1 Introduction

Comparison of various linear classifiers on artificial datasets.

The aim of supervised learning is inferring a function $f(.)$ between target attribute Y that we want to explain/predict, and one or more input attributes (descriptors) $(X_1, X_2, \ldots, X_p)$. The function may have parameters i.e. $Y = f(X_1, X_2, \ldots, X_p ; \alpha)$, where α is a vector of parameters related to $f(.)$.

We are faced with two issues in this process.

The first issue is the choice of the function $f(.)$. We are talking about "representation bias". There are a multitude of possible forms of relationship between the target and the input variables. Linear models are generally distinguished from nonlinear models. A priori, we always have an interest in choosing the most complex formulation, i.e. a nonlinear model: "which can do more, can do less". In fact, the situation is more subtle. The relationship between the descriptors and the target variable is a conjecture. Trying to express a hypothetical causality with a mathematical function is always risky. Thus, some authors advocate, at first when we have no idea about the nature of the relationship between the target and the descriptors, to check the behavior of linear models on the data that we deal with (Duda and al., 2001; page 215).

The second problem is the calculating the parameters of the function $f(.)$. We want to build the most effective function possible on the population. But we only have one sample, called training sample, for the learning process. The "search bias" describes how to explore the solutions. It allows you to choose between various competing solutions. It also helps to restrict the search. Often, but this is not always the case, the search bias is expressed by the criterion to be optimized during the learning process (maximum likelihood, least squares, margin maximization, etc.). A priori, we have an interest in choosing a method that explores all possible hypothesis so as to choose the best one. But this is not as simple as that. We run the risk of learning the noise of the data instead of the underlying relationships between the variables. This phenomenon is called "overfitting" i.e. the algorithm incorporates in the predictive model informations specific to the learning sample which are irrelevant in the population. The situation is even more difficult as it is likely that some descriptors are not relevant for the prediction. They can disturb the learning process.

In this tutorial, we study the behavior of 5 linear classifiers on artificial data. Linear models are often the baseline approaches in supervised learning. Indeed, based on a simple linear combination of predictive variables, they have the advantage of simplicity: the reading of the influence of each descriptor is relatively easy (signs and values of the coefficients); learning techniques are often (not always) fast, even on very large databases. We are interested in: (1) the naive bayes classifier; (2) the linear discriminant analysis; (3) the logistic regression; (4) the perceptron (single-layer perceptron); (5) the support vector machine (linear SVM).

We are in a particular context for the data. We generate an artificial dataset for a binary problem i.e. target attribute Y has 2 possible values {positive, negative}. The number of predictive variables can be parameterised (p ≥ 2), but only the first two ones (X1, X2) are relevant. The boundary which enables to distinguish the positives from the negatives instances is represented by a straight line in a two-dimensional representation space (X1, X2). To increase the difficulty, we can randomly add noise to the labels. We will then see which methods are the best ones, according to the size of the learning sample and the number of descriptors.

The experiment was conducted under R. The source code accompanies this document. My idea, besides the theme of the linear classifiers that concerns us, is also to describe the different stages of the elaboration of an experiment for the comparison of learning techniques. In addition, we show also the results provided by the linear approaches implemented in various tools such as Tanagra, Knime, Orange, Weka and RapidMiner.

2 Dataset

We use the following R source code to generate the dataset with n instances and p descriptors (+ the target attribute). The level of noise is also a parameter.

```
#function for generating dataset
generate.data <- function(n=100, p=2, noise=0.05){
  #generating the descriptors
  X <- data.frame(lapply(1:p,function(x){runif(n)}))
  colnames(X) <- paste("x",1:p,sep="")
  #create the labels
  y.clean <- ifelse(X$x2 > 4*X$x1,1,2)
  #possible adding noise (if noise > 0)
  y <- factor(ifelse(runif(n)>(1.0-noise),3-y.clean,y.clean))
  levels(y) <- c("neg","pos")
  all.data <- cbind(y,X)
  return(all.data)
}
```

The descriptors follows a uniform distribution $U(0, 1)$. The label is assigned according to the following classification rule:

IF (X2 > 4 * X1) **THEN** Y = Negative **ELSE** Y = Positive

We have imbalanced data with approximately: negative = 12.5%, positive = 87. 5%.

The noise is added on the class attribute by turning the label around in of $(100{*}\text{noise})$% cases i.e. each individual, whatever is class value, has (100*noise)% of chance to have a modified label (*Note: this mode of modification may change the proportions of the positive and negative instances in the dataset*). By construction, it is impossible with a linear classifier to obtain an error rate lower than “noise” (occasionally on some test samples, but not in average).

We can observe the theoretical separating boundary for a sample with n = 20,000 instances, p = 2 descriptors and noise = 5% (Figure 1). We use the following source code to generate the dataset.

```
#number of descriptors
p <- 2
#noise on the class attribute - theoretical error rate
noise <- 0.05
#generating test set
n.test <- 20000
test.data <- generate.data(n.test,p,noise)
#plotting test set
plot(test.data$x1,test.data$x2,pch=21,bg=c("red","blue")[unclass(test.data$y)])
```

We can see also the data points (5% of the instances) which are in the bad side of the boundary.

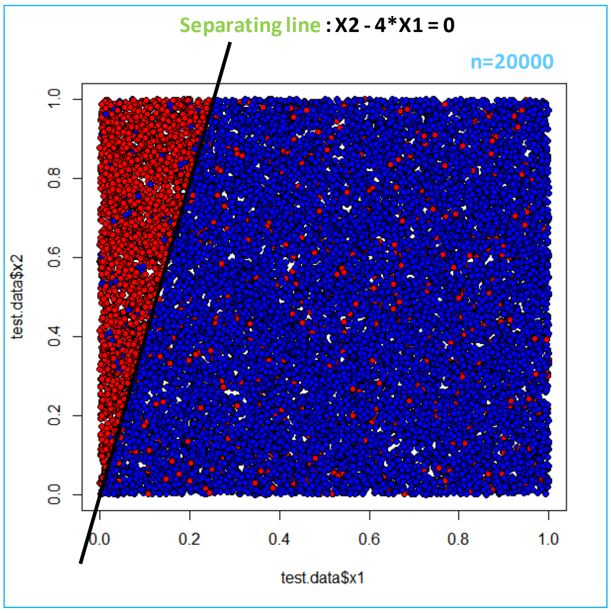


Figure 1 – Theoretical boundary on n = 20,000 instances

Unfortunately, and this is the crucial problem of the supervised learning process, the labeled data are rare, difficult to obtain in some contexts. Here, we generate a learning set with n = 300 instances with the same characteristics (underlying concept, number of descriptors, level of noise).

```
#training set size
n.train <- 300
#training set
train.data <- generate.data(n.train,p,noise)
#plotting training set
plot(train.data$x1,train.data$x2,pch=21,bg=c("red","blue")[unclass(train.data$y)])
```

The boundary is less obvious on a sample with n = 300 instances (Figure 2). If we try to draw it freehand, we do not really find the right solution. And moreover, if we have another learning sample of the same size, the boundary line induced will be (a little) different.

And yet, the learning algorithms only has this information (n.train = 300) to try to detect the “true” boundary line. This is the main issue of the supervised learning process.

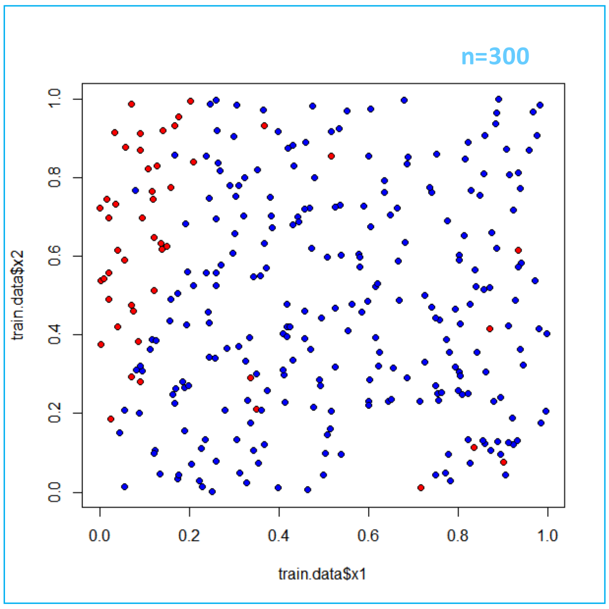


Figure 2 - Sample of 300 observations submitted to the supervised learning algorithms

Note: The value of using artificial data is that we control the conditions of the experiment fully. For instance, we know very well that our dataset goes against the assumptions underlying the discriminant analysis or naive Bayesian (conditional normality). The nature of the results should not surprise us, these methods will be disavantaged. On the other hand, the magnitude of the differences will be interesting to study, especially in terms of the number of instances and the number of irrelevant descriptors. In addition, we can modify the level of noise. Because we know the underlying concept to learn, we know in advance the best error rate that we can get.

3 Comparison of linear classifiers

We evaluate various learning algorithms in this section. The outline is always the same: we learn the model on the training set (n.train = 300 instances) (Figure 2); we measure the error rate on a second sample (which serves as test set, n.test = 20,000 instances) (Figure 1); we compare the inferred boundary with the theoretical separation line.

We use the following R source code to measure the error rate and visualize the boundary:

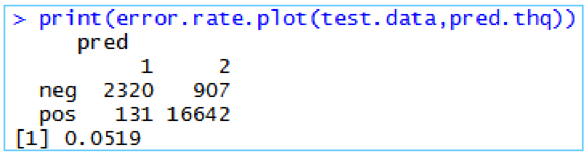
```
#function for computing error rate
#plotting the data points and the separation line
#data.test is the test set, data.test$y the target attribute
#pred is the prediction of the classifier to evaluate
#the function displays the confusion matrix and returns the error rate
error.rate.plot <- function(data.test,pred){
  #displaying the data points according to their class membership
  plot(data.test$x1,data.test$x2,pch=21,bg=c("red","blue")[unclass(pred)])
  #the boundary is a straight line
  abline(0,4,col="green",lwd=5)
  #confusion matrix and error rate
  mc <- table(data.test$y,pred)
  print(mc)
  err.rate <- 1-sum(diag(mc))/sum(mc)
  return(err.rate)
}
```

3.1 Theoretical model

First, to evaluate the baseline configuration, we calculated the error rate of the theoretical model using the following instructions:

```
#theoretical prediction
pred.thq <- factor(ifelse(test.data$x2-4*test.data$x1>0,1,2))
print(error.rate.plot(test.data,pred.thq))
```

The test error rate is 5.19%, close to the theoretical error (5%). The measured test error rate will be all the more accurate as we increase the size of the test sample.



```
> print(error.rate.plot(test.data,pred.thq))
     pred
          1     2
  neg  2320   907
  pos   131 16642
[1] 0.0519
```

The theoretical boundary (green) is perfectly reproduced. In red, we have the prediction of the negative instances, in blue the positive instances (Figure 3).

