    time_need_it = [];
else
    disp('PROCES LOADED');
    ultimo
    pause(2);
end

time0 = clock;
moverse = ultimo;
pos = 1;

for  w_ = param_w;

        if moverse > 0
            moverse = moverse -1;
            continue
        end
        time1 = clock;
        ultimo = ultimo +1;
```

```
    Name_File = ['svm_training_samplings_w' , num2str(w_) , '.dat'];
    testing_file_unfilted = ['svm_windows_testing_unfilted_w' ,
      num2str(w_),'.dat'];
    testing_file_filted = ['svm_windows_testing_filted_w' ,
      num2str(w_),'.dat'];
    crossvalidation_layers = 10;

    [CROSS,unfilted,filted] =
      make_crossvalidation2(Name_File,testing_file_unfilted,testing_fi
      le_filted,crossvalidation_layers,Paramiters);

    windows(w_ ).cross = CROSS;
    windows(w_ ).unfilted = unfilted;
    windows(w_ ).filted = filted;

    seconds = etime(clock,time1);
    time_need_it = [time_need_it , seconds];
    save('crosvalidation');
    save('crosvalidation2');
    disp (['step: ', num2str(ultimo) ,' w= ' ,num2str(w_), ' has been
      needit ' num2str(seconds/60) ' minutes. Runing time = '
      num2str(etime(clock,time0)/3600) ' hours ']);
    disp('.');
end
```

```
function confusion = make_confusion_matrix(prediction,label);
% make_confusion_matrix(prediction,label);
% This function compare the label and the prediction of one algorithm
% and return the confusion matrix.
TN = 0;
FP = 0;
FN = 0;
TP = 0;

for i = 1:size(prediction,1)
    if label(i) > 0
        if prediction(i) > 0
            TP = TP+1;
        else
            FN = FN +1;
        end
    else
        if prediction(i) > 0
            FP = FP +1;
        else
            TN = TN +1;
        end
    end
end

confusion = [TN FP ; FN TP];
```

```
function
      [results_k_layer,results_in_testing_filted,results_in_testing_un
      filted] = make_crossvalidation3
      (Name_File,testing_file_unfilted,testing_file_filted,crossvalida
      tion_layers,Paramiters)
% Paramiters:
%       Name of data file
%       Name of the testing file unfilter
%       Name of the testing file filter
%       Value for crosvalidation, number of subsets
%       list of paramiters for SVM
%
% Return:
%       function
      [results_k_layer,results_in_testing_filted,results_in_testing_un
      filted] =
%       make_crossvalidation make_crossvalidation3
%
      (Name_File,testing_file_unfilted,testing_file_filted,crossvalida
      tion_layers,Paramiters)



results_k_layer = [];
results_in_testing_filted = [];
results_in_testing_unfilted = [];

c_matrix = zeros(2,2);
fid = fopen(Name_File,'r');

if (fid < 1 )

    Name_File
    fid
    error('error in file')
end

text = fread(fid,'uint8=>char')';
fclose(fid);

text2 = line2lines(text);

long_text = max(size(text2));



for (turn = 1:crossvalidation_layers)
    disp(['turn = ', num2str(turn)]);
    begin = floor((turn -1)/crossvalidation_layers*long_text)+1;
    final = floor(turn/crossvalidation_layers*long_text);

    fid2 = fopen([num2str(turn),'at.dat'],'w');
    if (fid2 < 1 )
        error('error in file2')
    end

    fid3 = fopen([num2str(turn),'ates.dat'],'w');
   if (fid3 < 1 )
        error('error in file2')
    end
```

```
    first1 = 1;
    fff2 = 1;

    for (i = 1:long_text)
        if( i>begin && i< final )
            if first1 == 1
                fwrite(fid3,text2(i).s);
                first1 = 0;
            else
                fwrite(fid3,[,13, 10 ,'  ',  text2(i).s]);
            end


        else
            %text2(i).s
            if fff2 == 1
                %i
                fwrite(fid2, text2(i).s);
                fff2 == 0;
            else
                fwrite(fid2,['' ,13 ,10,' ', text2(i).s]);
            end
            fff2 = 0;

        end
    end


    fclose(fid2);
    fclose(fid3);
    OK = 0;


    exe = ['svmtrain ', Paramiters , ' ', num2str(turn),'at.dat
      ',num2str(turn),'at.model '];
    [OK,B] = system(exe);

    if (OK == 1)
        error('error');
    end
    %exe = ['svmpredict  ates.dat  at.model at.output'];
    %[A,B]= system(exe)

    exe = ['svm-matrix ' num2str(turn),'ates.dat ',
      num2str(turn),'at.model ',num2str(turn),'at.output '];
    %exe = ['svm-matrix ', Name_File ,' at.model at.output'];
    [OK,B]= system(exe);
    if (OK == 1) eval('stop_program');end

    confusion = str2num(B);
    [recall, precision, f_measure, accuracy, fp_rate, nCC] =...
        metrics(confusion);
    results_k_layer =
      [results_k_layer,[confusion(:,1);confusion(:,2);...
        recall; precision; f_measure; accuracy; fp_rate; nCC;turn]];
end

precision = results_k_layer(6);
fp_rate = results_k_layer(9);
```

```
res = precision .* (1 - fp_rate);
tope = max(res);
posicion = find (tope == res);
posicion = posicion(1,1); %only one;


exe = ['svm-matrix ', testing_file_unfilted ,'   ', num2str(posicion),
      'at.model tes.output '];
[OK,B]= system(exe);
if (OK == 1) eval('stop_program2');end
confusion = str2num(B);
[recall, precision, f_measure, accuracy, fp_rate, nCC] = ...
    metrics(confusion);
results_in_testing_unfilted =
      [results_in_testing_unfilted,[confusion(:,1);...
    confusion(:,2);recall; precision; f_measure; accuracy;...
    fp_rate; nCC;turn]];


exe = ['svm-matrix ', testing_file_filted ,'  ', num2str(posicion),
      'at.model tes2.output '];
[OK,B]= system(exe);
if (OK == 1) eval('stop_program2');end
confusion = str2num(B);
[recall, precision, f_measure, accuracy, fp_rate, nCC] = ...
    metrics(confusion);
results_in_testing_filted =
      [results_in_testing_filted,[confusion(:,1);...
    confusion(:,2);recall; precision; f_measure; accuracy;...
    fp_rate; nCC;turn]];
```

```
% script: make lineal
% these script give a combination of parameters to svm to make the
% experiments and stored information about confusion matrix and
      metrics
% of the experiment, the parameters that has been used: parameter
      cost,
% gamma and with file has been use to test (testing un-filter, testing
% filter or training file, and the windows size), and also information
% about how much time has been used to develop the task.
% It use lineal kernel for all the experiments
% parameters used:
% param_c = 2.^(-5:11);
% param_w = [1:2:13]


param_c = 2.^(-5:11);
param_w = 1:2:13;


readme = '[confusion(:,1);confusion(:,2);recall; precision; f_measure;
      accuracy; fp_rate; nCC; -c parameter; -g parameter; name of the
      data file]';
```

```
if ~exist('ultimo_lineal') || ultimo_lineal == 0

    ultimo_lineal = 0;
    results_unfilted = [];
    results_filted = [];
    results_samplings = [];
    time_need_it = [];
else
    disp('PROCES LOADED');
    ultimo_lineal
    pause(2);
end

time0 = clock;
moverse = ultimo_lineal;
pos = 1;
g_ = NaN
for  w_ = param_w
for  c_ = param_c
       %% For permiting continue.

       if moverse > 0
           moverse = moverse -1;
           continue
       end
       time1 = clock;
       ultimo_lineal = ultimo_lineal +1;

       aux_ = ['svmtrain -s 0 -b 1 -c ' , num2str(c_) , ' -t 0 -e
      0.00001  svm_training_samplings_w' , num2str(w_) , '.dat
      lineal.model'];
       disp(aux_);
       %[A] = system(aux_);
       [A,B] = system(aux_);



       if (A == 1) eval('stop_program_1'); end


       seconds = etime(clock,time1);
       disp(['testing  ', num2str(seconds/60)]);
       aux_ = ['svm-matrix svm_windows_testing_unfilted_w' ,
      num2str(w_),'.dat  lineal.model lineal.output'];

       [A,B]= system(aux_);
       if (A == 1) eval('stop_program_2');end
       confusion = str2num(B);
       [recall, precision, f_measure, accuracy, fp_rate, nCC] =
      metrics(confusion);
       results_unfilted = [ results_unfilted ,
      [confusion(:,1);confusion(:,2);recall; precision; f_measure;
      accuracy; fp_rate; nCC; c_; g_ ; w_]] ;
       system('del lineal.output');

       aux_ = ['svm-matrix svm_windows_testing_filted_w' ,
      num2str(w_),'.dat  lineal.model lineal.output'];

       [A,B]= system(aux_);
       if (A == 1) eval('stop_program_3');end
       confusion = str2num(B) ;
```

```
        [recall, precision, f_measure, accuracy, fp_rate, nCC] =
      metrics(confusion);
        results_filted = [ results_filted ,
      [confusion(:,1);confusion(:,2);recall; precision; f_measure;
      accuracy; fp_rate; nCC; c_; g_; w_]] ;
        system('del lineal.output');

        aux_ = ['svm-matrix svm_training_samplings_w' ,
      num2str(w_),'.dat  lineal.model lineal.output'];
        [A,B]= system(aux_);
        if (A == 1) eval('stop_program_4');end
        confusion = str2num(B);
        [recall, precision, f_measure, accuracy, fp_rate, nCC] =
      metrics(confusion);
        results_samplings = [ results_samplings ,
      [confusion(:,1);confusion(:,2);recall; precision; f_measure;
      accuracy; fp_rate; nCC; c_; g_; w_]];



        seconds = etime(clock,time1);
        time_need_it = [time_need_it , seconds];

        disp (['step: ', num2str(ultimo_lineal) ,' w= ' ,num2str(w_),
      ' has been needit ' num2str(seconds/60) ' minutes. Runing time =
      ' num2str(etime(clock,time0)/3600) ' hours ']);
        disp('.');


      save('lineal_kernel.mat','results_unfilted','results_filted','re
      sults_samplings','time_need_it','ultimo_lineal','readme');

      save('lineal_kernel_copy.mat','results_unfilted','results_filted
      ','results_samplings','time_need_it','ultimo_lineal','readme');

        system('del lineal.output');
        system('del lineal.model');
end
readme
end
```

```
% script: make_polinomial
% these script give a combination of parameters to svm to make the
% experiments and stored information about confusion matrix and
% metrics of the experiment, the parameters that has been used:
% parameter cost, gamma and with file has been use to test (testing
% un-filter, testing filter or training file, and the windows size),
% and also information about how much time has been used to develop
% the task.
% It use polynomial kernel for all the experiments (degree = 3)
% parameters used:
% param_c = [2^-3,2^-1,2^1,2^3,2^5,2^7,2^9,2^11]
% param_g = [2^-11,2^-9,2^-7,2^-5,2^-3,2^-1,2^1,2^3]
% param_w = [1:2:13]



param_c = [2^-3,2^-1,2^1,2^3,2^5,2^7,2^9,2^11]
param_g = [2^-11,2^-9,2^-7,2^-5,2^-3,2^-1,2^1,2^3]
param_w = [1:2:13]


readme = '[confusion(:,1);confusion(:,2);recall; precision; f_measure;
      accuracy; fp_rate; nCC; -c parameter; -g parameter; name of the
      data file]';

if ~exist('ultimo_polinomial') || ultimo_polinomial == 0

    ultimo_polinomial = 0;
    results_unfilted = [];
    results_filted = [];
    results_samplings = [];
    time_need_it = [];
else
    disp('PROCES LOADED');
    ultimo_polinomial
    pause(2);
end

time0 = clock;
moverse = ultimo_polinomial;
pos = 1;

for  w_ = param_w
for  c_ = param_c
for  g_ = param_g
       %% For permiting continue.

        if moverse > 0
            moverse = moverse -1;
            continue
        end
        time1 = clock;
        ultimo_polinomial = ultimo_polinomial +1;

        aux_ = ['svmtrain -s 0 -b 1 -c ' , num2str(c_) , ' -t 1 -g ' ,
      num2str(g_) , ' -e 0.00001  svm_training_samplings_w' ,
      num2str(w_) , '.dat  poli.model'];
        disp(aux_);
        %[A] = system(aux_);
        [A,B] = system(aux_);
```

```
  pause(2);

  if (A == 1) eval('stop_program_1'); end
  pause(2)

  seconds = etime(clock,time1);
  disp(['testing  ', num2str(seconds/60)]);
  aux_ = ['svm-matrix svm_windows_testing_unfilted_w' ,
num2str(w_),'.dat  poli.model poli.output'];
  %disp(aux_);
  [A,B]= system(aux_);
  if (A == 1) eval('stop_program_2');end
  confusion = str2num(B);
  [recall, precision, f_measure, accuracy, fp_rate, nCC] =
metrics(confusion);
  results_unfilted = [ results_unfilted ,
[confusion(:,1);confusion(:,2);recall; precision; f_measure;
accuracy; fp_rate; nCC; c_; g_; w_]] ;
  system('del poli.output');

  aux_ = ['svm-matrix svm_windows_testing_filted_w' ,
num2str(w_),'.dat  poli.model poli.output'];
  %disp(aux_);
  [A,B]= system(aux_);
  if (A == 1) eval('stop_program_3');end
  confusion = str2num(B) ;
  [recall, precision, f_measure, accuracy, fp_rate, nCC] =
metrics(confusion);
  results_filted = [ results_filted ,
[confusion(:,1);confusion(:,2);recall; precision; f_measure;
accuracy; fp_rate; nCC; c_; g_; w_]] ;
  system('del poli.output');

  aux_ = ['svm-matrix svm_training_samplings_w' ,
num2str(w_),'.dat  poli.model poli.output'];
  [A,B]= system(aux_);
  if (A == 1) eval('stop_program_4');end
  confusion = str2num(B);
  [recall, precision, f_measure, accuracy, fp_rate, nCC] =
metrics(confusion);
  results_samplings = [ results_samplings ,
[confusion(:,1);confusion(:,2);recall; precision; f_measure;
accuracy; fp_rate; nCC; c_; g_; w_]];



  seconds = etime(clock,time1);
  time_need_it = [time_need_it , seconds];

  disp (['step: ', num2str(ultimo_polinomial) ,' w= '
,num2str(w_), ' has been needit ' num2str(seconds/60) ' minutes.
Runing time = ' num2str(etime(clock,time0)/3600) ' hours ']);
  disp('.');


save('polinomial_results.mat','results_unfilted','results_filted
','results_samplings','time_need_it','ultimo_polinomial','readme
');

save('polinomial_results_copy.mat','results_unfilted','results_f
```

```
      ilted','results_samplings','time_need_it','ultimo_polinomial','r
      eadme');

        system('del poli.output');
        system('del poli.model');



end
end
readme
end
```

```
% script: make_sigmoidal
% these script give a combination of parameters to svm to make the
% experiments and stored information about confusion matrix and
% metrics of the experiment, the parameters that has been used:
% parameter cost, gamma and with file has been use to test (testing
% un-filter, testing filter or training file, and the windows size),
% and also information about how much time has been used to develop
% the task.
% It use sigmoidal kernel for all the experiments
% parameters used:
% param_c = [2^-3,2^-1,2^1,2^3,2^5,2^7,2^9,2^11]
% param_g = [2^-11,2^-9,2^-7,2^-5,2^-3,2^-1,2^1,2^3]
% param_w = [1:2:13]



param_c = [2^-3,2^-1,2^1,2^3,2^5,2^7,2^9,2^11]
param_g = [2^-11,2^-9,2^-7,2^-5,2^-3,2^-1,2^1,2^3]
param_w = [1:2:21]


readme = '[confusion(:,1);confusion(:,2);recall; precision; f_measure;
      accuracy; fp_rate; nCC; -c parameter; -g parameter; name of the
      data file]';

if ~exist('ultimo_sigmoid') || ultimo_sigmoid == 0

    ultimo_sigmoid = 0;
    results_unfilted = [];
    results_filted = [];
    results_samplings = [];
    time_need_it = [];
else
    disp('PROCES LOADED');
    ultimo_sigmoid
    pause(2);
end

time0 = clock;
moverse = ultimo_sigmoid;
pos = 1;
```

```
for  w_ = param_w
for  c_ = param_c
for  g_ = param_g
       %% For permiting continue.

       if moverse > 0
           moverse = moverse -1;
           continue
       end
       time1 = clock;
       ultimo_sigmoid = ultimo_sigmoid +1;

       aux_ = ['svmtrain -s 0 -b 1 -c ' , num2str(c_) , ' -t 3 -g ' ,
     num2str(g_) , ' -e 0.00001  svm_training_samplings_w' ,
     num2str(w_) , '.dat  sigmoid.model'];
       disp(aux_);
       %[A] = system(aux_);
       [A,B] = system(aux_);

       pause(2);

       if (A == 1) eval('stop_program_1'); end
       pause(2)

       seconds = etime(clock,time1);
       disp(['testing  ', num2str(seconds/60)]);
       aux_ = ['svm-matrix svm_windows_testing_unfilted_w' ,
     num2str(w_),'.dat  sigmoid.model sigmoid.output'];
       %disp(aux_);
       [A,B]= system(aux_);
       if (A == 1) eval('stop_program_2');end
       confusion = str2num(B);
       [recall, precision, f_measure, accuracy, fp_rate, nCC] =
     metrics(confusion);
       results_unfilted = [ results_unfilted ,
     [confusion(:,1);confusion(:,2);recall; precision; f_measure;
     accuracy; fp_rate; nCC; c_; g_; w_]] ;
       system('del sigmoid.output');

       aux_ = ['svm-matrix svm_windows_testing_filted_w' ,
     num2str(w_),'.dat  sigmoid.model sigmoid.output'];
       %disp(aux_);
       [A,B]= system(aux_);
       if (A == 1) eval('stop_program_3');end
       confusion = str2num(B) ;
       [recall, precision, f_measure, accuracy, fp_rate, nCC] =
     metrics(confusion);
       results_filted = [ results_filted ,
     [confusion(:,1);confusion(:,2);recall; precision; f_measure;
     accuracy; fp_rate; nCC; c_; g_; w_]] ;
       system('del sigmoid.output');

       aux_ = ['svm-matrix svm_training_samplings_w' ,
     num2str(w_),'.dat  sigmoid.model sigmoid.output'];
       [A,B]= system(aux_);
       if (A == 1) eval('stop_program_4');end
       confusion = str2num(B);
       [recall, precision, f_measure, accuracy, fp_rate, nCC] =
     metrics(confusion);
```