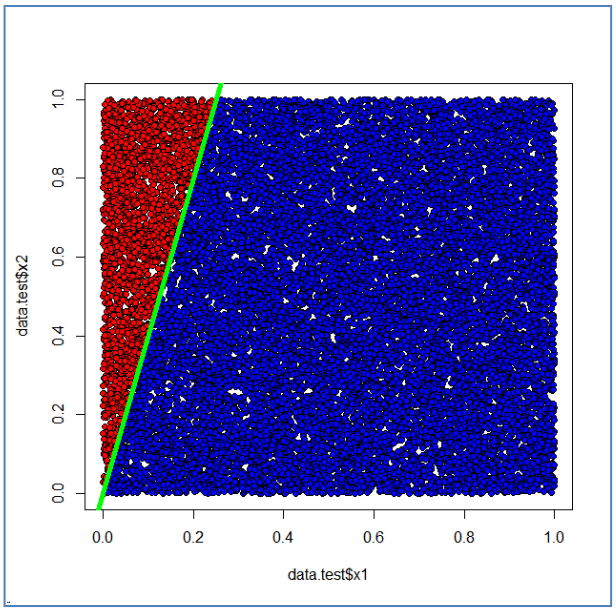


Figure 3 - Prediction of the theoretical model on the test set

3.2 Naive bayes classifier

The naïve bayes classifier is based on two assumptions about the conditional distribution of the descriptors over the class attribute Y: the conditional independence of the descriptors i.e. the descriptors are independents conditionally to the values of Y; each descriptor follows a Gaussian distribution for a given value of Y.

These two assumptions are not really observed on our dataset. The conditional densities show that the Gaussian nature of the distributions is not really credible (Figure 4)[1].

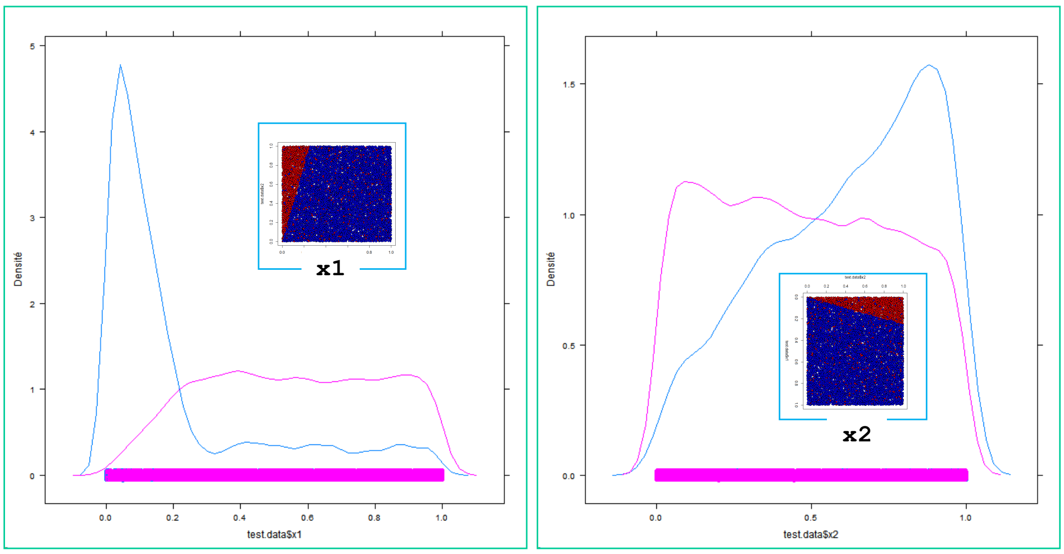


Figure 4 - Distributions of X1 and X2 conditionally to Y

Let us see what happens when we learn and test the naive bayes algorithm on our dataset.

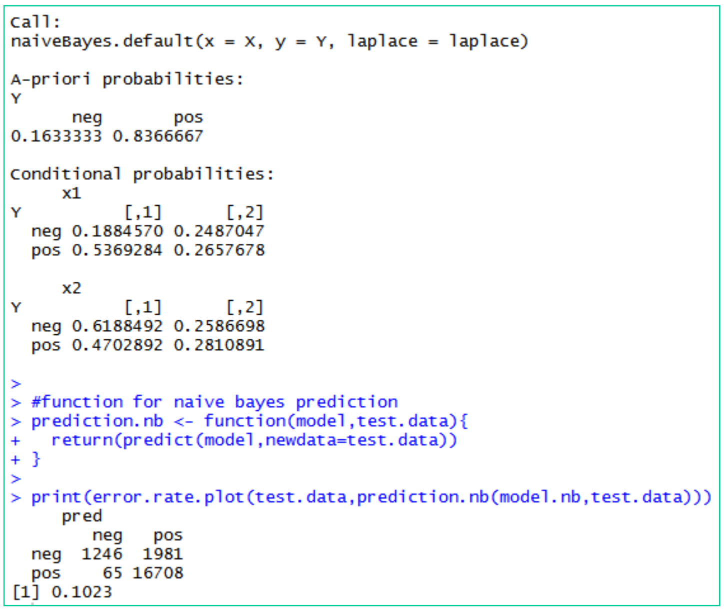
```
#load the e1071 package that we must install before
library(e1071)
#learning process - training set
model.nb <- naiveBayes(y ~ ., data = train.data)
print(model.nb)
#function for prediction on the test set
prediction.nb <- function(model,test.data){
  return(predict(model,newdata=test.data))
}
#plotting the boundary, measuring the test error rate
print(error.rate.plot(test.data,prediction.nb(model.nb,test.data)))
```

R displays the conditional average and standard deviation for X1 and X2.

The test error rate is **10.23%**. We are far from the theoretical error rate (5%).

[1] We use the following R code to obtain these graphs:

```
library(lattice)
densityplot(test.data$x1,groups=test.data$y)
densityplot(test.data$x2,groups=test.data$y)
```



```
Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
      neg       pos 
0.1633333 0.8366667 

Conditional probabilities:
     x1
Y          [,1]      [,2]
  neg 0.1884570 0.2487047
  pos 0.5369284 0.2657678

     x2
Y          [,1]      [,2]
  neg 0.6188492 0.2586698
  pos 0.4702892 0.2810891

> 
> #function for naive bayes prediction
> prediction.nb <- function(model,test.data){
+   return(predict(model,newdata=test.data))
+ }
> 
> print(error.rate.plot(test.data,prediction.nb(model.nb,test.data)))
     pred
        neg   pos
  neg  1246  1981
  pos    65 16708
[1] 0.1023
```

Indeed, the inferred boundary is shifted compared with the optimal separating line (Figure 5).

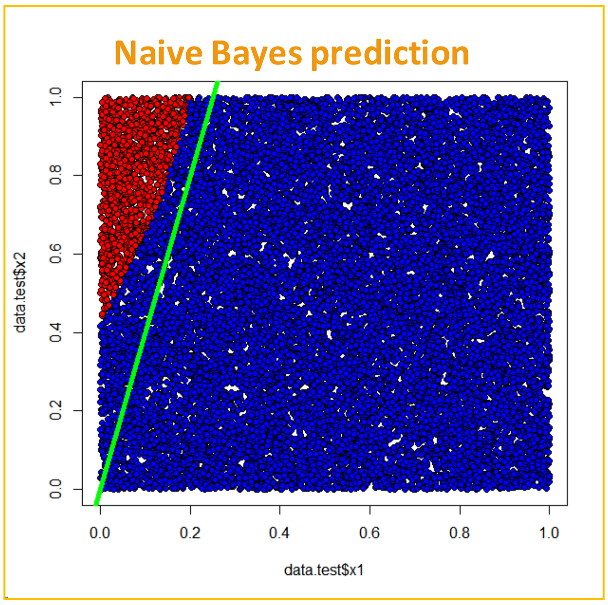


Figure 5 – Boundary inferred by naïve bayes classifier (in green the right separating line)

We can infer from the information provided by R the coefficients of the equation defining the separation line. We will see below that Tanagra can provide them directly (it will the same for linear discriminant analysis and linear SVM) (section 4).

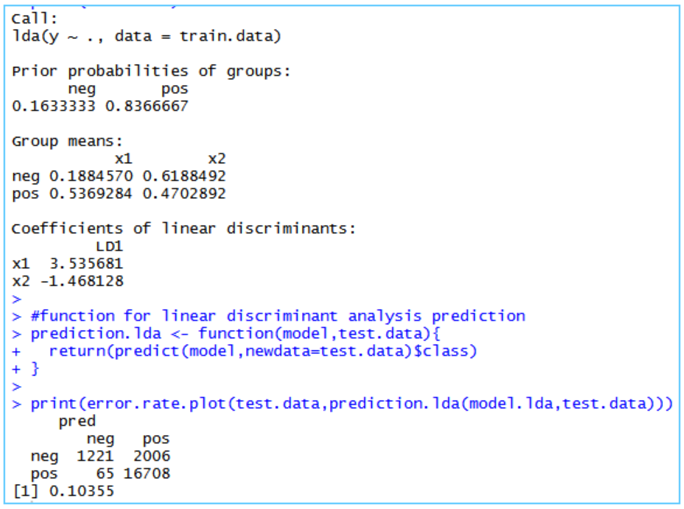
3.3 Linear discriminant analysis

Linear discriminant analysis is also a parametric machine learning algorithm. It is based on two assumptions: the descriptors follow a multivariate gaussian distribution conditionally to the values of the target attribute Y; the conditional covariances matrices are the same i.e. the shapes of the class-conditional point clouds are identical.

A quick glance to the scatter graph above (Figure 1) shows that these assumptions are not satisfied on our dataset. We know however that linear discriminant analysis is rather robust. We check if this is true on our dataset.

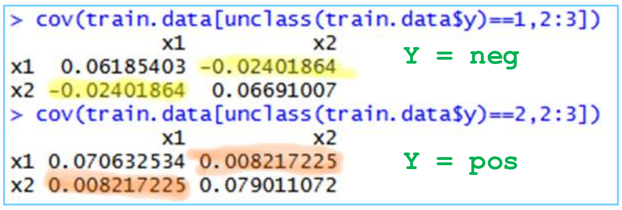
```
#load the MASS package
library(MASS)
#learning process
model.lda <- lda(y ~ ., data = train.data)
print(model.lda)
#function for prediction on the test set
prediction.lda <- function(model,test.data){
  return(predict(model,newdata=test.data)$class)
}
#graph and test error rate
print(error.rate.plot(test.data,prediction.lda(model.lda,test.data)))
```

This is not much better compared to naive bayes classifier. The test error rate is equal to **10.35%**,



```
Call:
lda(y ~ ., data = train.data)

Prior probabilities of groups:
      neg       pos 
0.1633333 0.8366667 

Group means:
           x1        x2
neg 0.1884570 0.6188492
pos 0.5369284 0.4702892

Coefficients of linear discriminants:
         LD1
x1  3.535681
x2 -1.468128
> 
> #function for linear discriminant analysis prediction
> prediction.lda <- function(model,test.data){
+   return(predict(model,newdata=test.data)$class)
+ }
> 
> print(error.rate.plot(test.data,prediction.lda(model.lda,test.data)))
     pred
        neg   pos
  neg  1221  2006
  pos    65 16708
[1] 0.10355
```

Alike naive bayes, the boundary is shifted (Figure 6). Essentially because the conditional point clouds have not identical size and shape. We find the same characteristic when we display the conditional covariance matrices below. They are very different.



```
> cov(train.data[unclass(train.data$y)==1,2:3])
            x1          x2
x1  0.06185403 -0.02401864      Y = neg
x2 -0.02401864  0.06691007
> cov(train.data[unclass(train.data$y)==2,2:3])
            x1          x2
x1 0.070632534 0.008217225      Y = pos
x2 0.008217225 0.079011072
```