```
        results_samplings = [ results_samplings ,
      [confusion(:,1);confusion(:,2);recall; precision; f_measure;
      accuracy; fp_rate; nCC; c_; g_; w_]];



        seconds = etime(clock,time1);
        time_need_it = [time_need_it , seconds];

        disp (['step: ', num2str(ultimo_sigmoid) ,' w= ' ,num2str(w_),
      ' has been needit ' num2str(seconds/60) ' minutes. Runing time =
      ' num2str(etime(clock,time0)/3600) ' hours ']);
        disp('.');


      save('polinomial_results.mat','results_unfilted','results_filted
      ','results_samplings','time_need_it','ultimo_sigmoid','readme');

      save('polinomial_results_copy.mat','results_unfilted','results_f
      ilted','results_samplings','time_need_it','ultimo_sigmoid','read
      me');

        system('del sigmoid.output');
        system('del sigmoid.model');



end
end
readme
end
```

```
function make_windows_data_new (Windows,Name_File)
% function make_windows_data_new (Windows,Name_File)
% This function produce the training and testing data for a certain
      windows size and file.
% Inputs:
%   Windows: size of the windows.
%   Name_file: Original file  (real predictions)
% Outputs: do not have, because the function saves the results into a
      file.


if (mod(Windows,2)== 0)
    error('error in windows size')
end
if (mod(Windows,1)~= 0)
    error('error in param windows mast be Natural');
end
if (Windows<1)
    error('param Windows mast be positive');
end

load limpio
matrix = data;

randn('state', 714);
```

```
rand('state', 714);


[Label,Matrix] = Window_ing(Windows,data(:,13),data(:,1:end-1));

windows_unfilted = [Matrix,Label];

%disp('saving unfilted');
%name = ['save windows_data_unfilted_w', num2str(Windows),'.dat' ];
%exe = [name, '  windows_unfilted -ascii '];
%disp (exe);
%eval(exe);

%disp('saving unfilted svm');
%save_in_svm_format(windows_unfilted,['svm_windows_data_unfilted_w',nu
      m2str(Windows),'.dat']);

%[name,'windows_unfilted']
%save(name,'windows_unfilted');


windows_unfilted =  [windows_unfilted(:,end),windows_unfilted(:,1:end-
      1)];

disp('spliting');


binding = find(windows_unfilted(:,1)== 1);
nobinding = find(windows_unfilted(:,1)== -1);


binding = windows_unfilted(binding,:);
nobinding = windows_unfilted(nobinding,:);

rows_binding = size(binding,1)
rows_nobinding = size(nobinding,1)


binding = binding(randperm(rows_binding),:);
nobinding = nobinding(randperm(rows_nobinding),:);

binding_trn = binding(1:floor(rows_binding*2/3),:);
binding_tes = binding((floor(rows_binding*2/3)+1):end,:);

nobinding_trn = nobinding(1:floor(rows_nobinding*2/3),:);
nobinding_tes = nobinding((floor(rows_nobinding*2/3)+1):end,:);

%size(binding_trn)
%size(nobinding_trn)
%pause(2000)
training_data = [binding_trn;nobinding_trn];
size_trn = size(training_data,1);

testing_data = [binding_tes; nobinding_tes];
size_tes =  size(testing_data,1);


training_data = training_data(randperm(size_trn),:);

testing_data =   testing_data(randperm(size_tes),:);
disp('.');
```

```
% size(find(training_data(:,1)==1),1)
% size(find(training_data(:,1)==-1),1)
% disp('kaka')
% pause(200)




disp('saving unfilted training svm');
save_in_svm_format(testing_data,['svm_windows_testing_unfilted_w',...
    num2str(Windows),'.dat']);




 disp('filting training');

 %windows_filted = filted(windows_unfilted);

% training_filted =
      ffilted([training_data(:,2:end),training_data(:,1)]);
 training_filted = filted([training_data]);%ojo con el orde que cambia
%training_filted = training_filted(:,[end,1:end-1]);


 size_trn_filted = size(training_filted,1)
 training_filted = training_filted(randperm(size_trn_filted),:);



 %size(find(training_filted(:,1)==1),1)
 %size(find(training_filted(:,1)==-1),1)


 %disp('kaka')
 %pause(200)

 disp('filting testing');

 %testing_filted = ffilted([testing_data(:,2:end),testing_data(:,1)]);
 testing_filted = filted([testing_data]); %ojo con el orden
 %testing_filted = testing_filted(:,[end,1:end-1]);

 size_tes_filted = size(testing_filted,1)
 testing_filted = testing_filted(randperm(size_tes_filted),:);


 pause(5)
 %windows_filted = windows_unfilted;
 %%OJO


%name = ['windows_data_filted_w', num2str(Windows)];

%disp('saving filted');
%name = ['save windows_data_filted_w', num2str(Windows),'.dat' ];
%exe = [name, '  windows_filted -ascii '];
%disp (exe);
```

```
%eval(exe);

%disp('saving filted svm');
%save_in_svm_format(windows_filted,['svm_windows_data_filted_w',num2st
      r(Windows),'.dat']);

%disp('saving training filted svm');
%save_in_svm_format(training_filted,['svm_windows_training_filted_w',n
      um2str(Windows),'.dat']);

disp('saving testing filted svm');
save_in_svm_format(testing_filted,['svm_windows_testing_filted_w',num2
      str(Windows),'.dat']);
% for testing data, the process is done.


%save(name,'windows_filted');


trndata_cons = [training_filted(:,2:end),training_filted(:,1)];


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% add
%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%


disp('SMOTE algorithme');
% Only consider the consistent data and without repetitions.
% We do oversampling using SOMTE, and undersampling using random, then
% we use ENN to clean up the whole dataset.

randn('state', 420);
rand('state', 420);
%addpath ..\..\..\generate_data\;

%load Def_WconstrnD_full.dat; % windowed imbalanced datasets. At the
      moment its size=7 primero generar los ficheros, esto no genera
      windows data
%trndata_cons = Def_WconstrnD_full;

%load windows_data_unfilted_w3.dat
%trndata_cons = windows_data_unfilted_w3;

%trndata_maj=trndata_cons(find(trndata_cons(:,end)==0),:); %@H
trndata_maj=trndata_cons(find(trndata_cons(:,end)==-1),:);
trndata_min=trndata_cons(find(trndata_cons(:,end)==1),:);

num_min = length(trndata_min)
num_maj = length(trndata_maj)

pause(3)

hector = trndata_cons;


KNN=5; % for Opt: KNN=11;
disp (['nº neighbours = ', num2str(KNN)]);
ratio = 2; % ratio is 1
```

```
%-----------------------------------
%width =3; % You need change this parameter so that it is the same as
      your windowed inputs
%-----------------------------------

% Oversampling using SMOTE-----------------------------------

%%%%%%%%%%-> ADD IT
%width = 1 %?????? I don't underestand this parameter.
width = Windows;

   [omindata, omintarget] = oversampling_RB_window(hector, KNN, ratio-
      1, width);%lib
   trndata_min =[omindata, omintarget]; % Tarjet is at the end.

   trndata = [trndata_min; trndata_maj];

   samplings = reptdata(trndata); % allowing repetitions in minority
      class
   index_min = find(samplings(:,end) == 1);
   numMin = length(index_min);
   trn_min = samplings(index_min,:);
   %index_maj = find(samplings(:,end) == 0);
   index_maj = find(samplings(:,end) == -1);
   trn_maj = samplings(index_maj,:);
   numMaj = length(index_maj);
   fprintf(1,'the difference is %d \n', (numMaj-numMin));
%-------------------------------------------------------------

disp('undersampling');

N = numMin*0.8; %% important ratio @y
% Undersampling using random-----------------------------------
%% undersampling majority:

newindexmaj = randperm(numMaj);
trn_maj = trn_maj(newindexmaj,:);

%umaj = trn_maj(1:floor(numMin*N),:);
N;
size(trn_maj);

if N > size(trn_maj,1)
    error('is not able to make undersampling minority class > majority
      class')
end
umaj = trn_maj(1:floor(N),:);

samplings=[umaj; trn_min];
%size(samplings)
%num = length(samplings);
num = size(samplings,1);
Perm = randperm(num);
samplings = samplings(Perm,:);

%samplings = tomek_for_clean(samplings);
%save rst_def_full_Wratio1 samplings

%load Def_WtstD_full.dat
%tstdata = Def_WtstD_full;
%load Def_WXtD_full.dat
```

```
%tstdata_cons = Def_WXtD_full;

%save windowed_size_samplings samplings; % keep the file name so that
      it's
%easy to understand @y

samplings = [samplings(:,end),samplings(:,1:end-1)];

name = ['windowed_samplings_w', num2str(Windows),'.dat'];
%cad = ['save ' name ' samplings -ascii'];

disp('saving sampling');
save_in_svm_format(samplings,[...
    'svm_training_samplings_w',num2str(Windows),'.dat']);
%cad
%eval(cad);

save(['W_',num2str(Windows)]);
```

```
%
% script: mappcross6
% these script give a combination of parameters to svm to make the
      experiments using cross-validation with 10 subsets and stored
      information about confusion matrix and metrics of the
      experiment, the parameters that has been used: parameter cost,
      gamma and with file has been use to test (testing un-filter,
      testing filter or training file, and the windows size), and also
      information about how much time has been used to develop the
      task.
% It use RBF kernel for all the experiments
% param_c = [2^-3,2^-1,2^1,2^3,2^5,2^7,2^9,2^11]
% param_g = [2^-11,2^-9,2^-7,2^-5,2^-3,2^-1,2^1,2^3]
% param_w = [1:2:3]
% crossvalidation_layers = 10



param_c = [2^-3,2^-1,2^1,2^3,2^5,2^7,2^9,2^11]
param_g = [2^-11,2^-9,2^-7,2^-5,2^-3,2^-1,2^1,2^3]
param_w = [1:2:3]
crossvalidation_layers = 10

readme = '[confusion(:,1);confusion(:,2);recall; precision; f_measure;
      accuracy; fp_rate; nCC; -c parameter; -g parameter; name of the
      data file]';

if ~exist('ultimo') || ultimo == 0

    ultimo = 0;
    results_unfilted = [];
    results_filted = [];
    results_samplings = [];
    time_need_it = [];
else
```

```
    disp('PROCES LOADED');
    ultimo
    pause(2);
end

time0 = clock;
moverse = ultimo;
pos = 1;

for  w_ = param_w
for  c_ = param_c
for  g_ = param_g
       %% For permiting continue.

        if moverse > 0
            moverse = moverse -1;
            continue
        end
        time1 = clock;
        ultimo = ultimo +1;


        paramiters = ['-s 0 -b 1 -c ' , num2str(c_) , ' -t 2 -g ' ,
      num2str(g_) , ' -e 0.00001  '];


      [results_k_layer,results_in_testing_filted,results_in_testing_un
      filted] = ...
            make_crossvalidation3(['svm_training_samplings_w' ,
      num2str(w_),'.dat'],...
            ['svm_windows_testing_unfilted_w' ,
      num2str(w_),'.dat'],...
            ['svm_windows_testing_filted_w' , num2str(w_),'.dat'],...
            crossvalidation_layers,paramiters);

        %results_samplings = [ results_samplings ,
      [confusion(:,1);confusion(:,2);recall; precision; f_measure;
      accuracy; fp_rate; nCC; c_; g_; w_]];
        results_filted = [ results_filted ,results_in_testing_filted]
      ;
        results_unfilted = [ results_unfilted
      ,results_in_testing_unfilted] ;



        seconds = etime(clock,time1);
        time_need_it = [time_need_it , seconds];

        disp (['step: ', num2str(ultimo) ,' w= ' ,num2str(w_), ' has
      been needit ' num2str(seconds/60) ' minutes. Runing time = '
      num2str(etime(clock,time0)/3600) ' hours ']);
        disp('.');

        save('mappcross6.mat');
        save('copymappcross6.mat');

        %system('del y.output');
        %system('del x.model');
```

```
end
end
readme
end
```

```
function [recall, precision, f_measure, accuracy, fp_rate, nCC] =
      metrics(C);
% This function works for 2 class confusion matrix
% This function returns the metrics parameters for one confusion
      matrix.
% Input:
% Confusion matrix [TN, FP; FN TN];
% Output:
% recall
% precision
% f_measure
% accuracy
% fp_rate
% nCC
%


TN = C(1,1);
TP = C(2,2);
FP = C(1,2);
FN = C(2,1);

ndata = TN+TP+FP+FN;

recall = TP/(TP+FN);
precision = TP/(TP+FP);
f_measure = 2/(1/precision+1/recall);

accuracy = (TP+TN)/ndata;

fp_rate = FP/(TN+FP);
%specificity = 1-fp_rate;
de = (TP+FN)*(TN+FP)*(TP+FP)*(TN+FN);
nCC = (TP*TN-FN*FP)/(sqrt(de));
```

```
function [newdata, target] = oversampling_RB_window(data, K, N,
      width);

% This is a modified version of SMOTE: the distances are computed
% using HVDM and the discrete values are randomly choosen from
      neighbours
% _RB_ means this is a function for mixed data sets
% This function is based on SMOTE algorithm

%
% data            an imbalanced ndata x ndim matrix;
% K               Number of nearest neighbours;
% newdata         new dataset
% target          labels of the new dataset
% N               amount of SMOTE, if N>K, then some samples are
% width           the size of a window
% interpoting more than once.

[ndata, ndim] = size(data);

labels_trn = data(:,ndim);
X = data(:,1:ndim-1);

index_min = find(labels_trn == 1);
numMin = length(index_min);
trn_min = X(index_min,:);
ltrn_min = labels_trn(index_min);

index_maj = find(labels_trn == 0);
numMaj = length(index_maj);
trn_maj = X(index_maj,:);
ltrn_maj = labels_trn(index_maj);

% Initialisation
omin=zeros((N+1)*numMin, ndim-1);

%% oversampling minority: SMOTE
Var_real = var(data(:,6:12));
% computing VDM for nominal
D_vdm = vdm(data(:,1:5), data(:,end));
for i = 1:numMin
  centre = trn_min(i,:);
  remin = trn_min;
  remin(i,:) =[];
  % computing the Heterogeneous value difference metric distance:
  hd = hvdm(remin, centre, Var_real, D_vdm);

  [dist, ind] = sort(hd);
  NN = remin(ind(1:K),:);

  if N>K
    more_num = N-K;
    more = fix(rand(1, more_num)*K+1);
    NN=[NN; NN(more,:)];
    total = N;
  else
    total = K;
  end

  Perm = randperm(total);
  ind_perm =1;
```

```
  all_synthe=[];
  newN =N;
  while newN~=0
    r = Perm(ind_perm);
    rs = NN(r,:);
   for Bin = 1:width
    diff = centre(:, Bin*12-6:Bin*12) - rs(:, Bin*12-6:Bin*12);
    synthe_r = centre(:, Bin*12-6:Bin*12) +
      rand(1,length(diff)).*diff;
    comb=[centre(:,1:5); rs(:,1:5)];
    comp = [rs(:,1:5)==centre(:,1:5)];
    for col = 1:5
        if comp(col)==1
          synthe_b(col) = rs(:,col);
        else
          rb = randperm(2);
          first = rb(1);
          synthe_b(col) = comb(first,col);
        end
    end
    synthe = zeros(1, 12*width);
    synthe(1, Bin*12-6:Bin*12) = synthe_r;
    synthe(1, (Bin-1)*12+1:(Bin-1)*12+5) = synthe_b;
   end % For Bin = 1:width
    all_synthe = [all_synthe; synthe];
    newN = newN-1;
    ind_perm = ind_perm+1;

  end
  addmin = zeros(N+1, ndim-1);
  addmin(1,:) = centre;
  addmin(2:N+1,:) = all_synthe;
  omin(N*(i-1)+i:i*(N+1),:) = addmin;
end
newdata = omin;
target = ones(size(omin,1),1);
```

```
% script runwindows.
% is for generate windows data
for i = 13:2:41
    disp ('starting one');
    make_windows_data_new (i);
    disp ('one is done');
    disp (num2str(i));
end
```

```
function save_in_svm_format(Matrix,File)
%
% function save_in_svm_format(Matrix,File)
% These functions save a Matrix in SVM format,
% Input:
% Matrix: label mast be in the first column.
% File: <string> the name of the file.
% Output: do not have

rows =  size(Matrix,1);
columns = size(Matrix,2);
```

```
fid3 = fopen(File,'w');
if (fid3 < 1 )
    error(['error in file  ',File]);
end


for (r = 1:rows)
    %line = Matrix(r,:);
    fwrite(fid3,num2str(Matrix(r,1)));
    fwrite(fid3,'  ');

    for(i = 2:columns)
        fwrite(fid3,num2str(i-1));
        fwrite(fid3,':');
        fwrite(fid3,num2str(Matrix(r,i)));
        fwrite(fid3,'   ');
    end

    if r ~= rows
        fwrite(fid3,[10,13]);
    end

end
fclose(fid3);
```

```
function [NewMatrix] = sortmatrix(Matrix);
% function [NewMatrix] = sortmatrix(Matrix);
% These function sort one matrix in alphabetical pre-order.
% Input:
% Matrix
% Output:
% NewMatrix.
%Label mast be at the end at the last column


columns = size(Matrix,2);
rows = size(Matrix,1);


if rows == 0 || columns == 0
    NewMatrix = [];
else

[A,B] = sort(Matrix,1);
positions = B(:,1);

AuxMatrix = Matrix(positions,:);




if rows > 1
    pivote = AuxMatrix(1,1);
    cont = 1;

    for pos = 1:rows
        if AuxMatrix(pos,1)==pivote
```

```
            continue
        else
            AuxMatrix(cont:pos-1,2:end) =
      sortmatrix(AuxMatrix(cont:pos-1,2:end));
            cont = pos;
            pivote = AuxMatrix(cont,1);
        end
    end
    AuxMatrix(cont:end,2:end) = sortmatrix(AuxMatrix(cont:end,2:end));

end
NewMatrix = AuxMatrix;
end
```

```
function d = vdm(data, labels)

% Value difference metric (VDM)
% From paper <<Reduction Techniques for Instance-Based learning
% Algorithms>>, authors: D. R. Wilson and T. R. Martinez
% Machine Learning, 38-3, pp. 257-286 (2000)

% attr1:  the column with attribute 1
% x:      indicating value 1
% y:      indicating value -1
% labels: patterns' lables

% Compute the number of times attribute a had value x and y;
x=[data(:,1)==1 data(:,2)==1 data(:,3)==1 data(:,4)==1 data(:,5)==1];
N_a_x = sum(x);

y=[data(:,1)==-1 data(:,2)==-1 data(:,3)==-1 data(:,4)==-1
      data(:,5)==-1];
N_a_y = sum(y);

% Compute the number of times attribute a had value x and y in c
      class;
for i = 1:2
    ind_x = find(data(:,i)==1);
    N_a_x_1 = length(find(labels(ind_x)==1));
    N_a_x_2 = length(find(labels(ind_x)==0));

    ind_y = find(data(:,i)==-1);
    N_a_y_2 = length(find(labels(ind_y)==0));
    N_a_y_1 = length(find(labels(ind_y)==1));

    d(i)=(N_a_x_1/N_a_x(i) - N_a_y_1./N_a_y(i))^2+ (N_a_x_2/N_a_x(i) -
      N_a_y_2/N_a_y(i))^2;
end
```

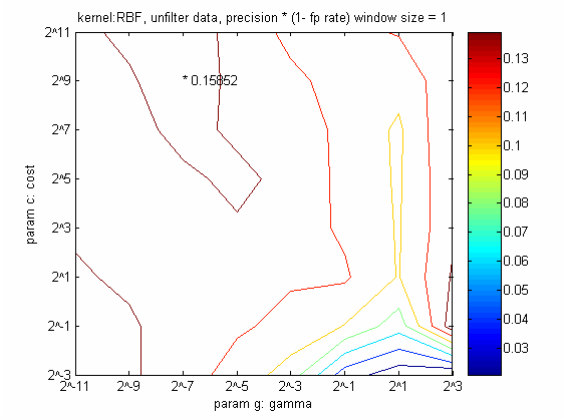
```
function [LabelW, MatrixW] = Window_ing
      (TamWindows,Labels,MatrixSingle);
% function [LabelW, MatrixW] = Window_ing
      (TamWindows,Labels,MatrixSingle);
%
% These function produce a un-filter windows data.
% Input:
%   TamWindows: Windows size.
%   Labels:  column with the labels.
%   MatrixSingle: Matrix with the inputs vectors.
% Output
%   LabelW = Labels for the new windowed data.
%   MatrixW = New inputs vectors.

side = (TamWindows-1)/2

LabelW = Labels(1+side:end-side);

long = size(LabelW,1);

MatrixW = zeros(long,12*TamWindows);

for i = 1:long
    for j = 0:TamWindows-1;
        k = j*12;
        MatrixW(i,k+1:k+12) = MatrixSingle(i+j,:);
    end
end
```

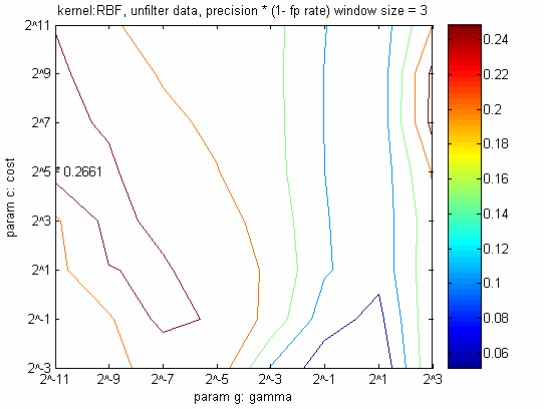
```
% Only consider the consistent data and without repetitions.
% We do oversampling using SOMTE, and undersampling using random, then
% we use ENN to clean up the whole dataset.

randn('state', 420);
rand('state', 420);
%addpath ..\..\..\generate_data\;

%load Def_WconstrnD_full.dat; % windowed imbalanced datasets. At the
      moment its size=7 primero generar los ficheros, esto no genera
      windows data
%trndata_cons = Def_WconstrnD_full;
load windows_data_unfilted_w3.dat
trndata_cons = windows_data_unfilted_w3;

%trndata_maj=trndata_cons(find(trndata_cons(:,end)==0),:); %@H
trndata_maj=trndata_cons(find(trndata_cons(:,end)==-1),:);
trndata_min=trndata_cons(find(trndata_cons(:,end)==1),:);
num_min = length(trndata_min)
num_maj = length(trndata_maj)
pause(3)

hector = trndata_cons;
%hector=trndata_cons(1:1000,:);

KNN=5; % for Opt: KNN=11;
ratio = 2; % ratio is 1
```

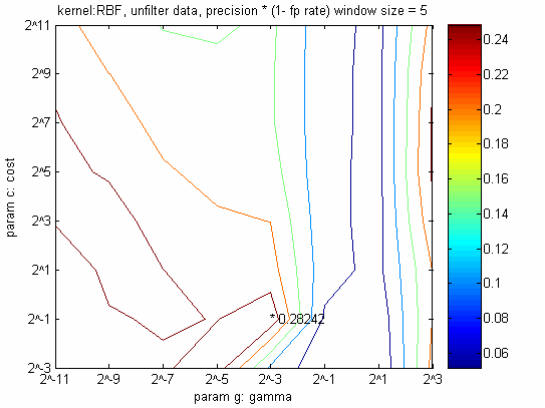
```
%-----------------------------------
%width =3; % You need change this parameter so that it is the same as
      your windowed inputs
%-----------------------------------

% Oversampling using SMOTE-----------------------------------

%   [omindata, omintarget] = oversampling_RB_window(hector, KNN,
      ratio-1, width);
%   trndata_min =[omindata, omintarget];

   trndata = [trndata_min; trndata_maj];

   samplings = reptdata(trndata); % allowing repetitions in minority
      class
   index_min = find(samplings(:,end) == 1);
   numMin = length(index_min);
   trn_min = samplings(index_min,:);
   %index_maj = find(samplings(:,end) == 0);
   index_maj = find(samplings(:,end) == -1);
   trn_maj = samplings(index_maj,:);
   numMaj = length(index_maj)
   fprintf(1,'the difference is %d \n', (numMaj-numMin));
%-----------------------------------------------------------------

N = numMin*0.8; %% important ratio @y
% Undersampling using random-----------------------------------
%% undersampling majority:

newindexmaj = randperm(numMaj)
trn_maj = trn_maj(newindexmaj,:)

%umaj = trn_maj(1:floor(numMin*N),:);
N
size(trn_maj)
umaj = trn_maj(1:floor(N),:);

samplings=[umaj; trn_min];
size(samplings)
num = length(samplings);
Perm = randperm(num);
samplings = samplings(Perm,:);

%samplings = tomek_for_clean(samplings);
%save rst_def_full_Wratio1 samplings

%load Def_WtstD_full.dat
%tstdata = Def_WtstD_full;
%load Def_WXtD_full.dat
%tstdata_cons = Def_WXtD_full;