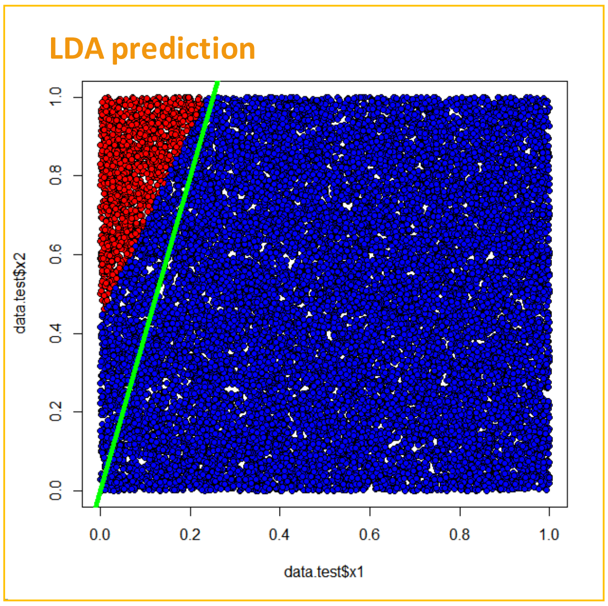


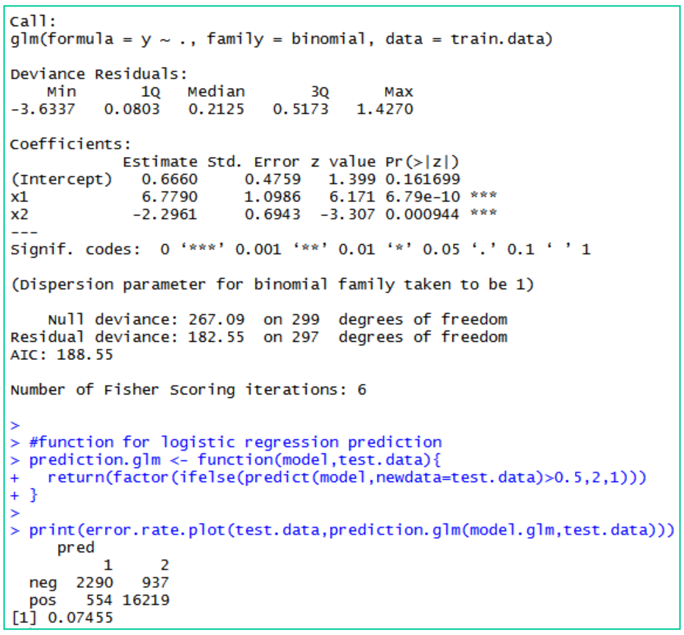
Figure 6 - Boundary inferred by linear discriminant analysis (in green the right separating line)

3.4 Logistic regression

Logistic regression is a statistical approach. Its main assumption is the linearity of the logit (log-odds). The second characteristic is that the output is considered to have an underlying probability distribution belonging to the family of exponential distributions. These include the normal distribution underlying the discriminant analysis which can be considered as one particular case (Bardos, 2001; page 64). Thus, the logistic regression is based on less restrictive assumptions.

```
#logistic regression
model.glm <- glm(y ~ ., data = train.data, family = binomial)
print(summary(model.glm))

#function for logistic regression prediction
prediction.glm <- function(model,test.data){
  return(factor(ifelse(predict(model,newdata=test.data)>0.5,2,1)))
}

#error rate
print(error.rate.plot(test.data,prediction.glm(model.glm,test.data)))
```

The obtained test error rate is **7.45%**, much better than those of naive bayes classifier or discriminant analysis.



```
Call:
glm(formula = y ~ ., family = binomial, data = train.data)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.6337   0.0803   0.2125   0.5173   1.4270

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   0.6660     0.4759   1.399 0.161699
x1            6.7790     1.0986   6.171 6.79e-10 ***
x2           -2.2961     0.6943  -3.307 0.000944 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 267.09  on 299  degrees of freedom
Residual deviance: 182.55  on 297  degrees of freedom
AIC: 188.55

Number of Fisher Scoring iterations: 6

>
> #function for logistic regression prediction
> prediction.glm <- function(model,test.data){
+   return(factor(ifelse(predict(model,newdata=test.data)>0.5,2,1)))
+ }
>
> print(error.rate.plot(test.data,prediction.glm(model.glm,test.data)))
     pred
         1     2
  neg  2290   937
  pos   554 16219
[1] 0.07455
```

We are approaching the theoretical separation line (Figure 7).

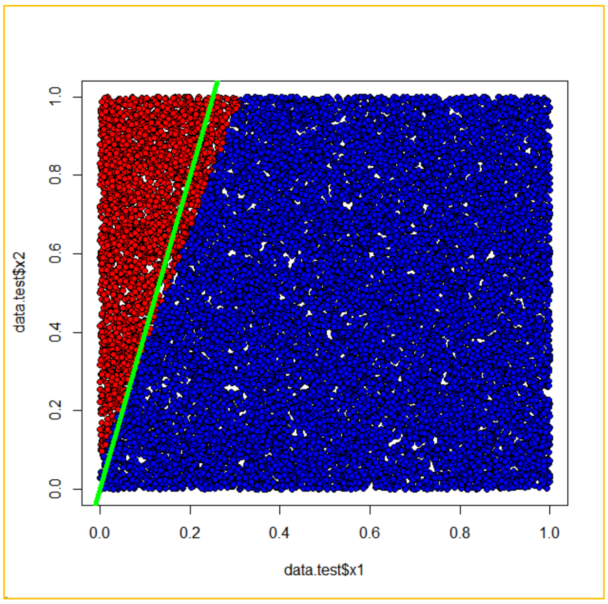


Figure 7 - Boundary inferred by logistic regression (in green the right separating line)

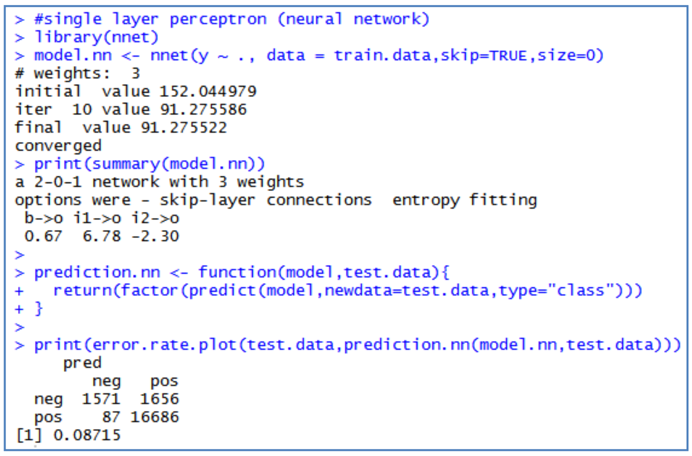
3.5 Perceptron (single-layer perceptron)

The Perceptron is a nonparametric approach. It minimizes a least squares criterion. In the case of a single-layer perceptron, we have a linear classifier.

We install and load the **nnet** package first under R.

```
#single layer perceptron (neural network)
library(nnet)
model.nn <- nnet(y ~ ., data = train.data,skip=TRUE,size=0)
print(summary(model.nn))
#function for the prediction
prediction.nn <- function(model,test.data){
  return(factor(predict(model,newdata=test.data,type="class")))
}
#graph and error rate
print(error.rate.plot(test.data,prediction.nn(model.nn,test.data)))
```

The test error rate is **8.715%**.



```
> #single layer perceptron (neural network)
> library(nnet)
> model.nn <- nnet(y ~ ., data = train.data,skip=TRUE,size=0)
# weights:  3
initial  value 152.044979
iter  10 value 91.275586
final  value 91.275522
converged
> print(summary(model.nn))
a 2-0-1 network with 3 weights
options were - skip-layer connections  entropy fitting
 b->o i1->o i2->o
 0.67  6.78 -2.30
>
> prediction.nn <- function(model,test.data){
+   return(factor(predict(model,newdata=test.data,type="class")))
+ }
>
> print(error.rate.plot(test.data,prediction.nn(model.nn,test.data)))
     pred
        neg   pos
  neg  1571  1656
  pos    87 16686
[1] 0.08715
```

We have the same shift than the other previous approaches compared with the optimal separation line (Figure 8).

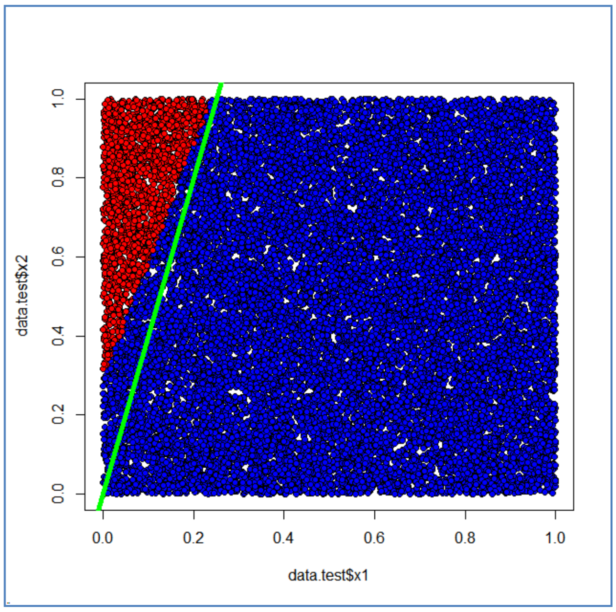


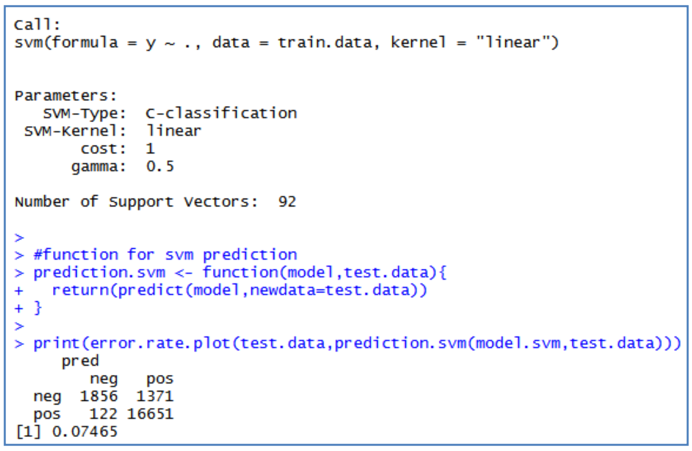
Figure 8 - Boundary inferred by perceptron (in green the right separating line)

3.6 Support vector machine

With a linear kernel, the model coming from the support vector machine (SVM) algorithm is a linear classifier. Yet, few software provides the explicit equation of the separation line (or the hyperplane in higher dimension, p > 2). The procedure **svm()** from the **e1071** package for instance simply provides the list of support points. We have no information about the influence of the descriptors, even if we can deploy the model on unseen instance with this information.

```
#linear support vector machine
library(e1071)
model.svm <- svm(y ~ ., data = train.data,kernel="linear")
print(model.svm)
#function for svm prediction
prediction.svm <- function(model,test.data){
  return(predict(model,newdata=test.data))
}
#graph and error rate
print(error.rate.plot(test.data,prediction.svm(model.svm,test.data)))
```

The test error rate is **7.465%**. This is the best model among all the linear methods presented in this section.



```
Call:
svm(formula = y ~ ., data = train.data, kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.5

Number of Support Vectors:  92

>
> #function for svm prediction
> prediction.svm <- function(model,test.data){
+   return(predict(model,newdata=test.data))
+ }
>
> print(error.rate.plot(test.data,prediction.svm(model.svm,test.data)))
     pred
       neg   pos
  neg  1856  1371
  pos   122 16651
[1] 0.07465
```

We are getting closer to the theoretical separation line (Figure 9).

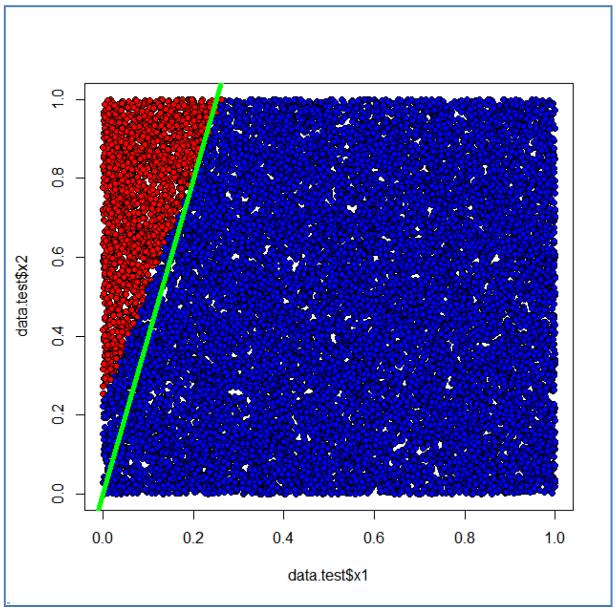


Figure 9 - Boundary inferred by linear SVM (in green the right separating line)

The e1071 package offers an excellent tool for visualizing the regions associated to the classes, and therefore the separation line. It also identifies the support points related to the classifier.

```
#plotting data points "o" and support points "x"
plot(model.svm,data=train.data,svSymbol="x",dataSymbol="o")
```

The scatter graph is transposed in relation to ours (X2 in abscissa, X1 in ordinate). But the nature of the results is quite the same: the regions associated to the classes are linearly delimited. Due to noise (noise = 5%), the support points are relatively numerous despite the simplicity of the underlying concept (Figure 10). They would be less numerous and located along the separation line if the data were not noisy.

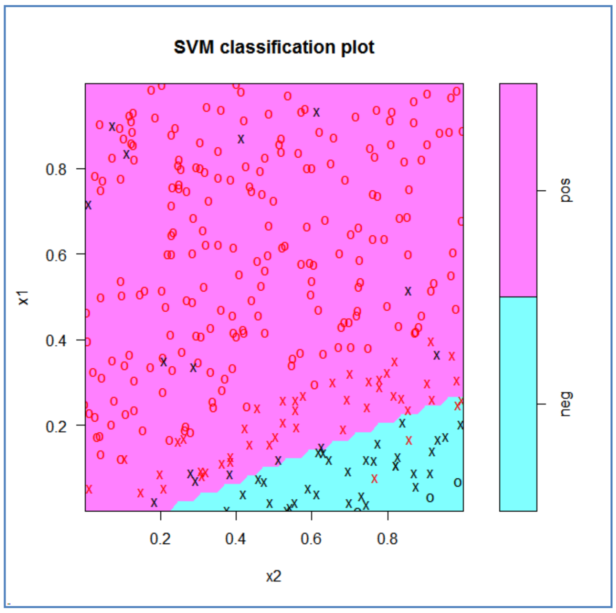


Figure 10 - SVM: regions associated to classes, data points ("o") and support points ("x")

3.7 Behavior of some nonlinear approaches

We know that the frontier separating the classes is linear because it has been intentionally generated. In real studies, we do not have that information. We would therefore have to test different learning algorithms before choosing the model adapted to the problem to be handled. A priori, we said in introduction, nonlinear models with a more efficient representation system would be better. But, in addition to the difficulties of interpreting the results, we are faced with the greater variability of these techniques because they are often more complex. We need more observations to combat overfitting. This is often not possible in practice. We cannot define the learning set size.

In this section, we study the behavior of some nonlinear approaches. We analyze their performance and the shape of the inferred boundary.

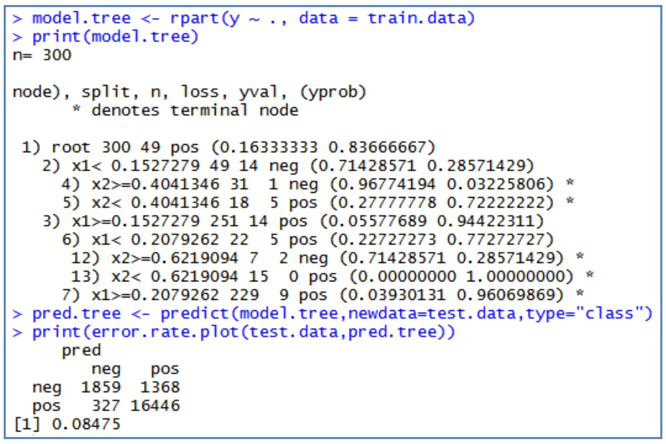
3.7.1 Decision tree – CART

A decision tree is a nonlinear classifier. It divides the feature space into axis-parallel rectangles. When we split a node using a descriptor during the learning process, we define an axis-parallel separation line. The combination of these separations provides a nonlinear classifier. We use the rpart()

procedure coming from the rpart package which implements an approach very similar to the famous CART algorithm (Breiman and al., 1984).

```
#decision tree learning
library(rpart)
model.tree <- rpart(y ~ ., data = train.data)
print(model.tree)
pred.tree <- predict(model.tree,newdata=test.data,type="class")
print(error.rate.plot(test.data,pred.tree))
```

The decision tree uses successively and repeatedly the variables x1 and x2 for the splitting processes.



```
> model.tree <- rpart(y ~ ., data = train.data)
> print(model.tree)
n= 300

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 300 49 pos (0.16333333 0.83666667)
   2) x1< 0.1527279 49 14 neg (0.71428571 0.28571429)
     4) x2>=0.4041346 31  1 neg (0.96774194 0.03225806) *
     5) x2< 0.4041346 18  5 pos (0.27777778 0.72222222) *
   3) x1>=0.1527279 251 14 pos (0.05577689 0.94422311)
     6) x1< 0.2079262 22  5 pos (0.22727273 0.77272727)
      12) x2>=0.6219094 7  2 neg (0.71428571 0.28571429) *
      13) x2< 0.6219094 15  0 pos (0.00000000 1.00000000) *
     7) x1>=0.2079262 229  9 pos (0.03930131 0.96069869) *
> pred.tree <- predict(model.tree,newdata=test.data,type="class")
> print(error.rate.plot(test.data,pred.tree))
     pred
        neg   pos
  neg  1859  1368
  pos   327 16446
[1] 0.08475
```

The test error rate is **8.475%**. Although the estimated decision boundary appears visually rather rough (Figure 11), it is aligned to the separation line. Ultimately, the performances are quite comparable to those of the linear methods.

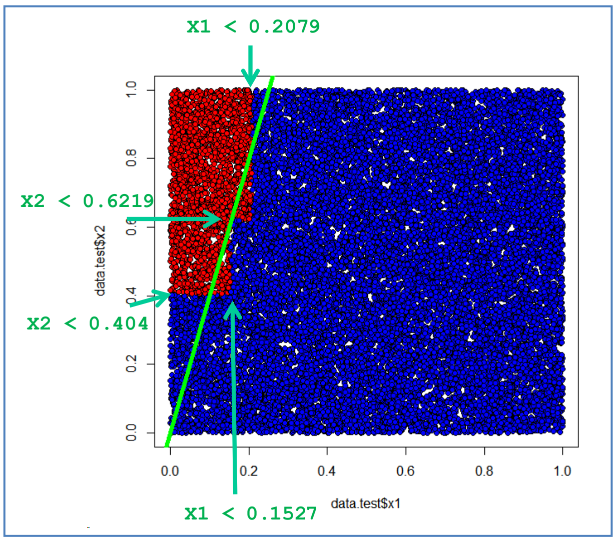


Figure 11 - Boundary inferred by decision tree (in green the right separating line)

The approximation depends on the number of the leaves of trees, which itself is dependent on the size of the learning sample. If it (the learning sample) is infinite size (impossible in practice), the separation is perfectly reproduced. In fact, the performance of trees, more than any other method, depends heavily on the availability of observations.

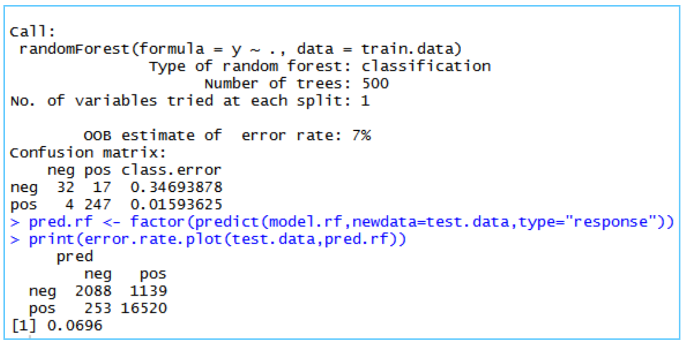
3.7.2 Random Forest

The Random Forest (Breiman & Cutler, 2000) is an ensemble learning method. The idea is to make cooperate an ensemble of decision trees learned from various version of the learning set. The decision tree learning algorithm is also modified to improve the diversity of the trees. One of the main consequences of the approach is that it transcends the constraint of tree representation, to the point of being able to approach the linear boundary directly (because each individual tree is very deep, with a low representation bias), with the same learning sample of 300 observations.

We install and load the **rf** package before using the **randomForest()** procedure.

```
#random forest
library(randomForest)
model.rf <- randomForest(y ~ ., data = train.data)
print(model.rf)
pred.rf <- factor(predict(model.rf,newdata=test.data,type="response"))
print(error.rate.plot(test.data,pred.rf))
```

The test error rate is **6.96%**. This is the best classifier of our comparative study.



```
Call:
 randomForest(formula = y ~ ., data = train.data)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 1

        OOB estimate of  error rate: 7%
Confusion matrix:
    neg pos class.error
neg  32  17  0.34693878
pos   4 247  0.01593625
> pred.rf <- factor(predict(model.rf,newdata=test.data,type="response"))
> print(error.rate.plot(test.data,pred.rf))
     pred
        neg   pos
  neg  2088  1139
  pos   253 16520
[1] 0.0696
```

The classifier is not linear by nature. It is nevertheless able to produce a good approximation of the theoretical decision boundary (Figure 12).