save windowed_size_samplings samplings;
```

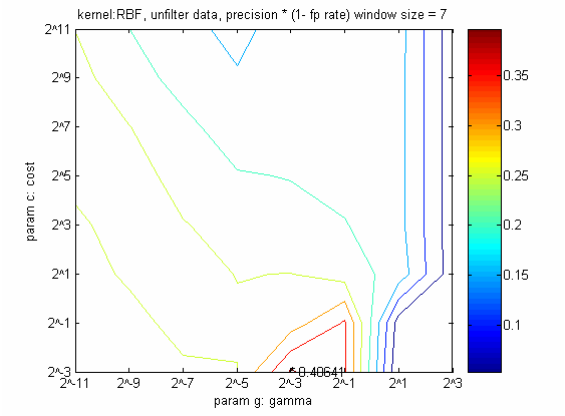
APPENDIX H:
FIGURES FOR RBF KERNEL:

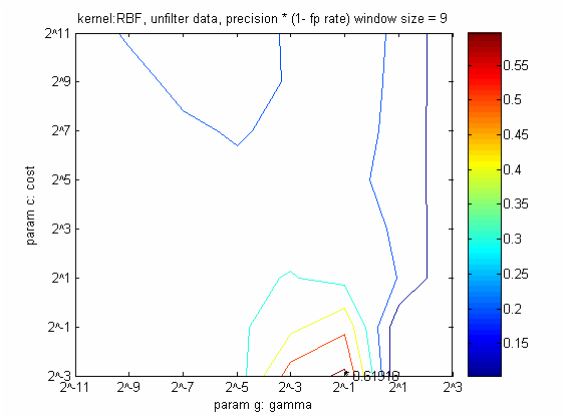
Un-filter data:
Ratio: precision*(1-fp_rate)

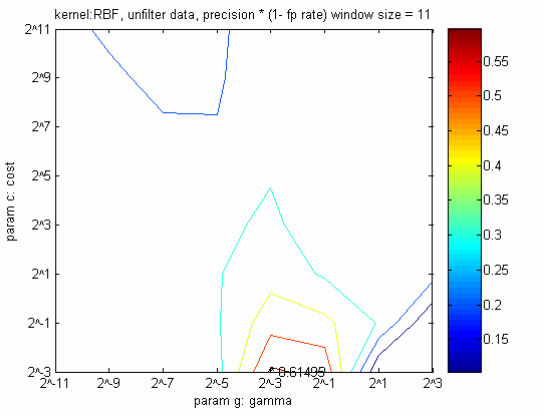


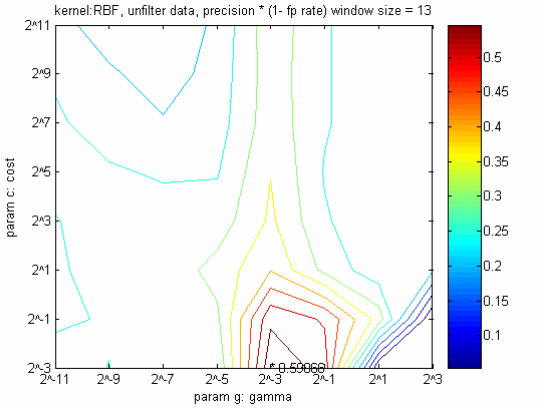




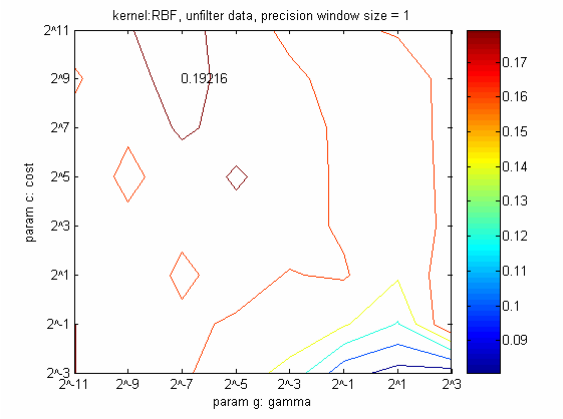


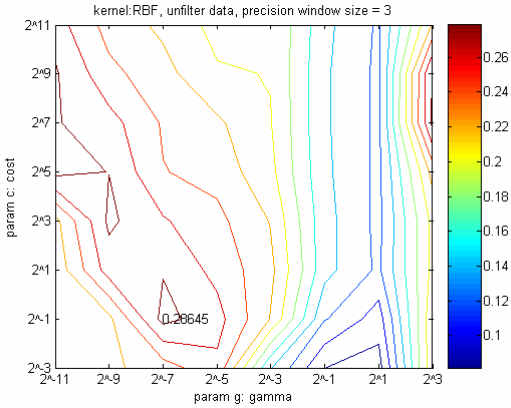


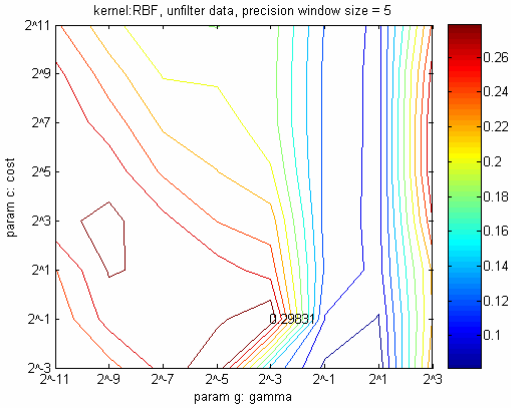


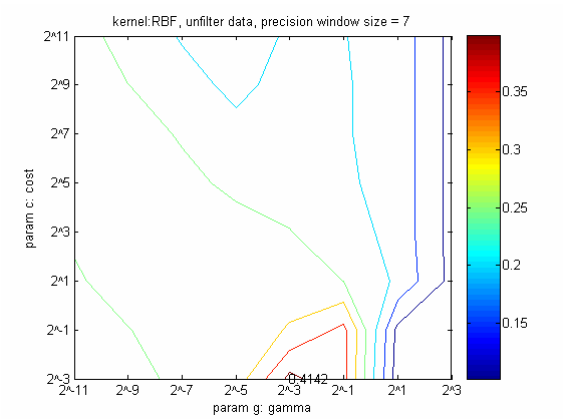


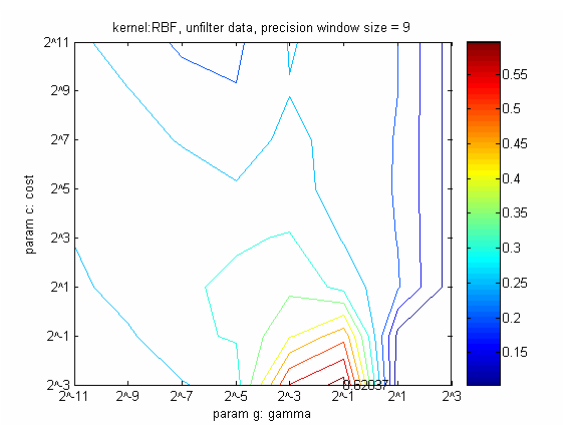
Un-filter data:
Ratio: precision

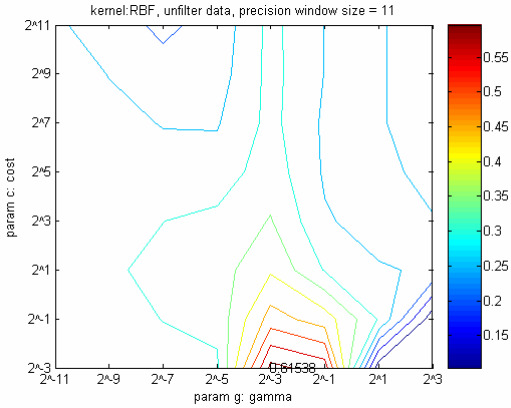


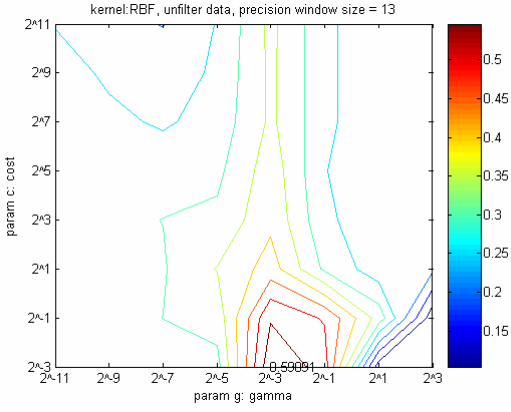




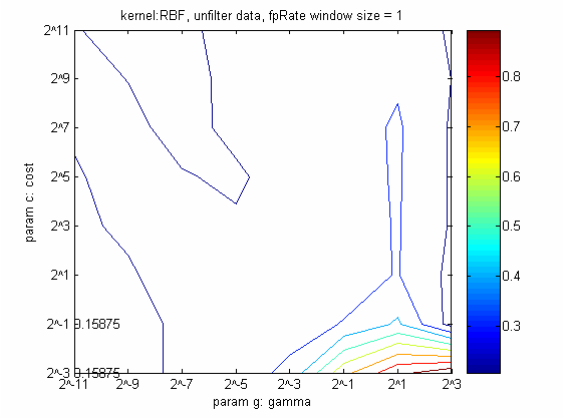


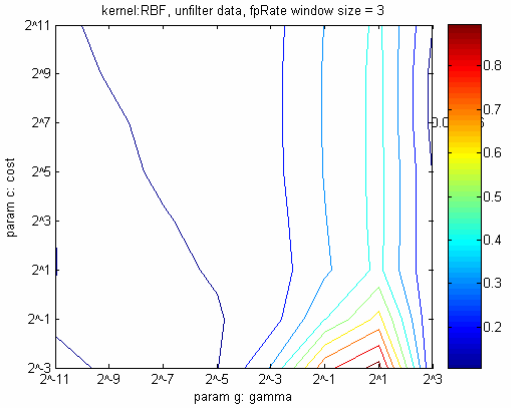


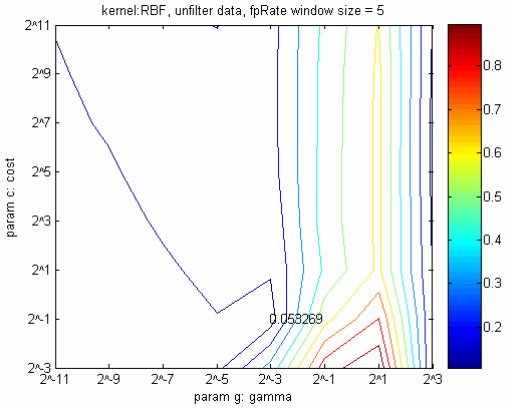


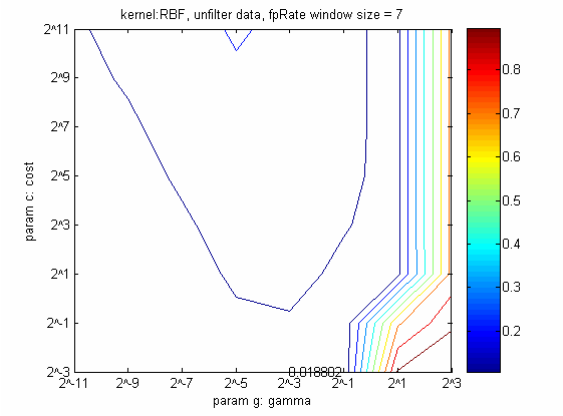


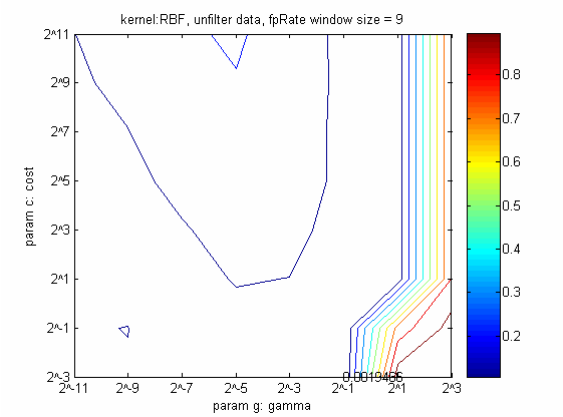
Un-filter data: False Positive Rate

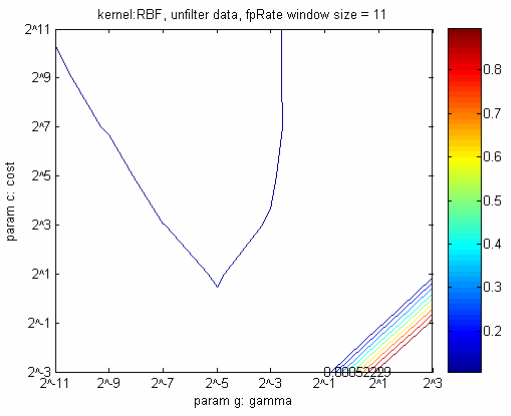


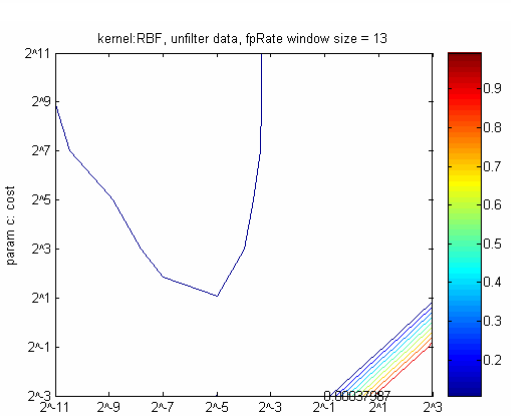




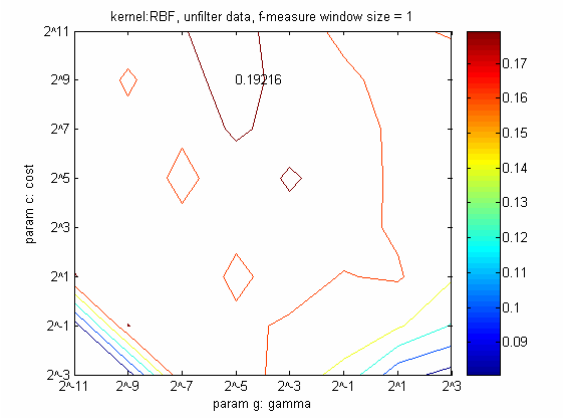


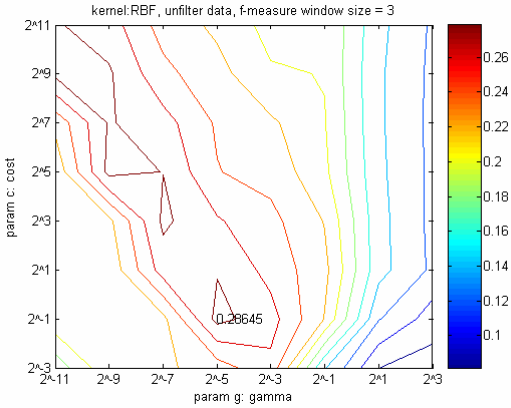


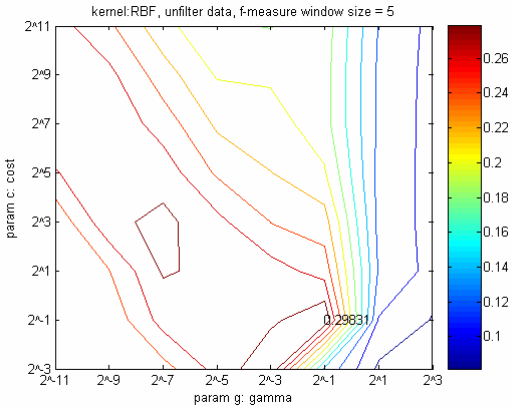


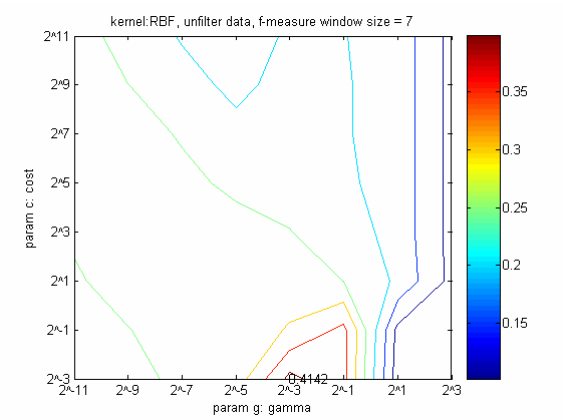


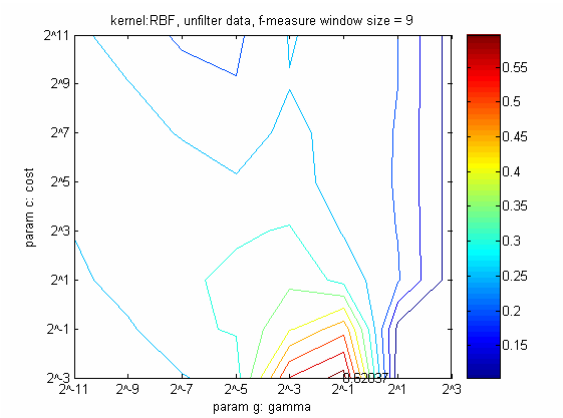
Un-filter data: F-measure

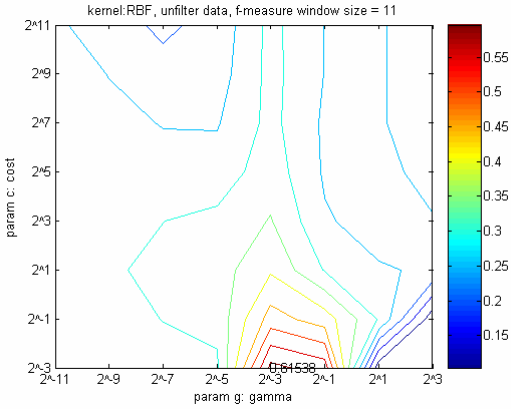


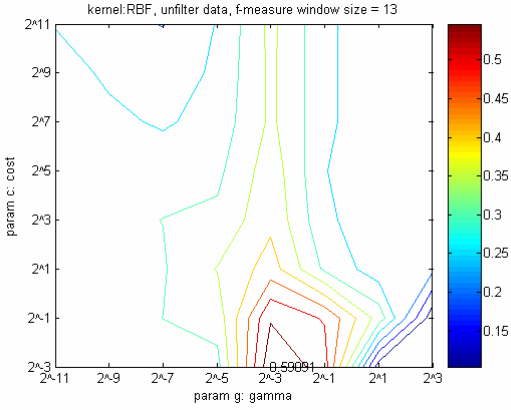




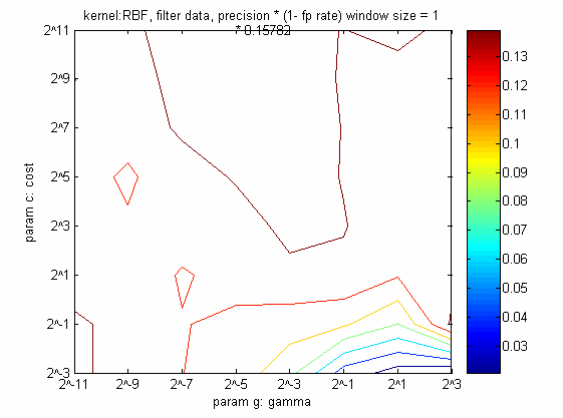


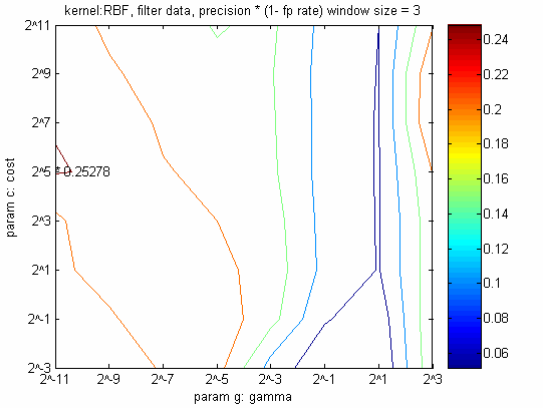


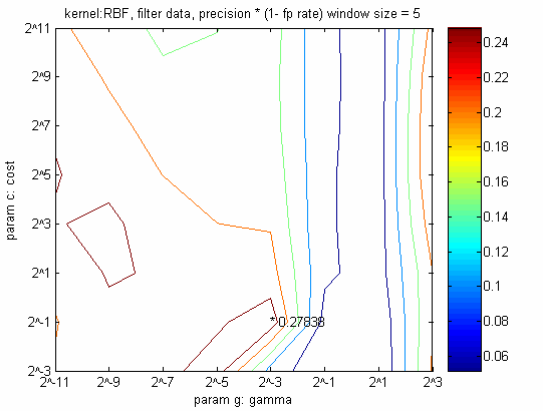


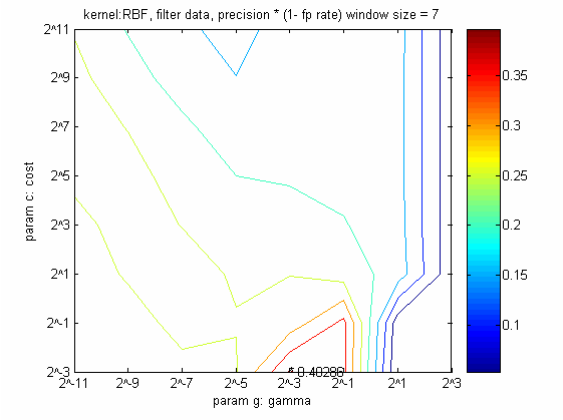


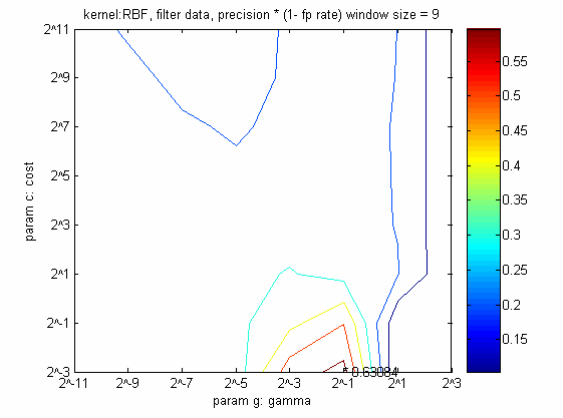
Filter data:
Ratio: precision*(1-fp_rate)

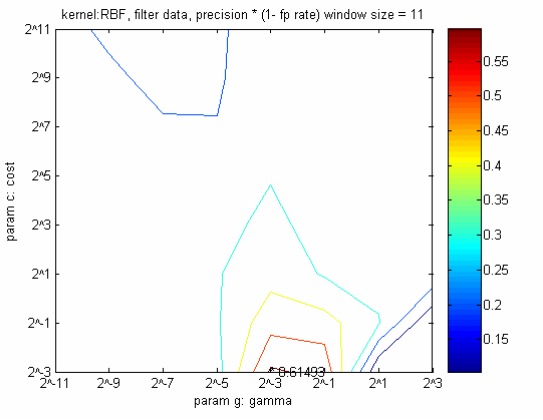


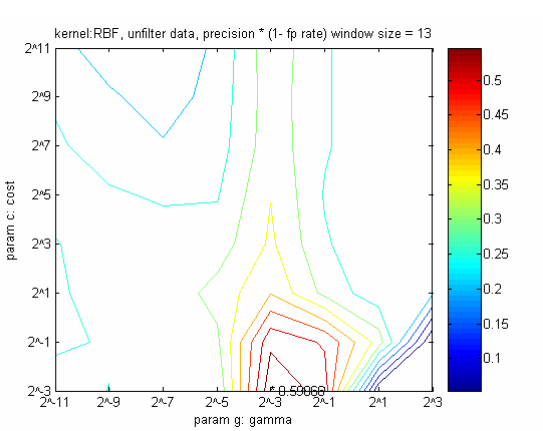




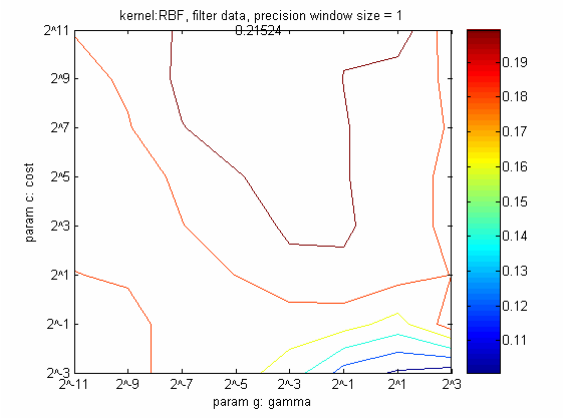


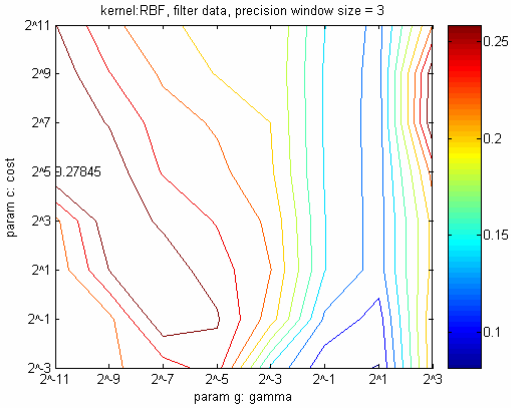


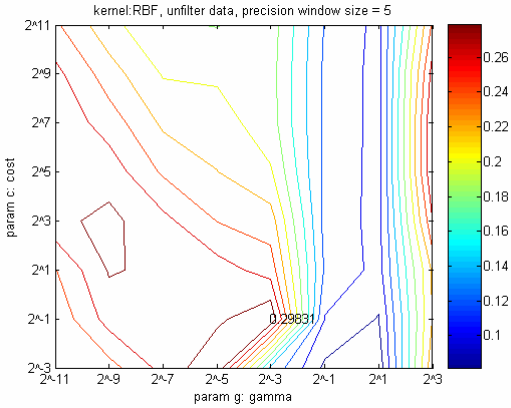


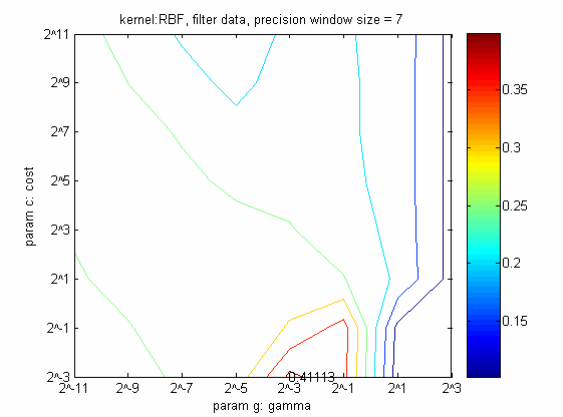


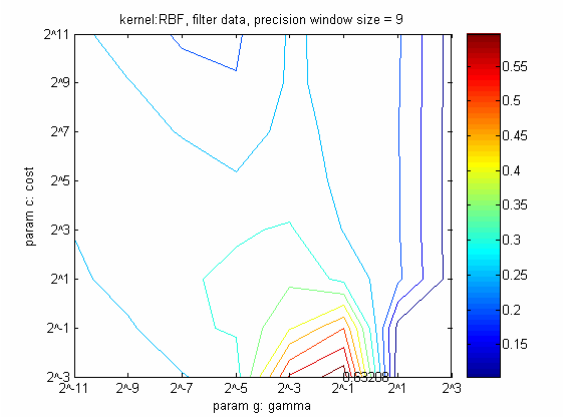
Filter data: Precision

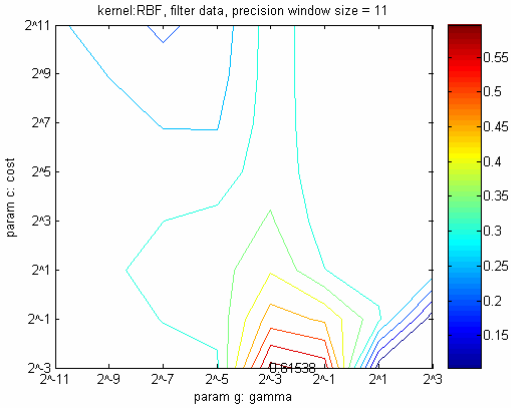


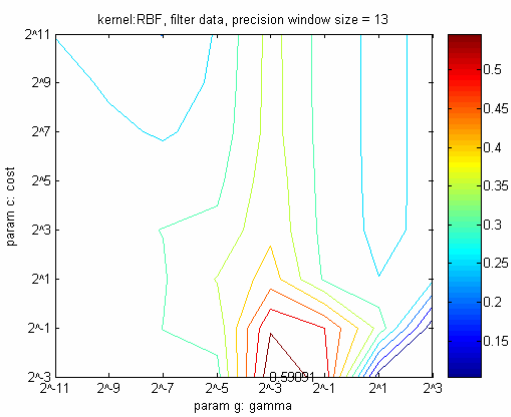




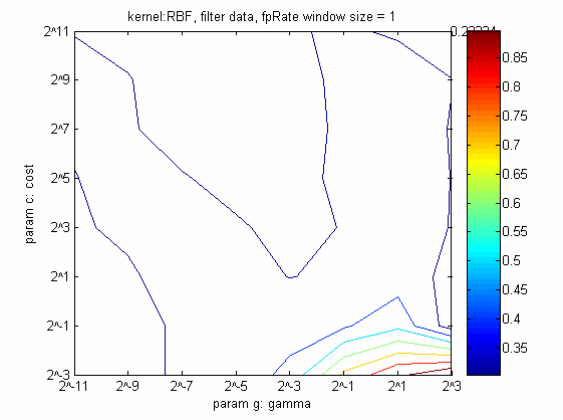


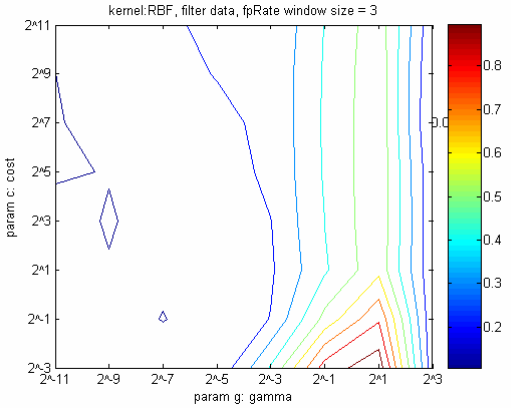