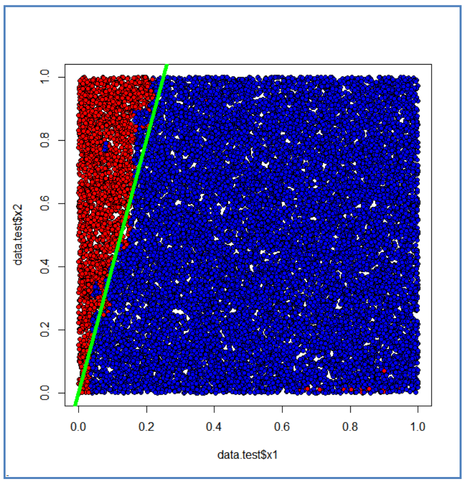


Figure 12 - Boundary inferred by random forest (in green the right separation line)

This is impressive. Especially since, in relation to the characteristics of random forest, we are placed in extreme conditions: the number of instances is low, producing sufficiently dissimilar bootstrap samples is difficult; we only have the two relevant variables in the dataset (no other descriptors), producing sufficiently diversified trees in these conditions is not obvious.

3.7.3 K-Nearest Neighbors

K-Nearest Neighbors is not constrained by a representation system. The parameter "k" (the number of neighbors to account in the prediction of the labels of unseen instance) influence the behavior of the classifier. If we increase "k", we reduce the variance of the approach but increase the bias. On the other hand, if we decrease "k", we can represent complex concept, but overdependence to the learning sample may occur (overfitting). In our experiment, we try "**k = 1**" and "**k = 5**".

The knn() procedure is available into the **class** package. There is not a learning process strictly speaking. We deploy directly the "model" on the test set. First, we evaluate the approach with **k = 1** neighbors used for the classification of the instances.

```
#nearest neighbor
library(class)
#k = 1
print(error.rate.plot(test.data,knn(train.data[,2:3],test.data[,2:3],train.data$y,k=1)))
```

The test error rate is **10.84%**. Performances seem acceptable compared to some linear techniques.

```
> print(error.rate.plot(test.data,knn(train.data[,2:3],test.data[,2:3],train.data$y,k=1)))
     pred
        neg   pos
  neg  2006  1221
  pos   947 15826
[1] 0.1084
```

Indeed, the separation line seems to be more or less accurately reproduced (Figure 13).

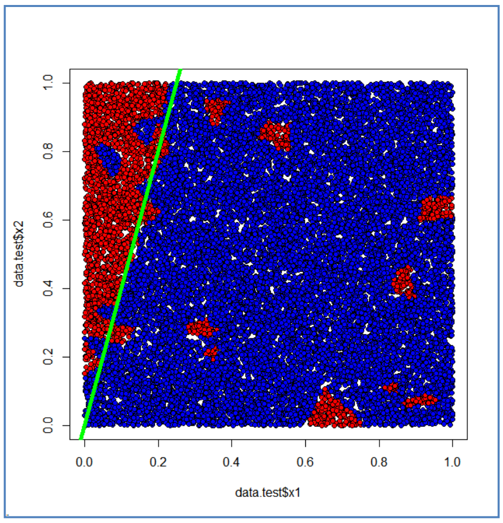


Figure 13 – Regions of assignment inferred by the 1-NN (in green the right separation line)

But we also note that some mislabelled observations have defined areas of erroneous influence on both sides of the theoretical separation line (Figure 13). Of course, these areas would not have existed if we have data without noise on labels.

These areas disappear when we set **k = 5**, improving the error rate (**7.55%**).

```
> #k=5
> print(error.rate.plot(test.data,knn(train.data[,2:3],test.data[,2:3],train.data$y,k=5)))
    pred
       neg   pos
  neg  1976  1251
  pos   260 16513
[1] 0.07555
```

But there are still areas of bad decision along the border line (Figure 14).

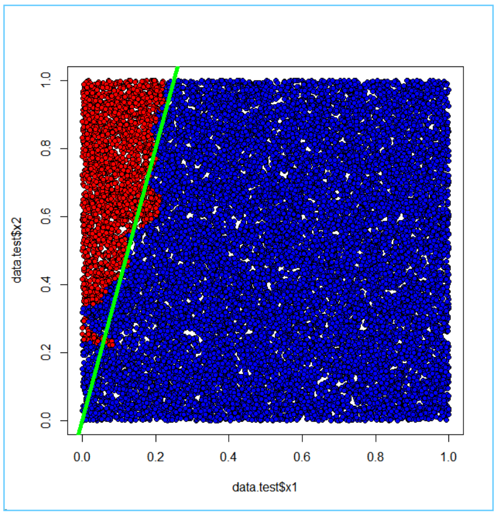


Figure 14 - Regions of assignment inferred by the 5-NN (in green the right separation line)

3.8 Summary of the results

In this section, we make a recap of the results. Of course, we must be cautious about the following table, it comes from the result of an experiment on only one dataset. But maybe we can draw some tendencies however.

Approach	**Test error rate (%)**
Theoretical model	**5.19**
Linear methods	
Naïve bayes	10.23
Discriminant analysis	10.35
Logistic regression	7.45
Perceptron	8.71
Linear SVM	7.46
Nonlinear methods	
Decision tree	8.48
Random Forest	6.96
1-NN	10.84
5-NN	7.55

Finally, apart from the 3 wrong approaches, because based on assumptions that are not adapted to our dataset (naïve bayes, linear discriminant analysis) or because based on bad values of the parameters (1-NN), the methods seem have similar behavior. Except that, and this is a very important element, linear classifiers offer explicit models that can be easily interpreted (the coefficients of the linear combination) and deployed. Aside from the decision trees, these tasks are not easy for the random forest or the nearest neighbors.

4 Processing with Tanagra

While linear classifiers are produced, R (about the packages used) does not provide the coefficient of the separation line for certain methods, notably concerning naive bayes (section **Erreur ! Source du renvoi introuvable.**) and linear SVM (section 3.6). In this section, we use Tanagra to reproduce the calculations on the same learning sample. The interest is that Tanagra provides the explicit equation when it produces a linear model. We will be able to compare the obtained coefficients.

4.1 Data importation

From R, we export the learning set with the **write.table()** command (text file with tabulation-separated values).

```
write.table(train.data,file="training_set.txt",sep="\t",dec=".",quote=F,row.names=F)
```

After launching Tanagra, we create a new diagram (File/New menu) and we import the data file.

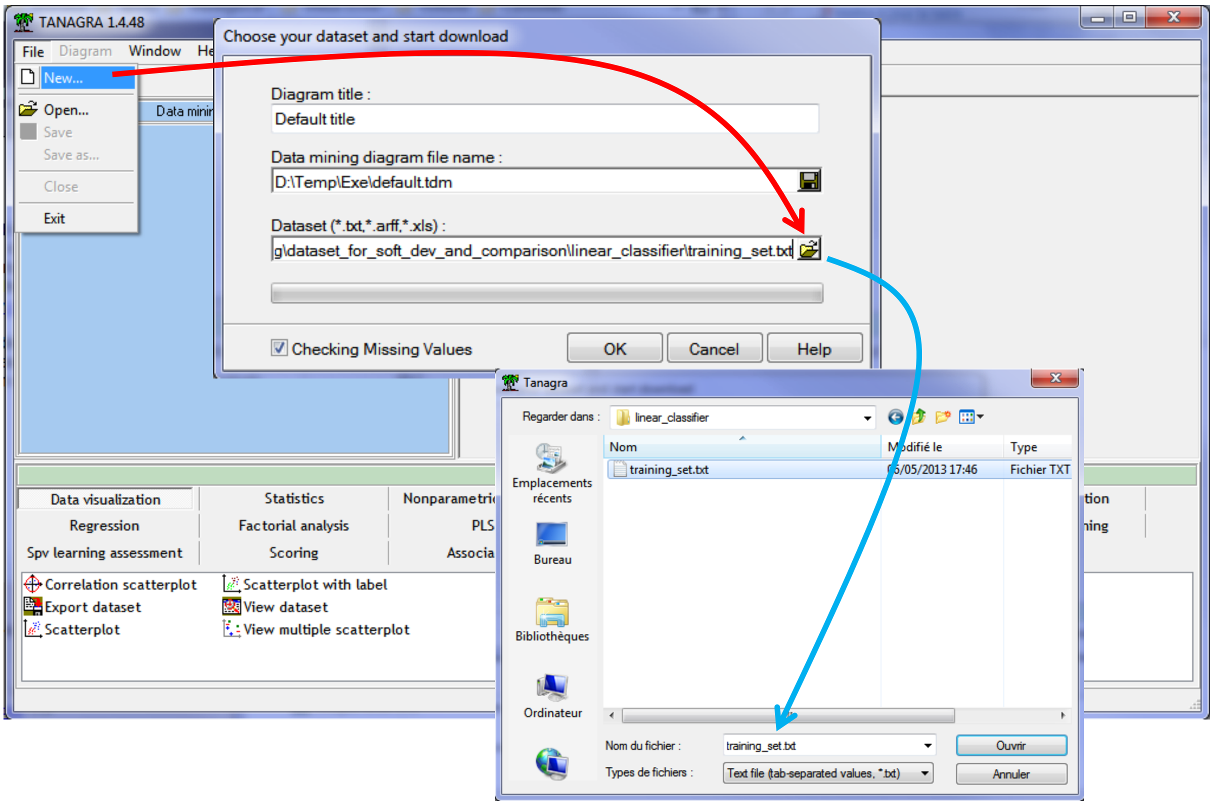
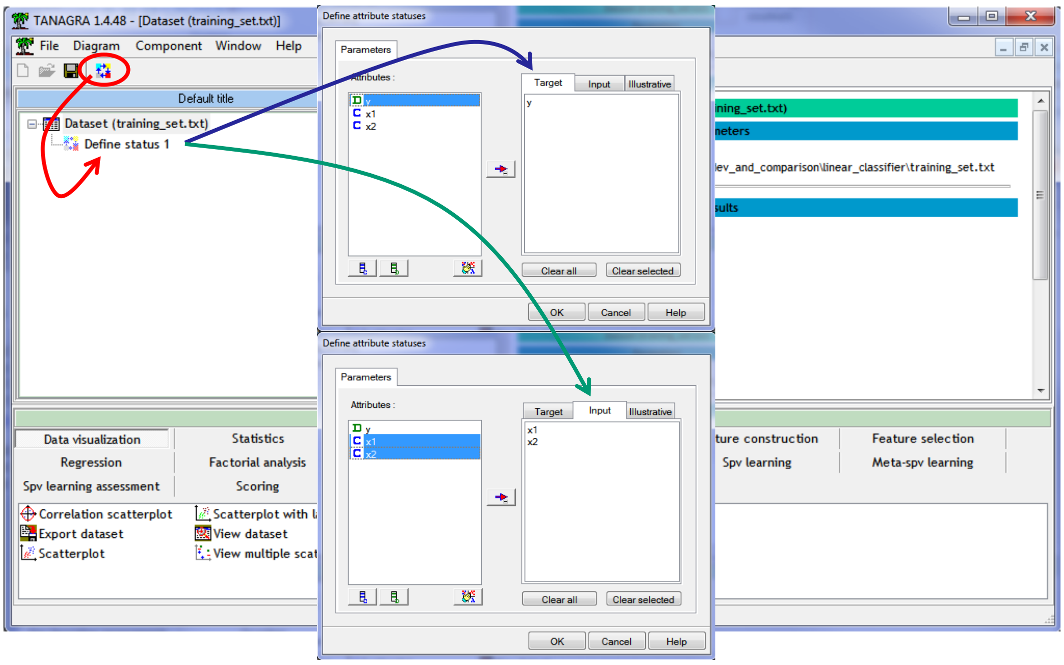