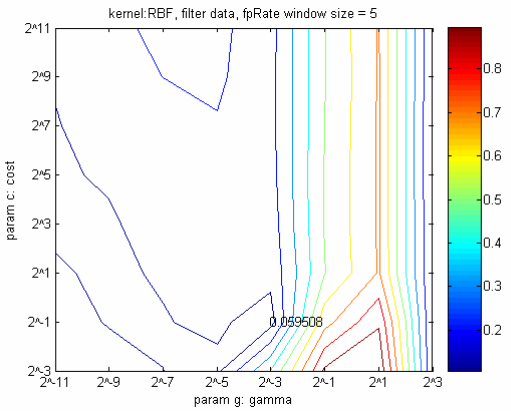


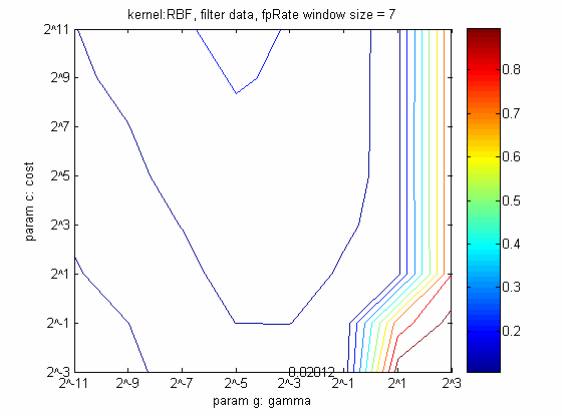


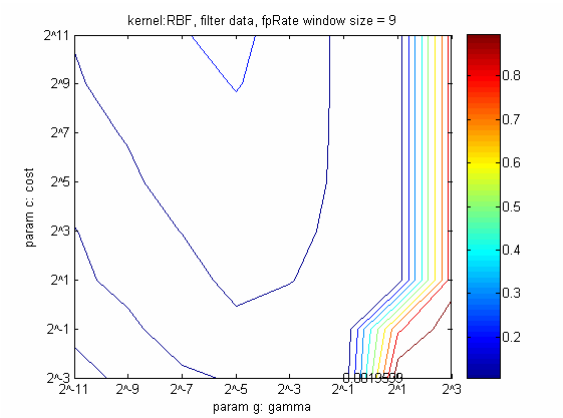
Filter data:
False Positive Rate:

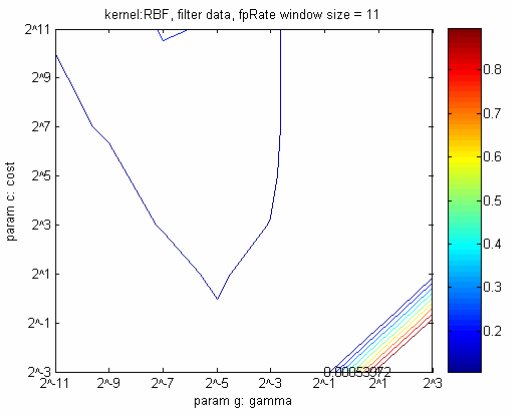


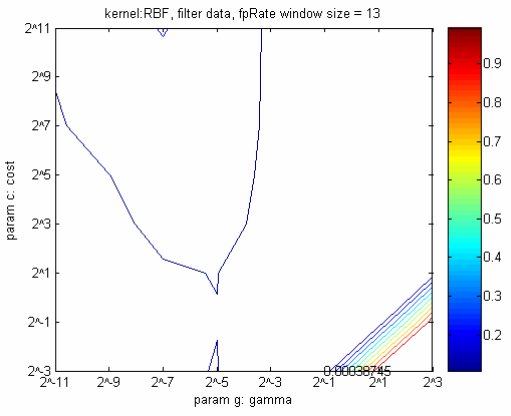




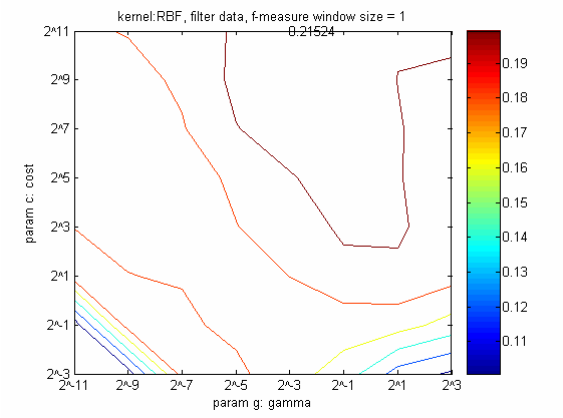


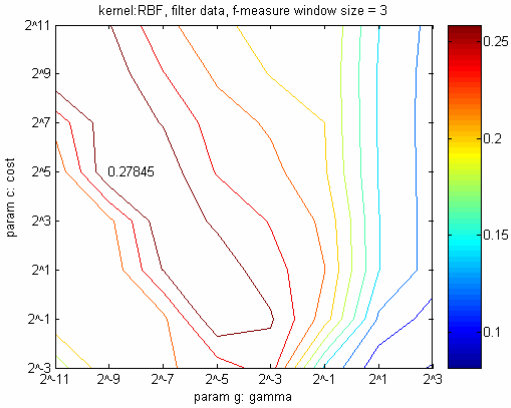


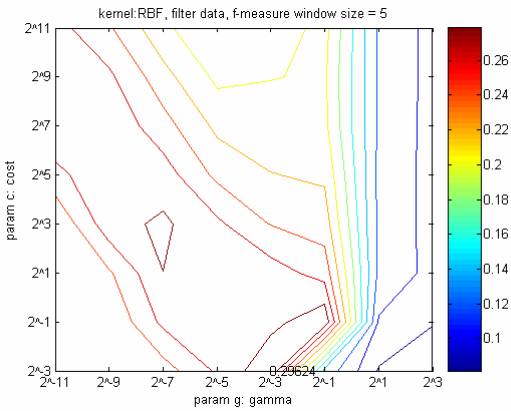


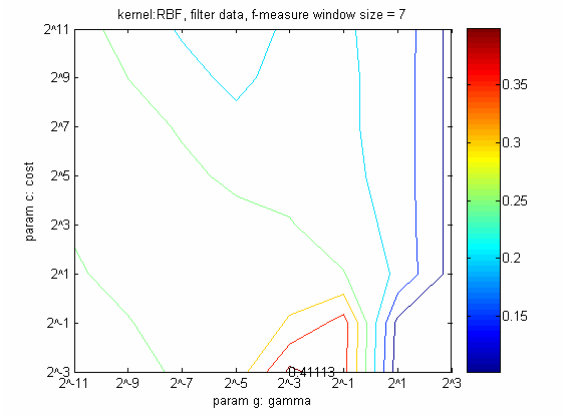


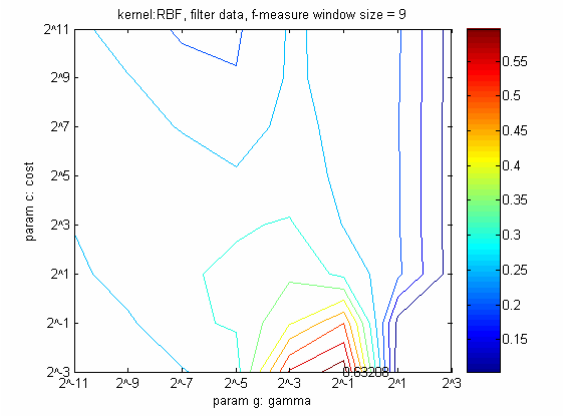
Filter data:
F-measure:

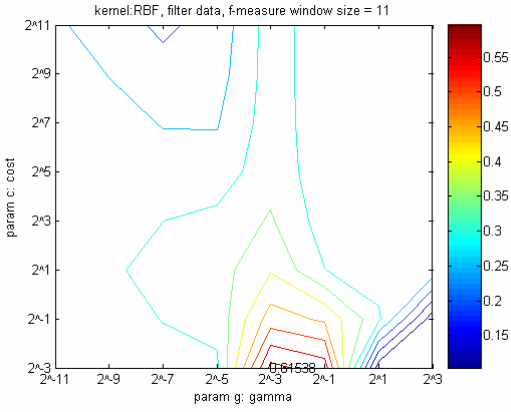


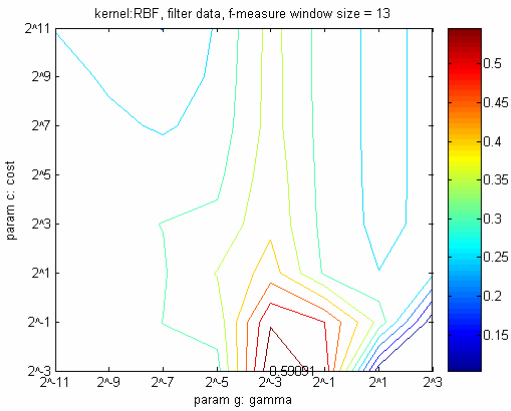




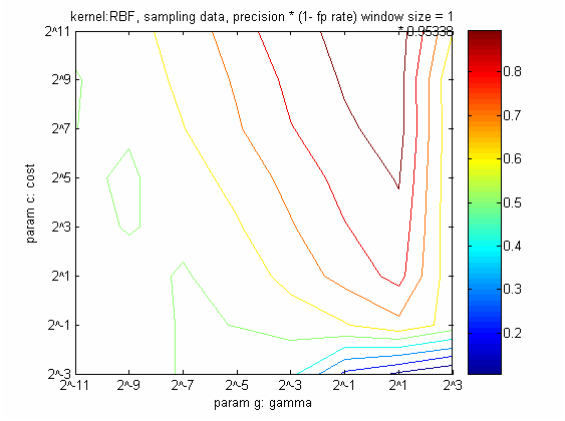


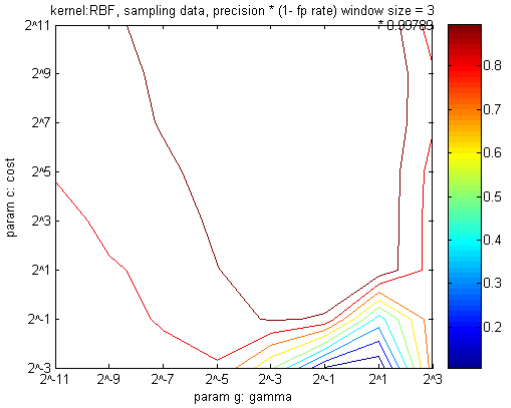


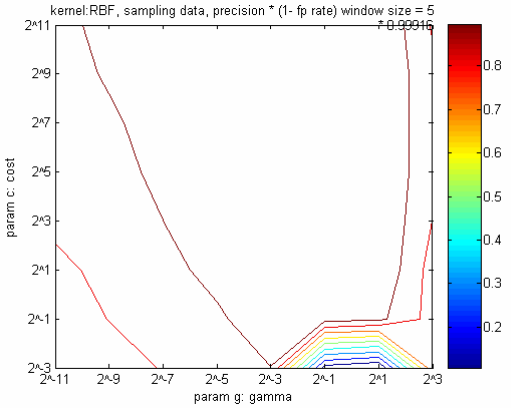


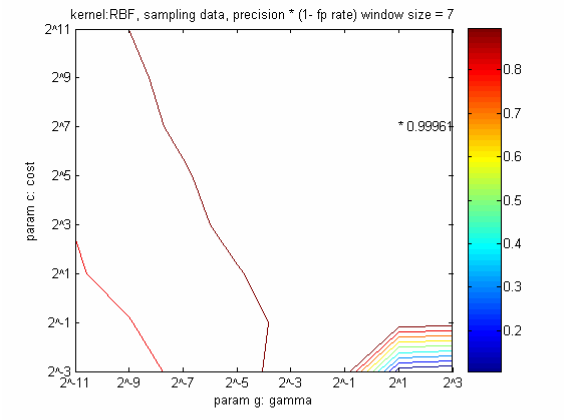


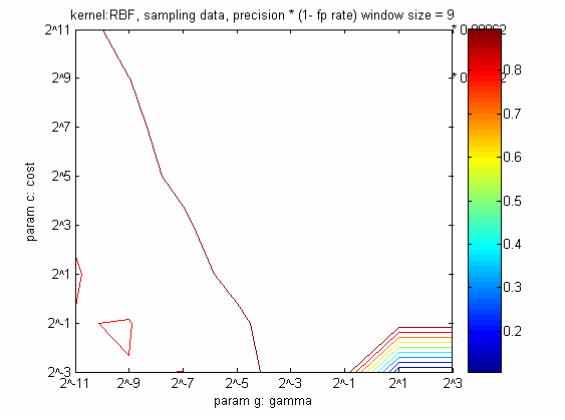
Training data:
Ratio: precision*(1-fp_rate)

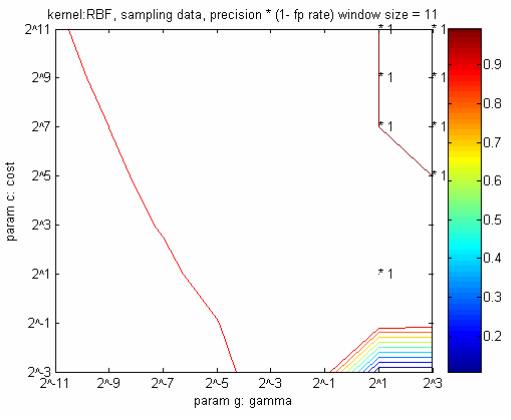


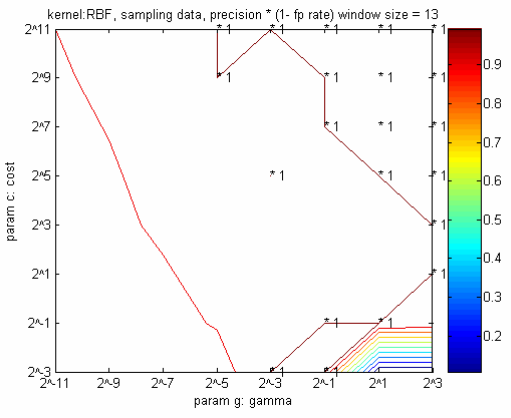




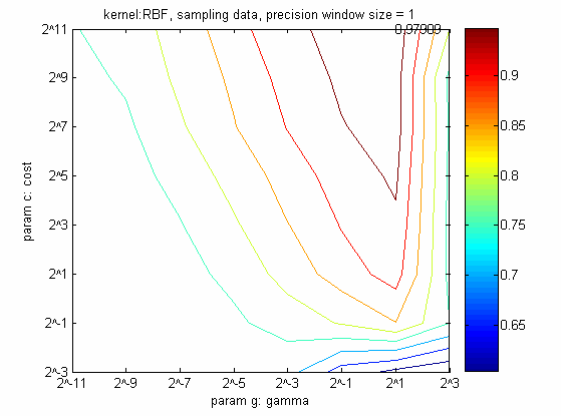


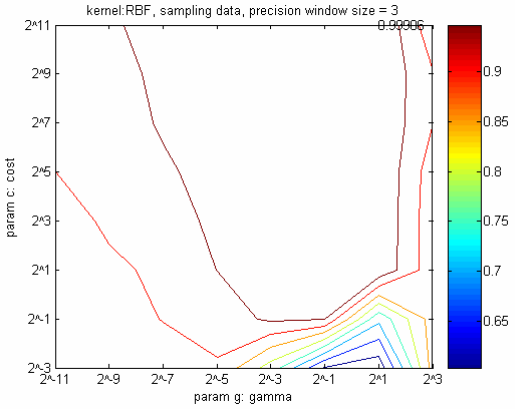


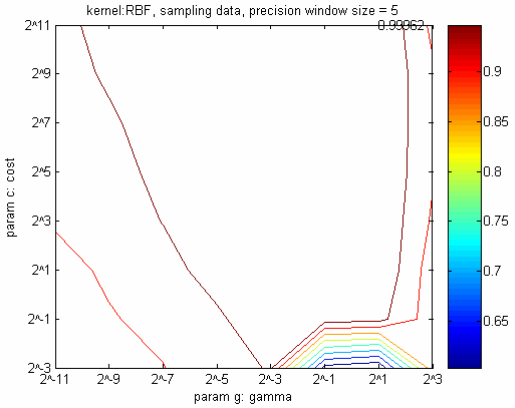


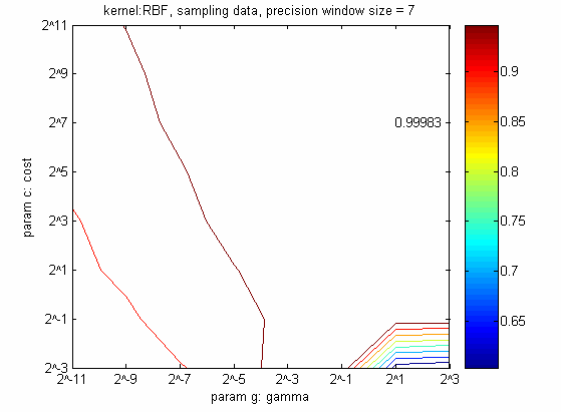


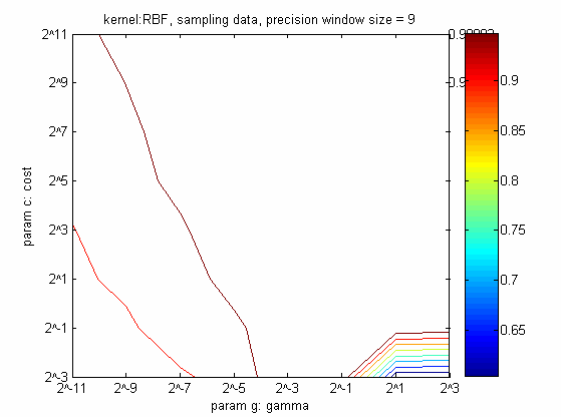
Training data: Precision

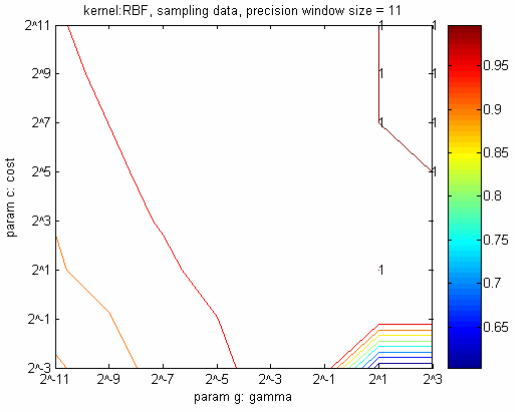


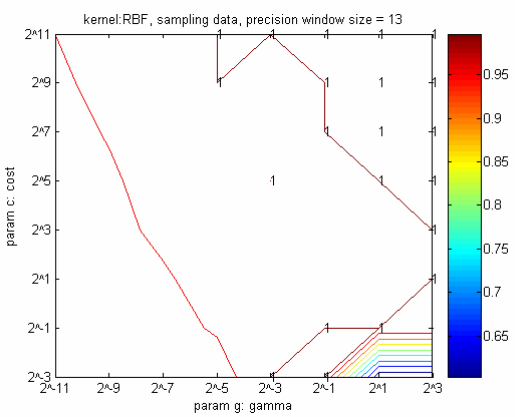




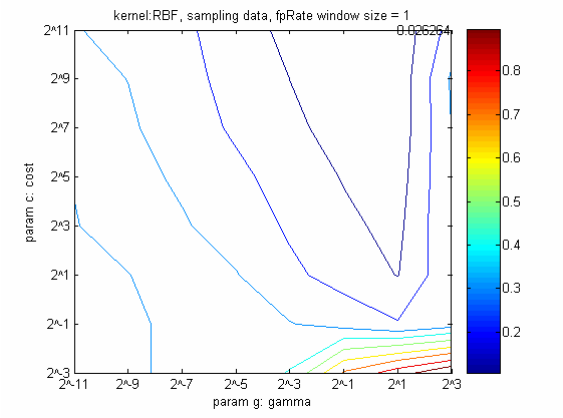


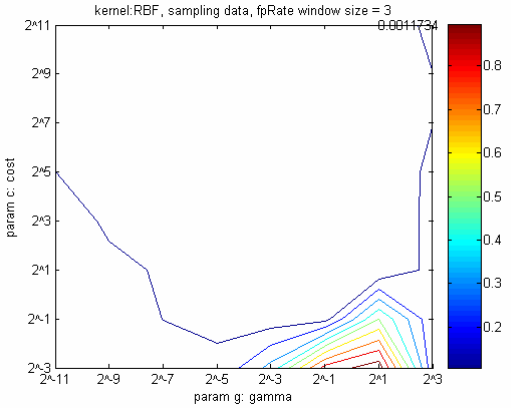


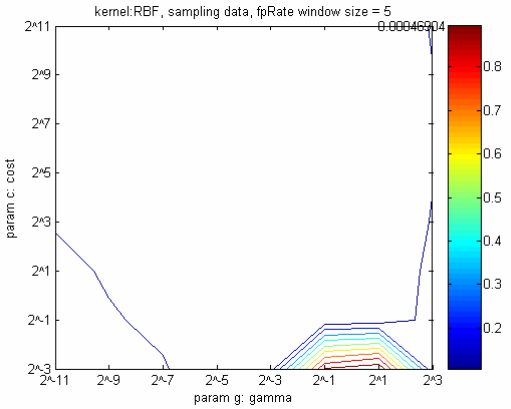


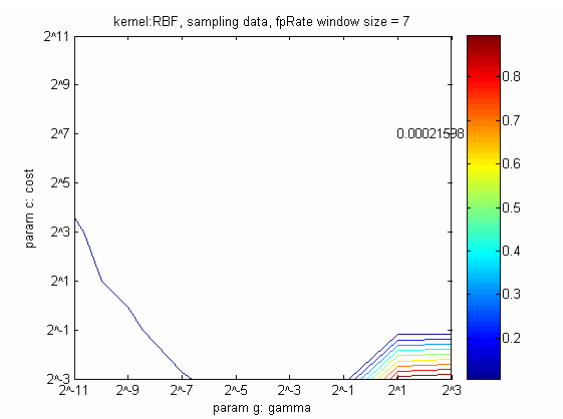


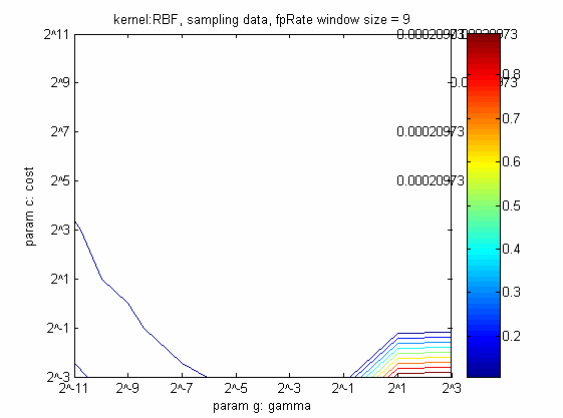
Training data:
False Positive Rate:

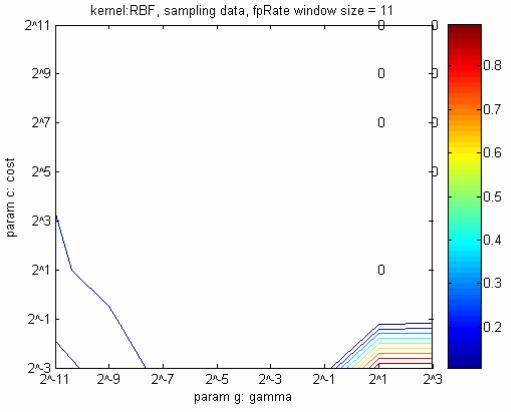


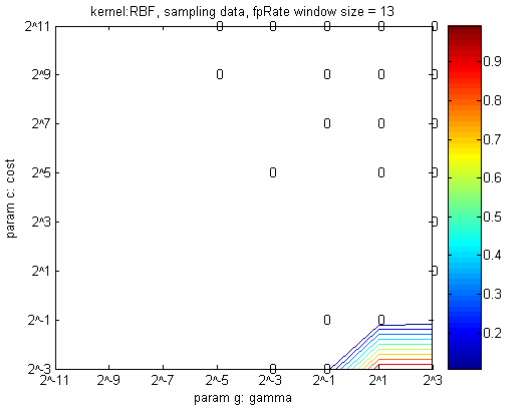




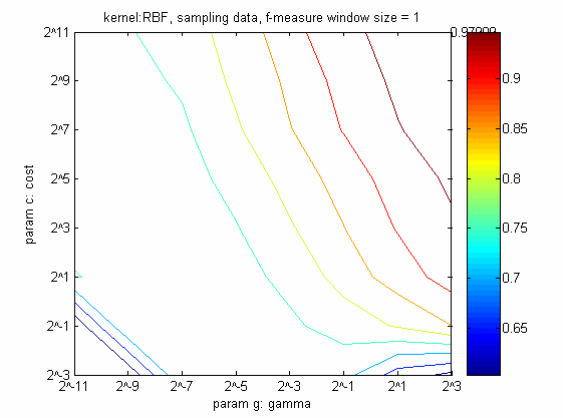


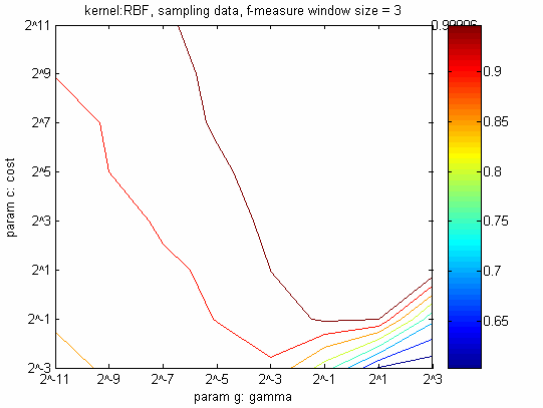


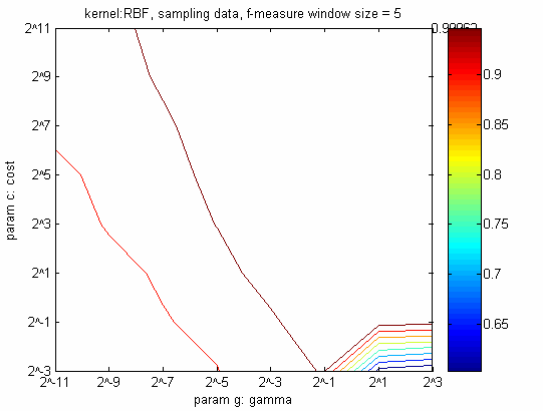


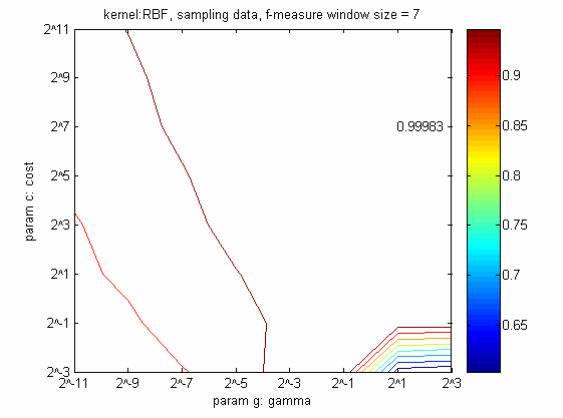


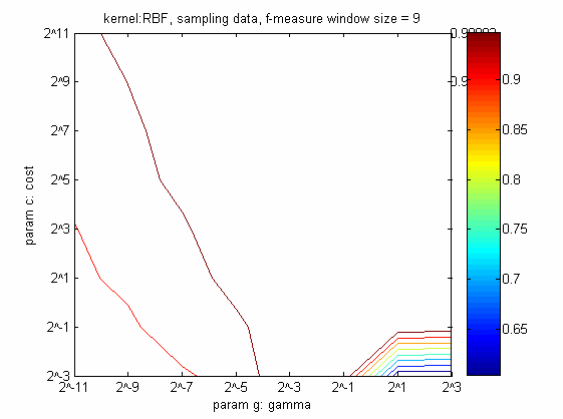
Training data: F-measure

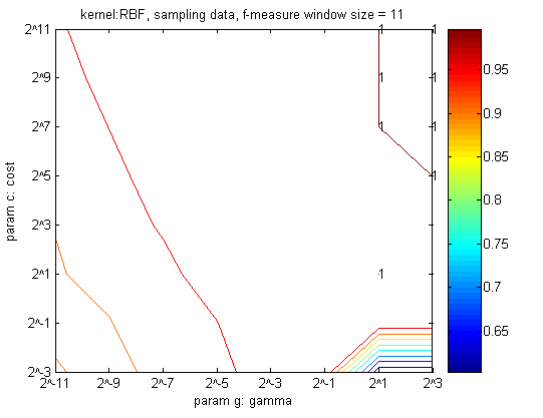


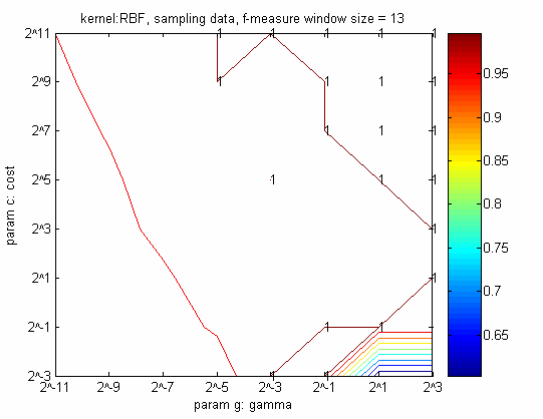




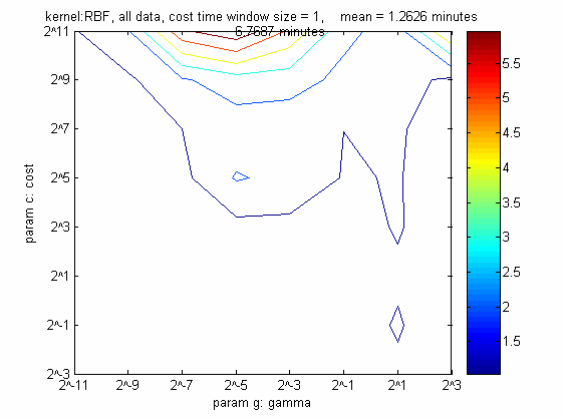


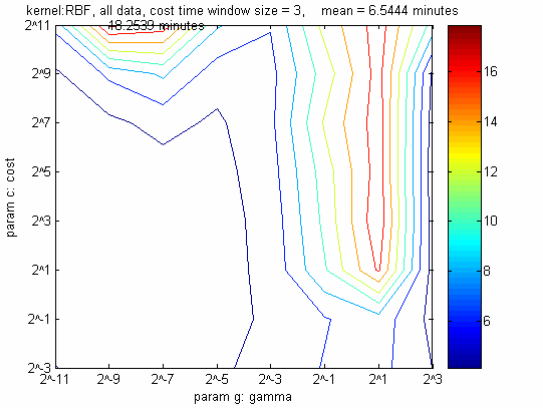


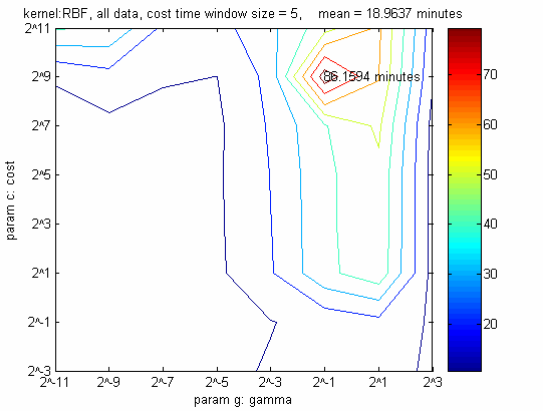


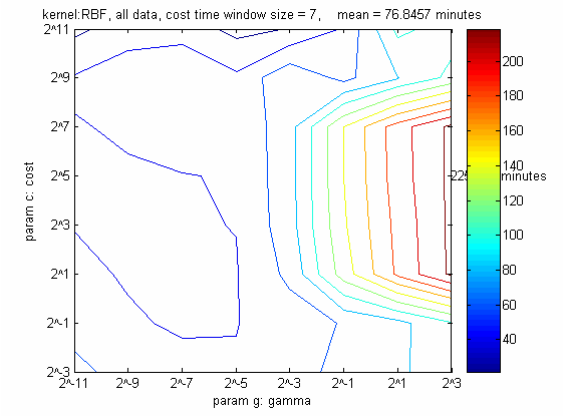


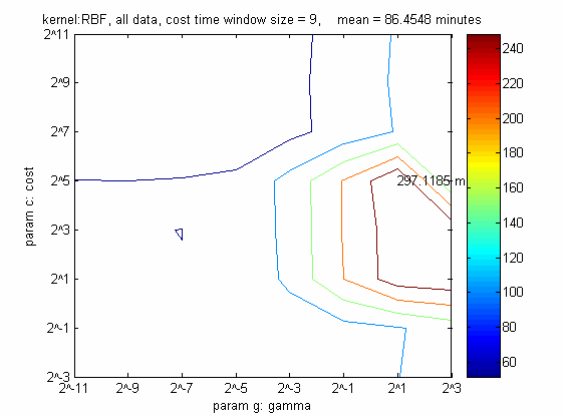
All data:
Computational time:
(The values can be affected by other process in the background of the CPU)

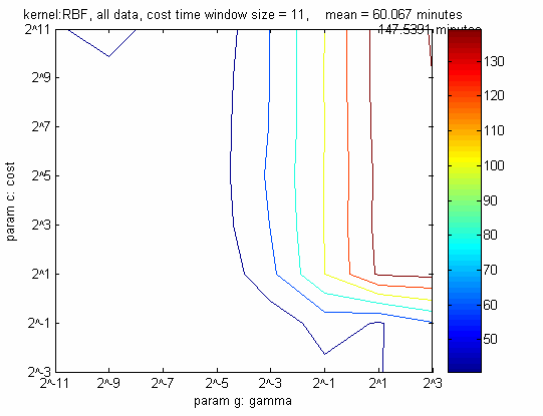


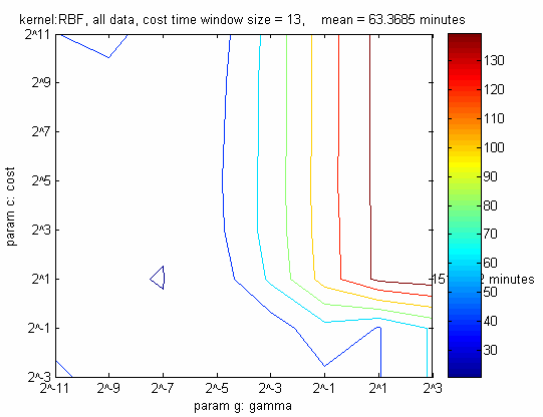




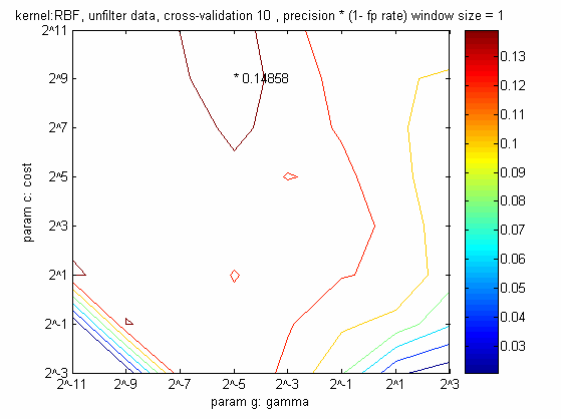


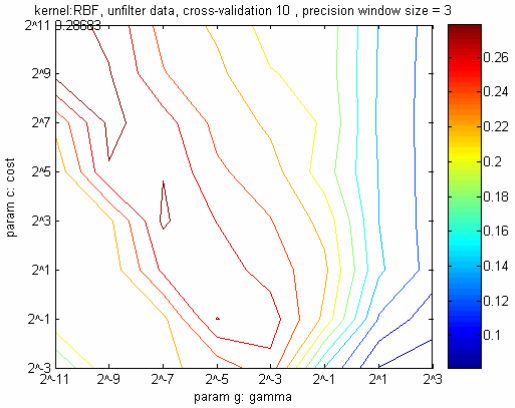


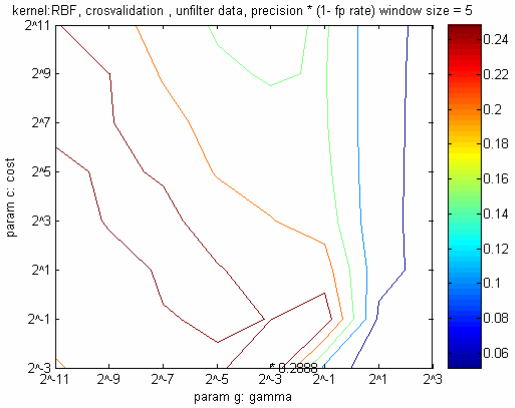




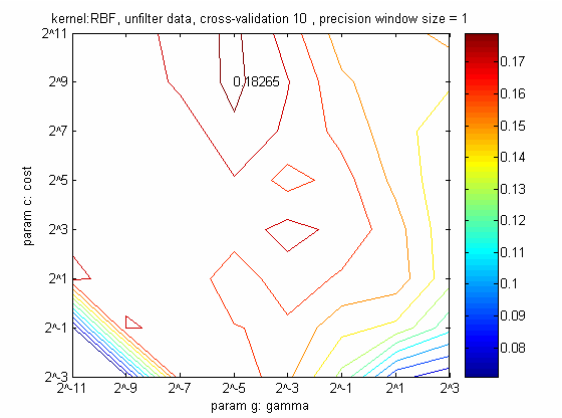
Un-filter data with cross-validation Ratio: precision*(1-fp_rate)

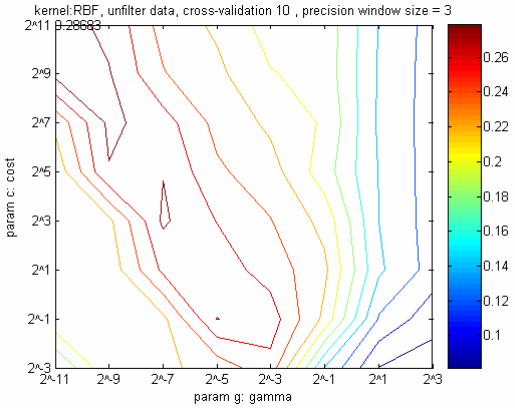


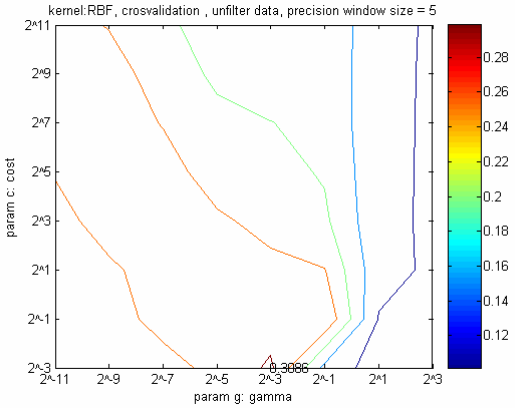




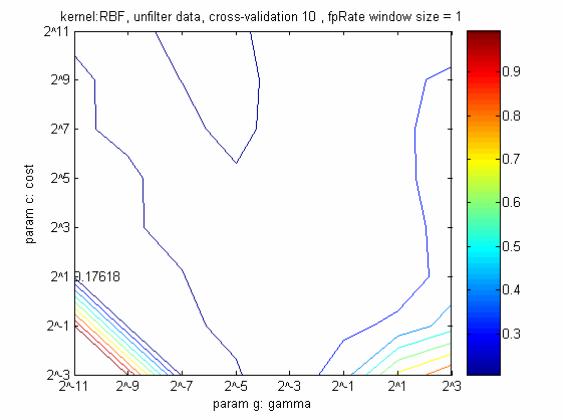
Un-filter data with cross-validation Precision

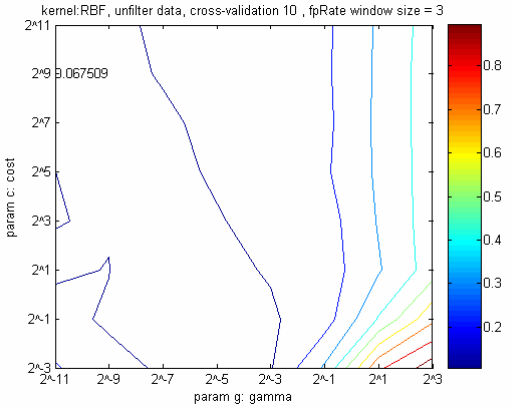


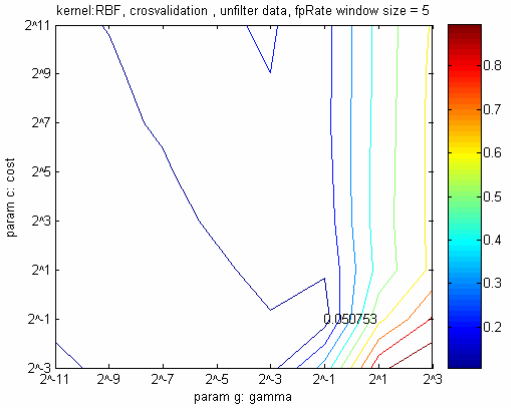




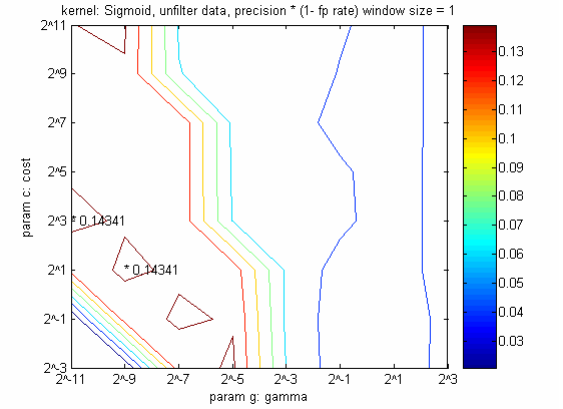
Un-filter data with cross-validation False Positive Rate

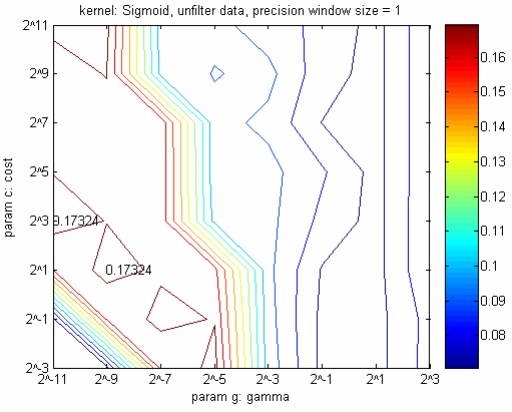


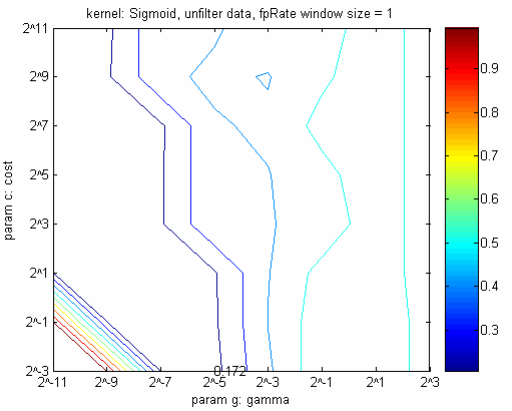


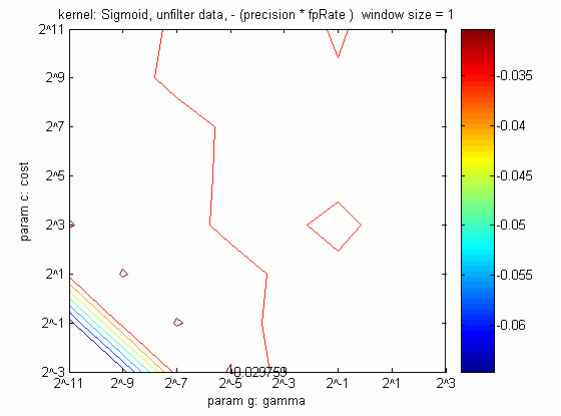


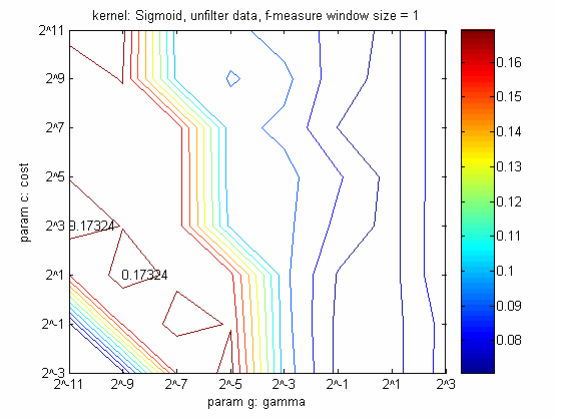
APPENDIX I: FIGURES FOR SIGMOID KERNEL:

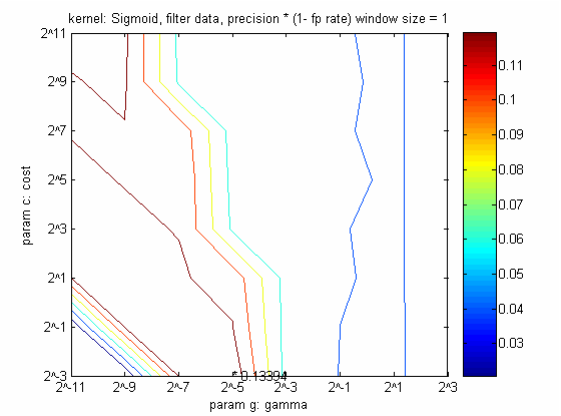


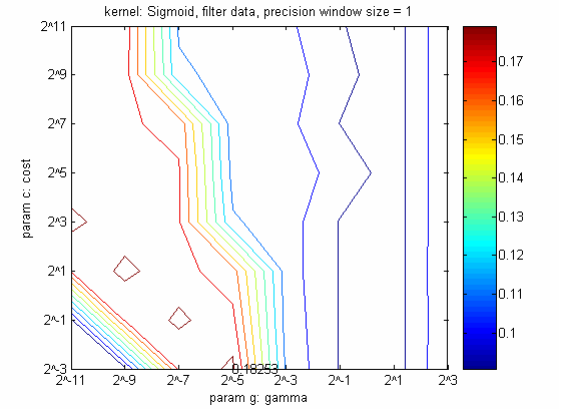


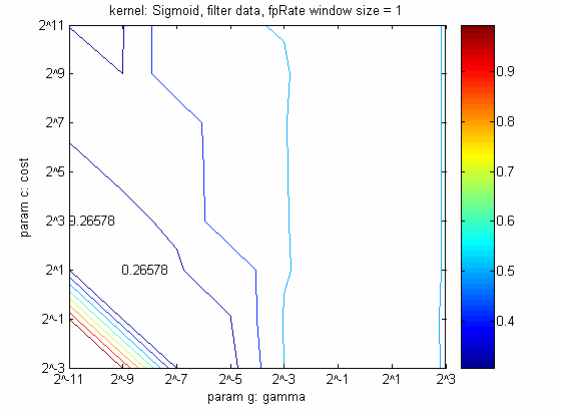


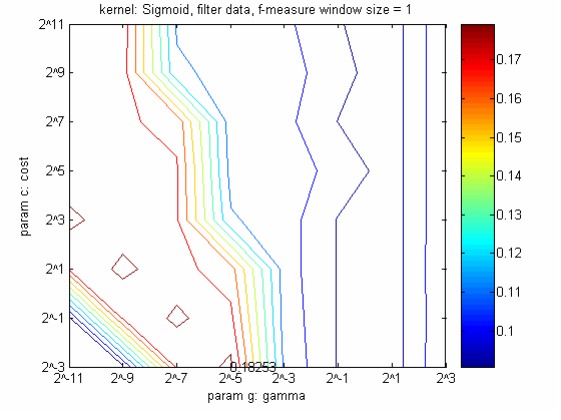

kernel: Sigmoid, unfilter data, - (precision * fpRate) window size = 1
param c: cost
param g: gamma
2^11
2^9
2^7
2^5
2^3
2^1
2^-1
2^-3
2^-11
2^-9
2^-7
2^-5
2^-3
2^-1
2^1
2^3
-0.035
-0.04
-0.045
-0.05
-0.055
-0.06
0.029759

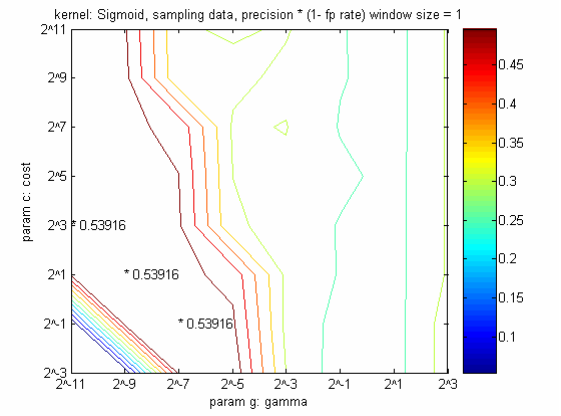

kernel: Sigmoid, unfilter data, f-measure window size = 1
param c: cost
param g: gamma
0.16
0.15
0.14
0.13
0.12
0.11
0.1
0.09
0.08
0.17324
0.17324

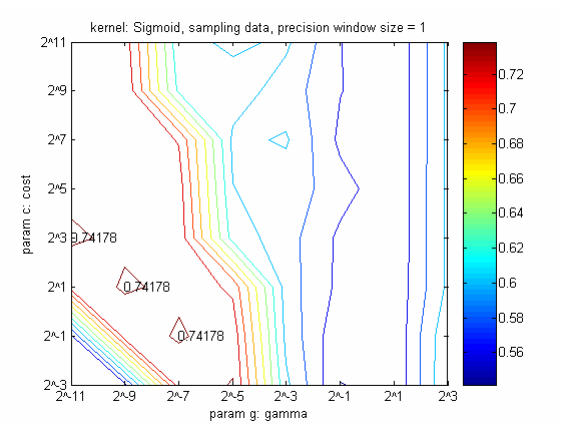

kernel: Sigmoid, filter data, precision * (1- fp rate) window size = 1
param c: cost
param g: gamma
0.11
0.1
0.09
0.08
0.07
0.06
0.05
0.04
0.03
0.13394

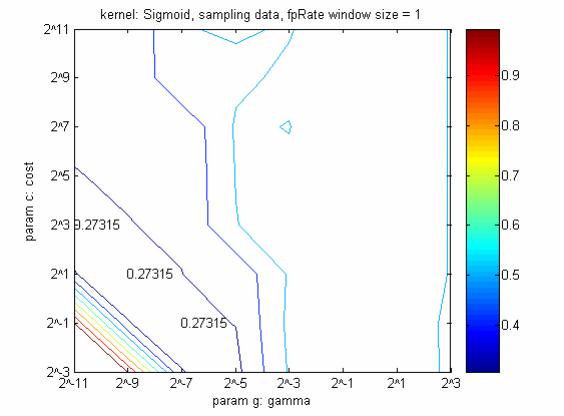

kernel: Sigmoid, filter data, precision window size = 1
param c: cost
param g: gamma
0.17
0.16
0.15
0.14
0.13
0.12
0.11
0.1
0.18253