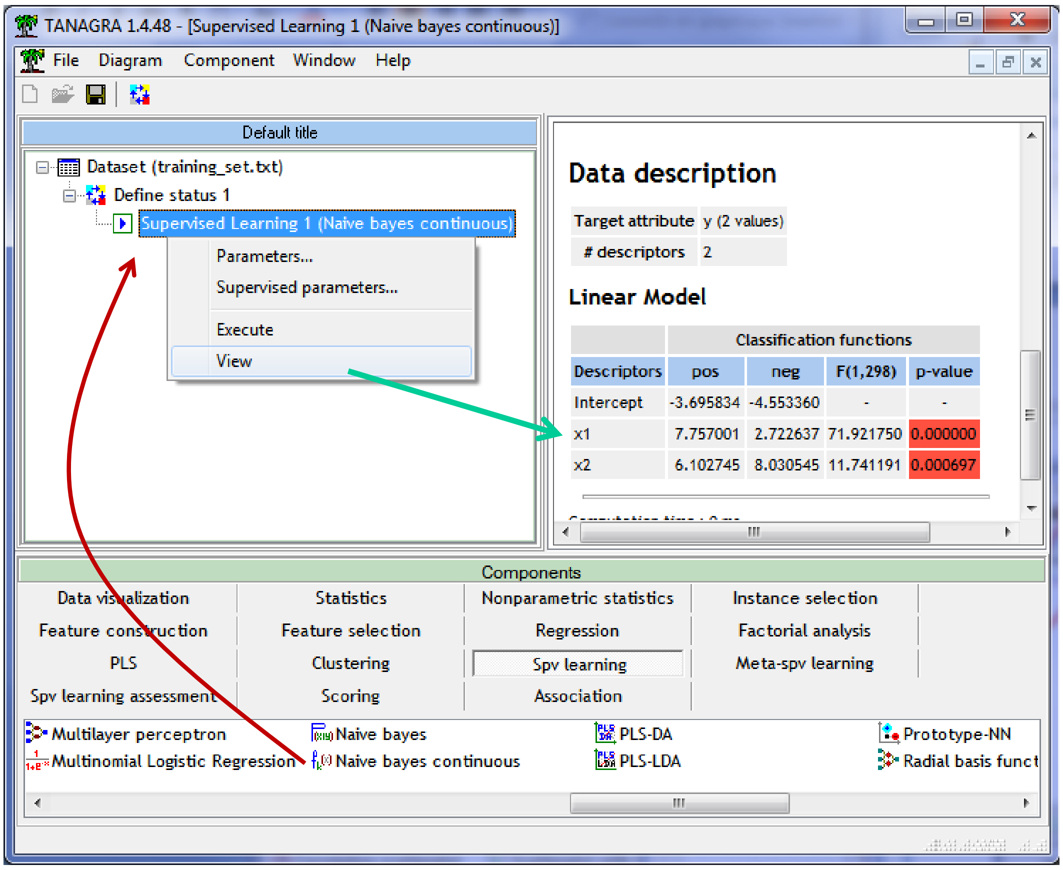
Diagram title : Default title
Data mining diagram file name : D:\Temp\Exe\default.tdm
Dataset (*.txt,*.arff,*.xls) : g\dataset_for_soft_dev_and_comparison\linear_classifier\training_set.txt
Checking Missing Values
Regarder dans : linear_classifier
Nom | Modifié le | Type
training_set.txt | 05/05/2013 17:46 | Fichier TXT
Nom du fichier : training_set.txt
Types de fichiers : Text file (tab-separated values, *.txt)

4.2 Naive bayes

We specify the role of the variables with the DEFINE STATUS component that we add into the diagram. Y is the target attribute, (X1, X2) are the input ones.


TANAGRA 1.4.48 - [Dataset (training_set.txt)]
File Diagram Component Window Help
Default title
Dataset (training_set.txt)
Define status 1
Define attribute statuses
Parameters
Attributes :
Target Input Illustrative
Clear all Clear selected
OK Cancel Help
Data visualization Statistics
Regression Factorial analysis
Spv learning assessment Scoring
Correlation scatterplot
Export dataset
Scatterplot
View dataset
Feature selection
Spv learning
Meta-spv learning

Then, we add the NAIVE BAYES CONTINUOUS tool (SPV LEARNING tab). We click on the VIEW contextual menu.



Data description

Target attribute	y (2 values)
# descriptors	2

Linear Model

	Classification functions			
Descriptors	pos	neg	F(1,298)	p-value
Intercept	-3.695834	-4.553360	-	-
x1	7.757001	2.722637	71.921750	0.000000
x2	6.102745	8.030545	11.741191	0.000697

From the two classification functions, we can deduce the equation which defines the separation line.

	Classification functions			Separating
Descriptors	pos	neg	pos-neg	Line
Intercept	-3.6958	-4.5534	0.8575	**-0.4448**
x1	7.7570	2.7226	5.0344	**-2.6115**
x2	6.1027	8.0305	-1.9278	**1.0000**

Here is the corresponding equation:

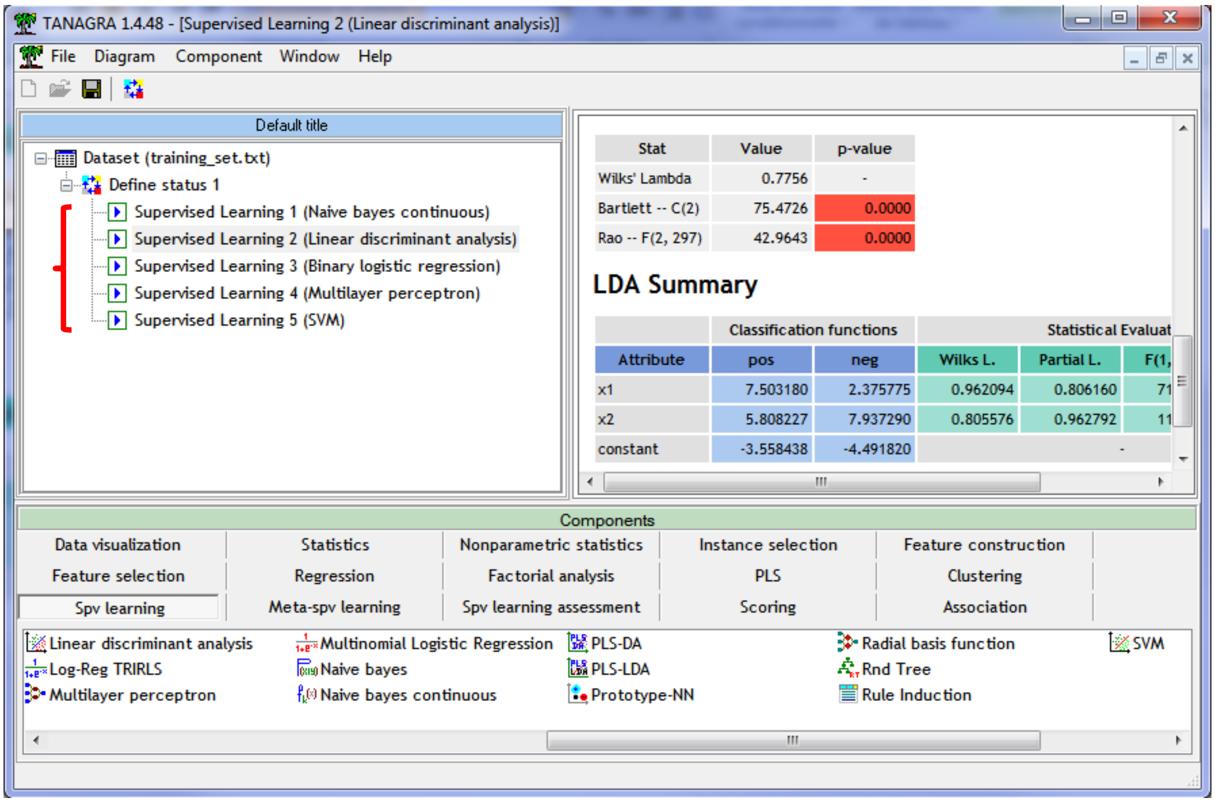
Naïve bayes: X2 – 2.6115 * X1 = 0.4448

Knowing that the true (theoretical) equation of the boundary is:

Theoretical boundary: X2 – 4.0 * X1 = 0

4.3 The other approaches

We have done the same for the other linear approaches. Here is the processing diagram under Tanagra (Figure 15) :


TANAGRA 1.4.48 - [Supervised Learning 2 (Linear discriminant analysis)]
File Diagram Component Window Help
Default title

Dataset (training_set.txt)
Define status 1
Supervised Learning 1 (Naive bayes continuous)
Supervised Learning 2 (Linear discriminant analysis)
Supervised Learning 3 (Binary logistic regression)
Supervised Learning 4 (Multilayer perceptron)
Supervised Learning 5 (SVM)

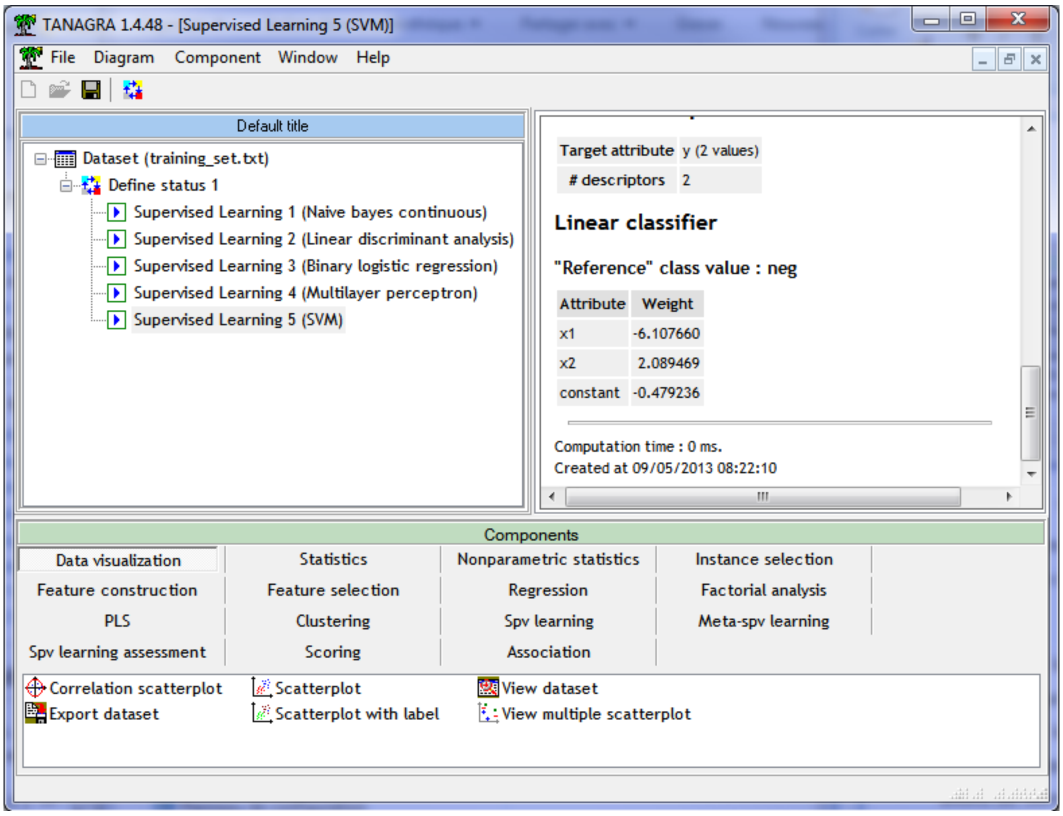
Stat	Value	p-value
Wilks' Lambda	0.7756	-
Bartlett -- C(2)	75.4726	0.0000
Rao -- F(2, 297)	42.9643	0.0000

LDA Summary

	Classification functions		Statistical Evaluat		
Attribute	pos	neg	Wilks L.	Partial L.	F(1,
x1	7.503180	2.375775	0.962094	0.806160	71
x2	5.808227	7.937290	0.805576	0.962792	11
constant	-3.558438	-4.491820			-

Figure 15 – Processing diagram under Tanagra – Results of Linear Discriminant Analysis

All methods, which can detect the influence of the variables, have highlighted the relevance of X1 and X2 (naive bayes, linear discriminant analysis, logistic regression).