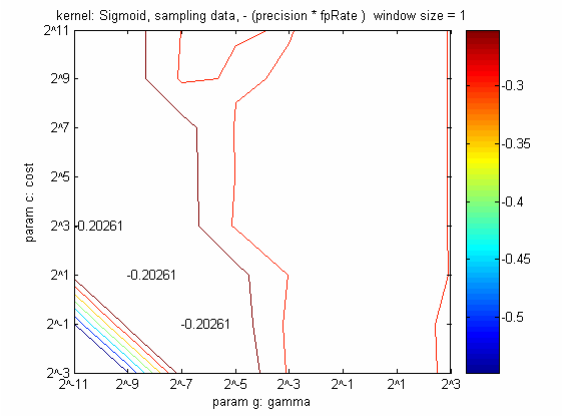

kernel: Sigmoid, filter data, fpRate window size = 1
param c: cost
param g: gamma
0.9
0.8
0.7
0.6
0.5
0.4
0.26578
0.26578

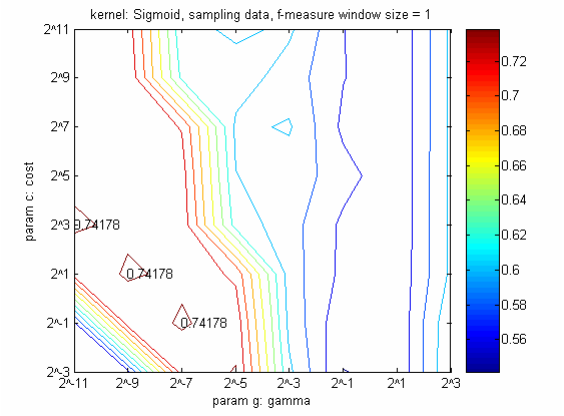

kernel: Sigmoid, filter data, f-measure window size = 1
param c: cost
param g: gamma
0.17
0.16
0.15
0.14
0.13
0.12
0.11
0.1
0.18253

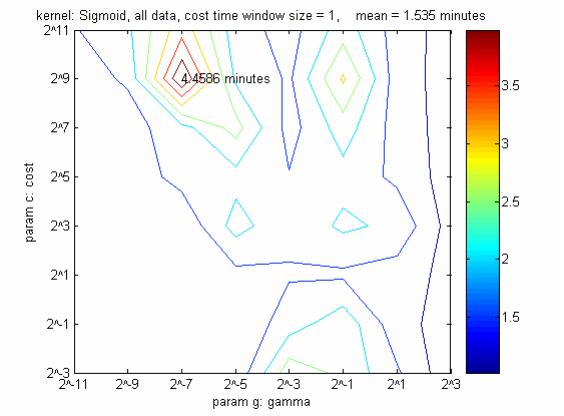

kernel: Sigmoid, sampling data, precision * (1- fp rate) window size = 1
param c: cost
param g: gamma
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.53916
0.53916
0.53916

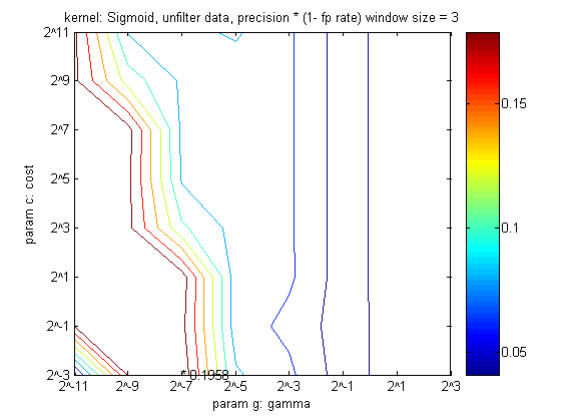

kernel: Sigmoid, sampling data, precision window size = 1
param c: cost
param g: gamma
0.72
0.7
0.68
0.66
0.64
0.62
0.6
0.58
0.56
0.74178
0.74178
0.74178

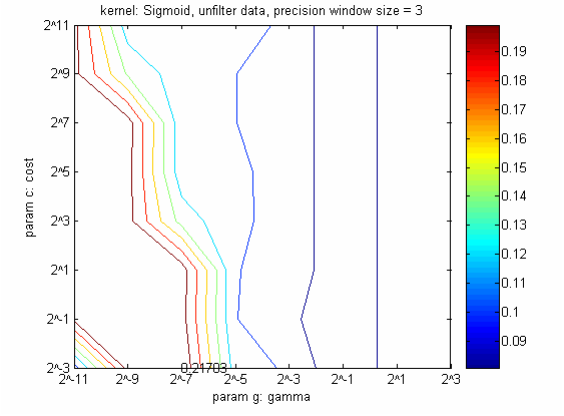

kernel: Sigmoid, sampling data, fpRate window size = 1
param c: cost
param g: gamma
0.27315
0.27315
0.27315

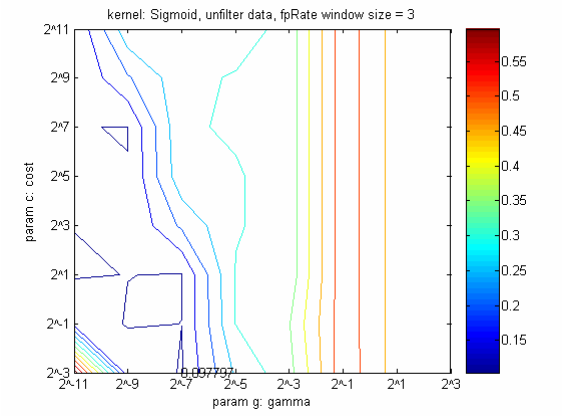

kernel: Sigmoid, sampling data, - (precision * fpRate) window size = 1
param c: cost
param g: gamma
-0.20261
-0.20261
-0.20261

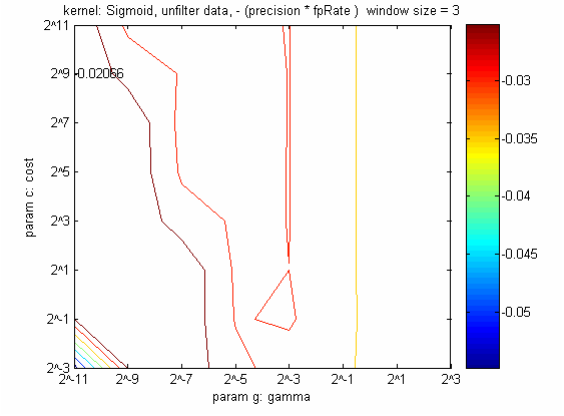

kernel: Sigmoid, sampling data, f-measure window size = 1
param c: cost
param g: gamma
0.74178
0.74178
0.74178

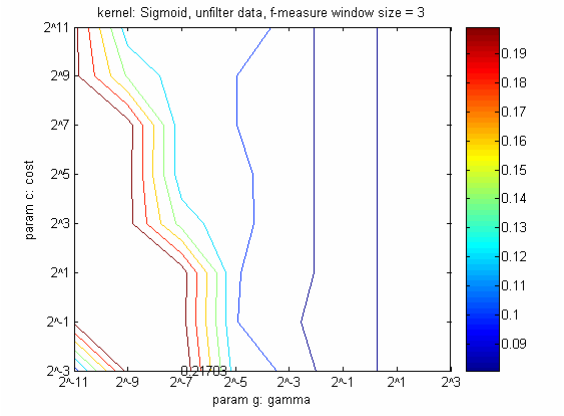

kernel: Sigmoid, all data, cost time window size = 1, mean = 1.535 minutes
param c: cost
param g: gamma
4.4586 minutes

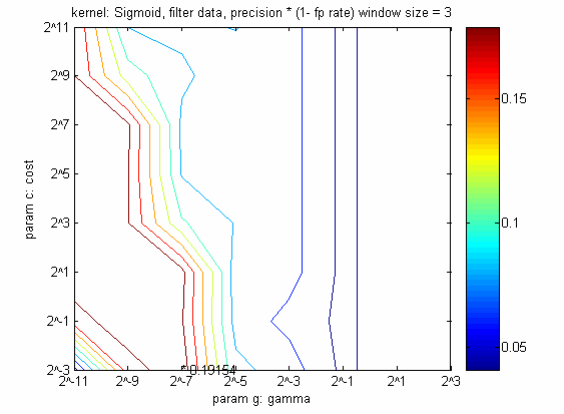

kernel: Sigmoid, unfilter data, precision * (1- fp rate) window size = 3
param c: cost
param g: gamma
0.11958

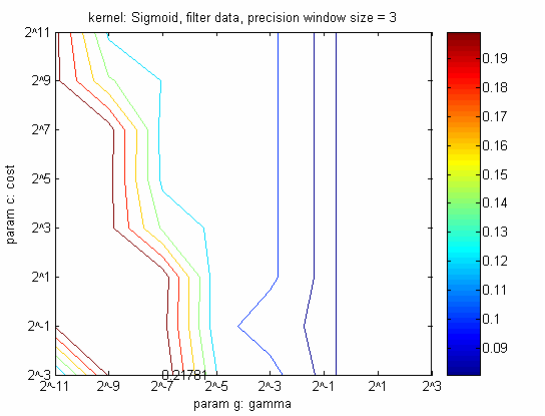

kernel: Sigmoid, unfilter data, precision window size = 3
param c: cost
param g: gamma
0.21703

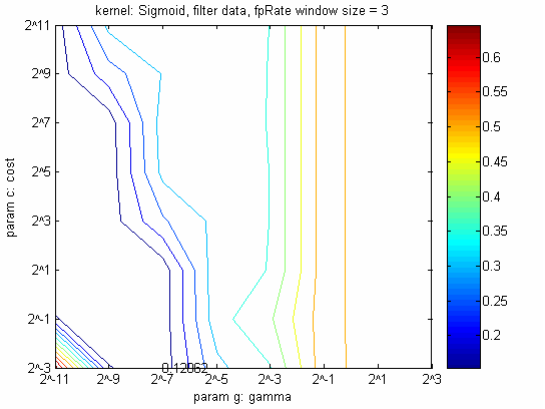

kernel: Sigmoid, unfilter data, fpRate window size = 3
param c: cost
param g: gamma
0.007797

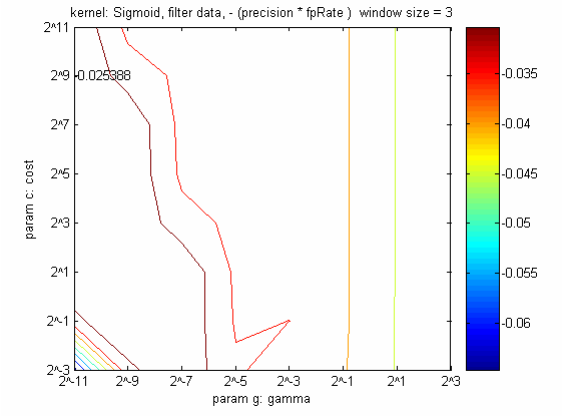

kernel: Sigmoid, unfilter data, - (precision * fpRate) window size = 3
param c: cost
param g: gamma
-0.02066

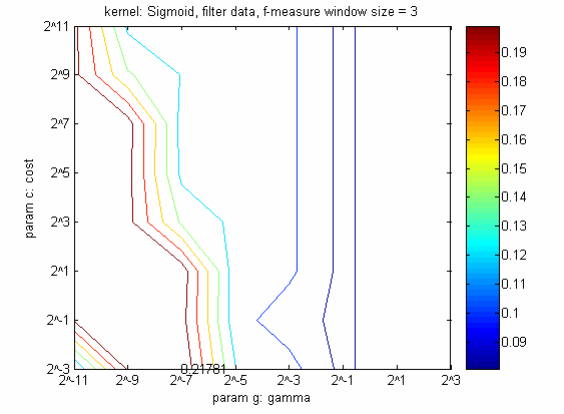

kernel: Sigmoid, unfilter data, f-measure window size = 3