TANAGRA 1.4.48 - [Supervised Learning 5 (SVM)]
File Diagram Component Window Help
Default title
Dataset (training_set.txt)
Define status 1
Supervised Learning 1 (Naive bayes continuous)
Supervised Learning 2 (Linear discriminant analysis)
Supervised Learning 3 (Binary logistic regression)
Supervised Learning 4 (Multilayer perceptron)
Supervised Learning 5 (SVM)
Components
Data visualization
Statistics
Nonparametric statistics
Instance selection
Feature construction
Feature selection
Regression
Factorial analysis
PLS
Clustering
Spv learning
Meta-spv learning
Spv learning assessment
Scoring
Association
Correlation scatterplot
Scatterplot
View dataset
Export dataset
Scatterplot with label
View multiple scatterplot

Target attribute	y (2 values)
# descriptors	2

Linear classifier

"Reference" class value : neg

Attribute	Weight
x1	-6.107660
x2	2.089469
constant	-0.479236

Computation time : 0 ms.
Created at 09/05/2013 08:22:10

Figure 16 – Tanagra - Results of the SVM component (Linear kernel)

About SVM (Figure 16), we used our implementation (SVM) because it provides the coefficients of the linear model. The C-SVC component from the library LIBSVM does not do it. We have standardized the variables for SVM.

4.4 Overview – Comparing the coefficients of the separation line

In this section, we analyze the equations provided by the various approaches by comparing them with the theoretical frontier.

Approach	**Equation of the frontier**
Theoretical frontier	**X2 - 4.0000 * X1 = 0.0000**
1 – Naïve Bayes	X2 - **2.6115** * X1 = **0.4448**
2 – Linear Discriminant Analysis	X2 - **2.4083** * X1 = **0.4384**
3 – Logistic Regression	X2 - **2.9525** * X1 = **0.2901**
4 – Perceptron	X2 - **2.9468** * X1 = **0.2024**
5 – Linear SVM (SVM component – Tanagra)	X2 - **2.9231** * X1 = **0.2294**

All boundaries are shifted to the left of the theoretical frontier. This positioning is the consequence of the approach we used to add noise to the labels. But the differences are not the same. We can draw the inferred separation line by the various approaches into a graph. In this way, we can analyze the divergences and the resemblances between the approaches (Figure 17).

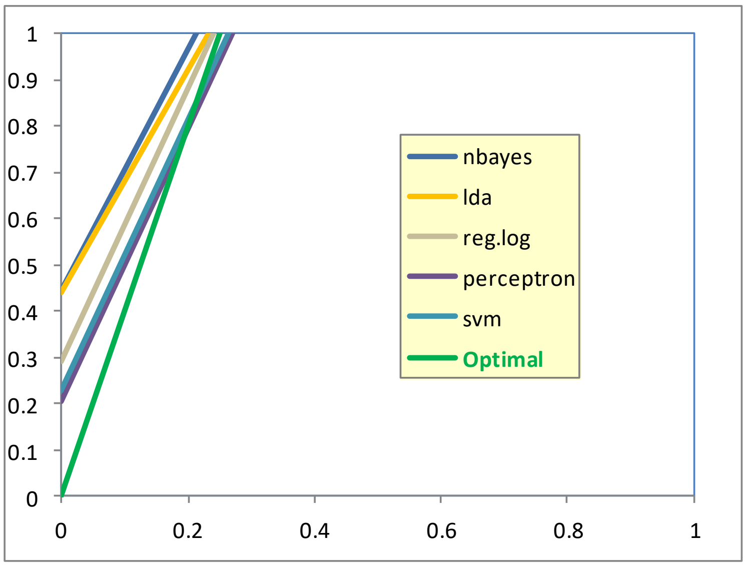


Figure 17 – Separation lines inferred by the various approaches (in green the right separation line)

5 Processing with other data mining tools

Various data mining tools provide linear classifiers. Some of them incorporate variants (e.g. kernel estimation for naive bayes, original display of the results with the "nomograms" for Orange, etc.). In this section, we create a processing diagram for these tools, diagram which is like the one defined under Tanagra. We use Knime, Orange, RapidMiner and Weka.

We summarize below the characteristics of the methods implemented in these data mining tools[2].

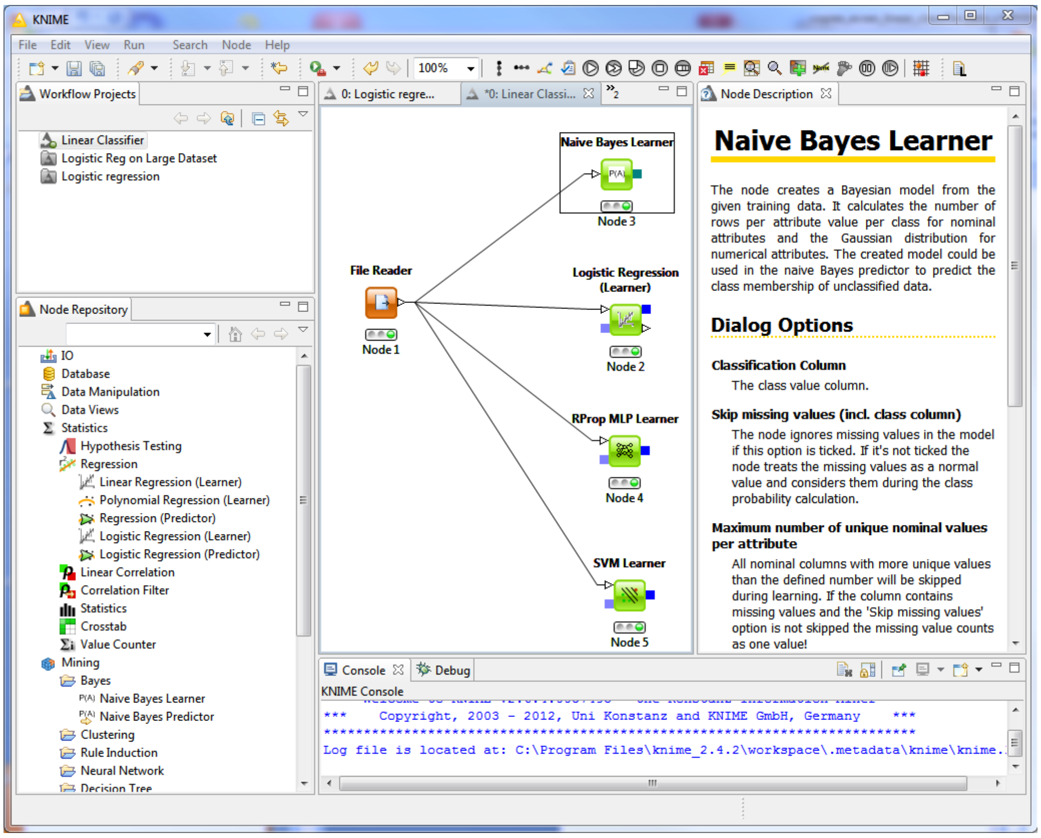
[2] The French version of this tutorial was written in May 2013. The versions of the softwares used correspond to that available at this period.

Approach	Knime 2.6.4 (Figure 18)	Orange 2.6.1 (Figure 19)[3]	RapidMiner 5.2.008 (Figure 20)	Weka 3.7.4 (Figure 21)
Naive bayes	Gaussian assumption. Outputs: conditional average and standard deviation of the descriptors.	Kernel estimation of the conditional probabilities. Description of the influence of the variables using the "nomogram".	Gaussian assumption. Outputs: conditional average and standard deviation of the descriptors.	Kernel estimation of the conditional probabilities or discretization of the variables on-the-fly. Outputs: conditional average and standard of the descriptors.
Linear discriminant analysis	-	-	Display only the distribution of the classes (???).	-
Logistic regression	Display the coefficients of the regression equation, including the tests for significance. X2 – 2.9524 * X1 = 0.2901	Possibility of variable selection (stepwise approaches). Description of the influence of the variables with the "nomogram".	Based on the myKLR[4] implementation and not on the usual "Fisher scoring". Display the coefficients of the equation. X2 – 2.9606 * X1 = 0.2319	Based on the BFGS implementation. Provide the coefficients but not the tests for significance. X2 – 2.9524 * X1 = 0.2901
Perceptron	For a single-layer perceptron, we must set only one neuron into the hidden layer. The standard display includes only the decreasing of the error. The weights are available into the PMML output. X2 – 4.0106 * X1 = 0.0617	-	The single layer is available. The output includes the weights of the linear equation. X2 – 13 * X1 = 0 (???)	0 neuron into the hidden layer to obtain the single-layer perceptron. X2 – 3.9886 * X1 = 0.0219
Linear SVM	Set a polynom with a degree 1 to obtain a linear SVM. Display the supports points for each class.	Based on the LIBSVM library. The supports points are visualized into a table or a graph (limited to a two-dimensional representation space).	Based on LIBSVM. It provides both the support points and the coefficients of the hyperplane for the linear kernel. X2 – 2.5144 * X1 = 0.1315	It provides the coefficients of the hyperplane when we set a linear kernel. X2 – 2.8646 * X1 = 1.3680

Most of the results are consistent. There are still some disparities for some software/methods. The failure of the calculations can be the result of a poorly controlled setup. I tried to set the same parameters from one software to another, when the comparison was possible. I have systematically disabled the normalization/standardization of variables since X1 and X2 are defined on the same scale (0, 1).

[3] Online documentation: http://orange.biolab.si/docs/latest/widgets/rst/

[4] http://www-ai.cs.uni-dortmund.de/SOFTWARE/MYKLR/index.html.en



Naive Bayes Learner

The node creates a Bayesian model from the given training data. It calculates the number of rows per attribute value per class for nominal attributes and the Gaussian distribution for numerical attributes. The created model could be used in the naive Bayes predictor to predict the class membership of unclassified data.

Dialog Options

Classification Column
The class value column.

Skip missing values (incl. class column)
The node ignores missing values in the model if this option is ticked. If it's not ticked the node treats the missing values as a normal value and considers them during the class probability calculation.