param c: cost
param g: gamma
0.21703

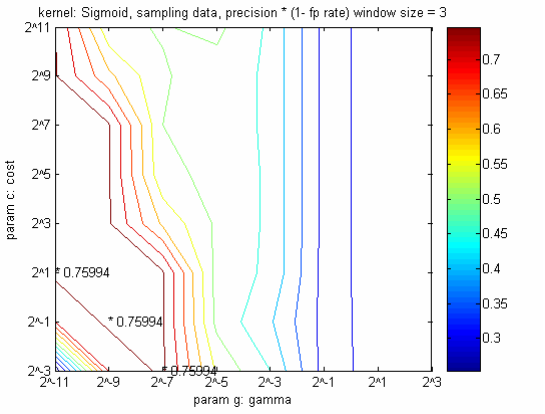

kernel: Sigmoid, filter data, precision * (1- fp rate) window size = 3
param c: cost
param g: gamma
0.19154

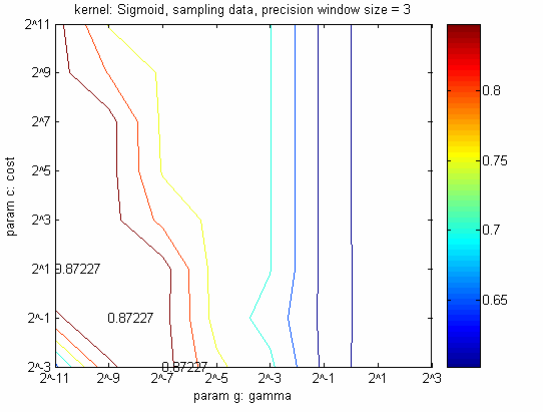

kernel: Sigmoid, filter data, precision window size = 3
param c: cost
param g: gamma
0.21781

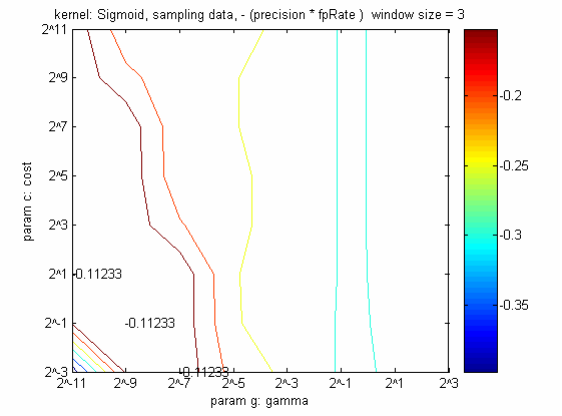

kernel: Sigmoid, filter data, fpRate window size = 3
param c: cost
param g: gamma
0.12062

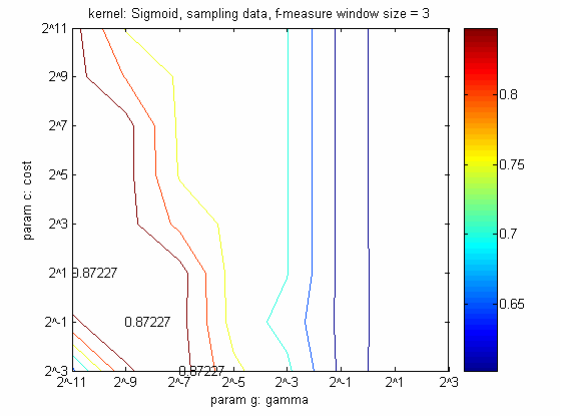

kernel: Sigmoid, filter data, - (precision * fpRate) window size = 3
param c: cost
param g: gamma
0.025388

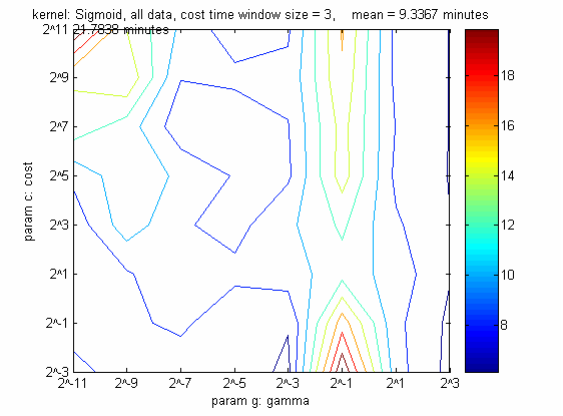

kernel: Sigmoid, filter data, f-measure window size = 3
param c: cost
param g: gamma
0.21781


kernel: Sigmoid, sampling data, precision * (1- fp rate) window size = 3
param c: cost
param g: gamma
0.75994
0.75994
0.75994


kernel: Sigmoid, sampling data, precision window size = 3
param c: cost
param g: gamma
0.87227
0.87227
0.87227


kernel: Sigmoid, sampling data, - (precision * fpRate) window size = 3
param c: cost
param g: gamma
2^11
2^9
2^7
2^5
2^3
2^1
2^-1
2^-3
2^-11
2^-9
2^-7
2^-5
2^-3
2^-1
2^1
2^3
-0.11233
-0.11233
-0.11233
-0.2
-0.25
-0.3
-0.35


kernel: Sigmoid, sampling data, f-measure window size = 3
param c: cost
param g: gamma
2^11
2^9
2^7
2^5
2^3
2^1
2^-1
2^-3
2^-11
2^-9
2^-7
2^-5
2^-3
2^-1
2^1
2^3
0.87227
0.87227
0.87227
0.8
0.75
0.7
0.65


kernel: Sigmoid, all data, cost time window size = 3, mean = 9.3367 minutes
21.7838 minutes
param c: cost
param g: gamma
2^11
2^9
2^7
2^5
2^3
2^1
2^-1
2^-3
2^-11
2^-9
2^-7
2^-5
2^-3
2^-1
2^1
2^3
18
16
14
12
10
8

APPENDIX J: CONTENT OF THE CD.

Are too many information about the experiments for show in paper. So I decide live the real values that made the graphs and all the implementation and work in these CD.

The CD contains all the information need it for extend this project.

The CD contains:

1. The whole report.
2. All the appendixes.
3. All the modules implemented.
4. All datasets.
5. All the results.