Maximum number of unique nominal values per attribute
All nominal columns with more unique values than the defined number will be skipped during learning. If the column contains missing values and the 'Skip missing values' option is not skipped the missing value counts as one value!

```
KNIME Console
***   Copyright, 2003 - 2012, Uni Konstanz and KNIME GmbH, Germany    ***
*************************************************************************
Log file is located at: C:\Program Files\knime_2.4.2\workspace\.metadata\knime\knime.
```

Figure 18 – Linear classifiers under Knime

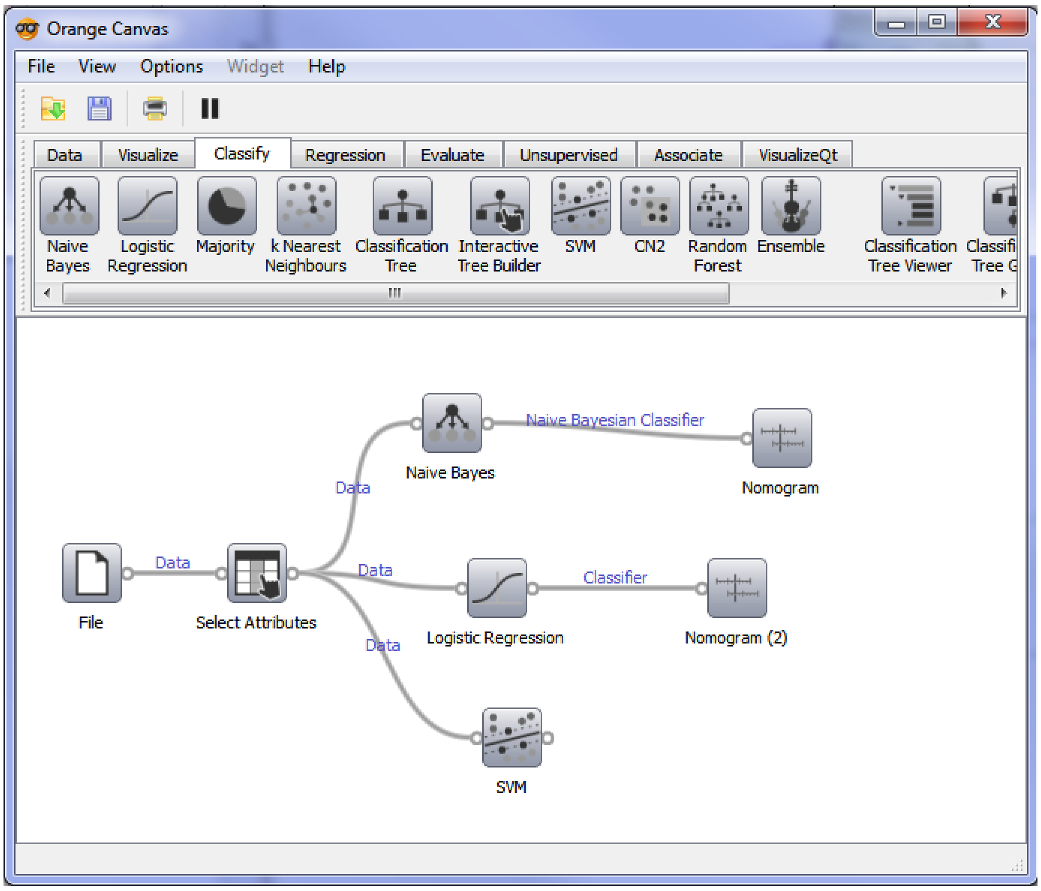
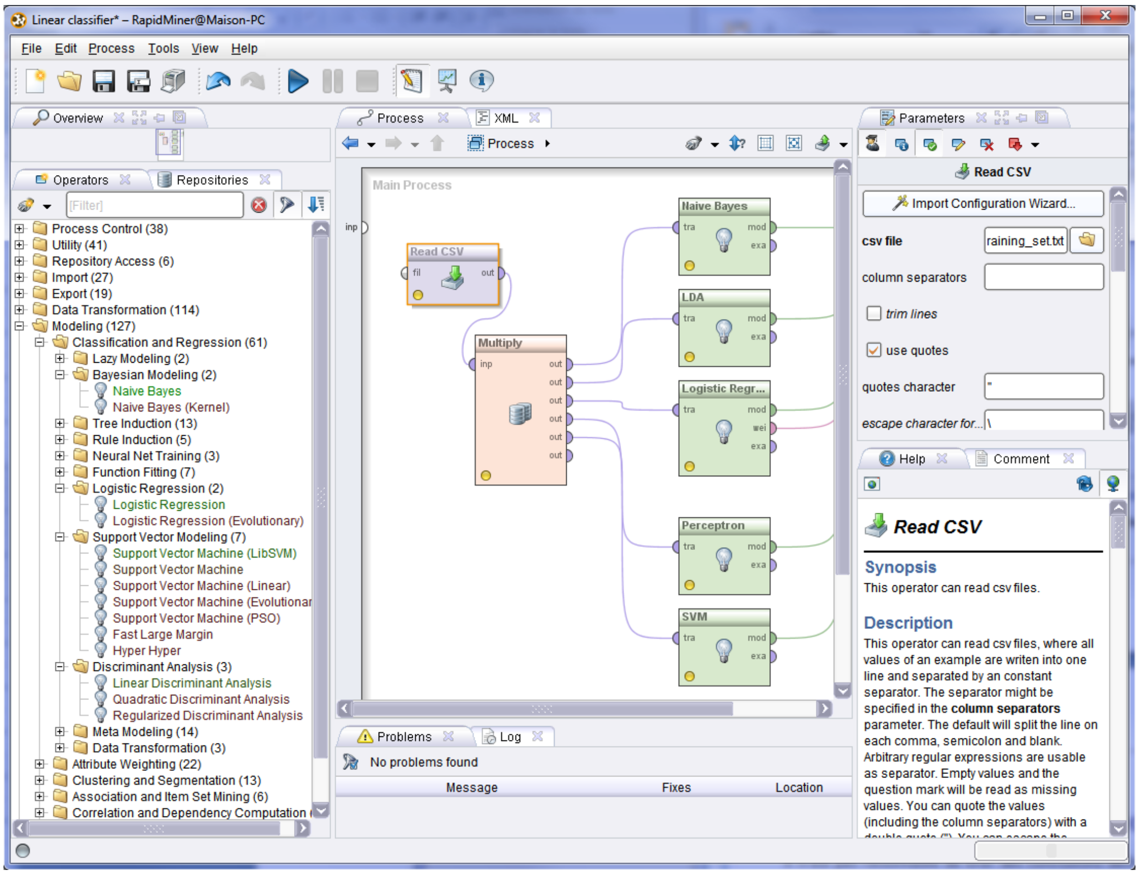


Figure 19 - Linear classifiers under Orange



Read CSV

Import Configuration Wizard...

csv file: raining_set.txt

column separators:

☐ trim lines

☑ use quotes

quotes character: "

escape character for...: \

Read CSV

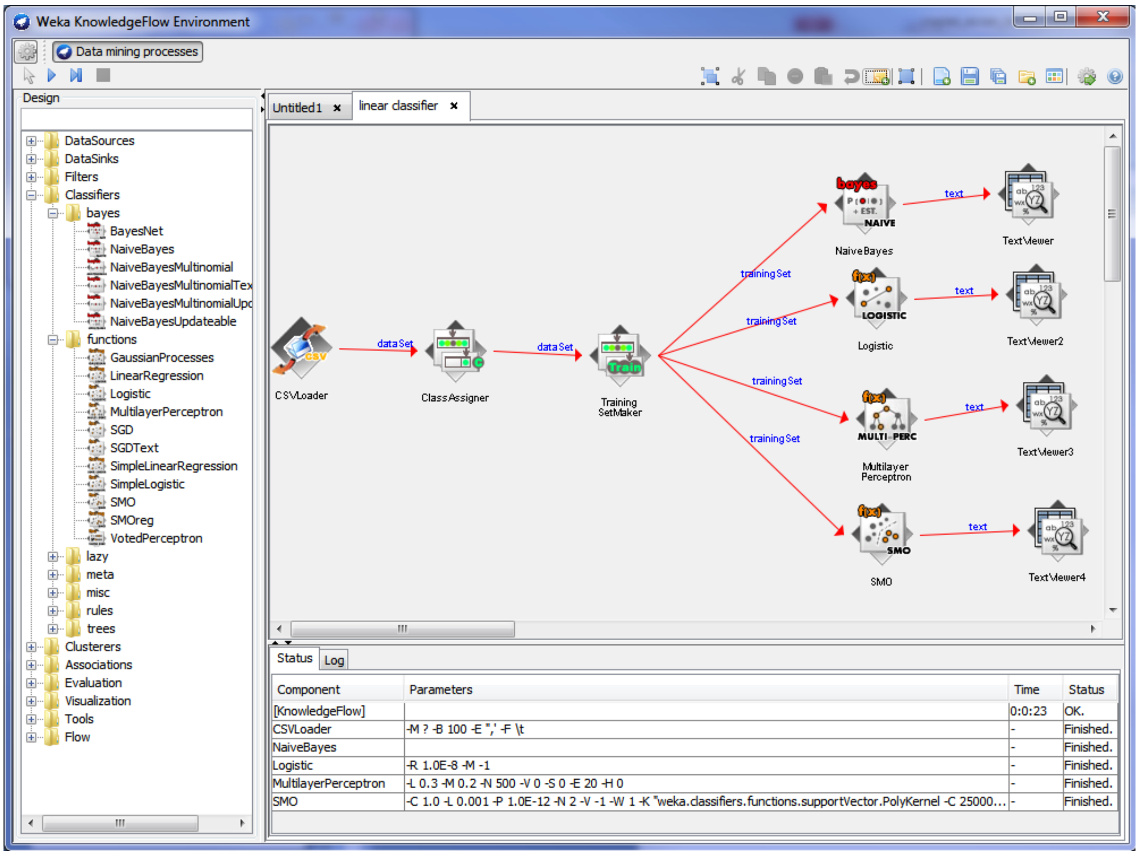
Synopsis

This operator can read csv files.

Description

This operator can read csv files, where all values of an example are writen into one line and separated by an constant separator. The separator might be specified in the **column separators** parameter. The default will split the line on each comma, semicolon and blank. Arbitrary regular expressions are usable as separator. Empty values and the question mark will be read as missing values. You can quote the values (including the column separators) with a

Figure 20 - Linear classifiers under RapidMiner



Component	Parameters	Time	Status
[KnowledgeFlow]		0:0:23	OK.
CSVLoader	-M ? -B 100 -E ",' -F \t	-	Finished.
NaiveBayes		-	Finished.
Logistic	-R 1.0E-8 -M -1	-	Finished.
MultilayerPerceptron	-L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H 0	-	Finished.
SMO	-C 1.0 -L 0.001 -P 1.0E-12 -N 2 -V -1 -W 1 -K "weka.classifiers.functions.supportVector.PolyKernel -C 25000…	-	Finished.

Figure 21 - Linear classifiers under Weka

6 Experiment 1 – Size of the learning sample "n"

In this section, we study the influence of the training set size of the behavior of the various linear approaches. To do this, we increase gradually the training set size and we measure the error rate on the test set.

It is not reasonable to draw definitive conclusions from the experiment conducted on only one dataset. To control results-related variability, we repeat 100 times the experiment for a given learning set size "n.train". On the other hand, we use a large dataset as test set with "n.test" = 100,000 instances. By using the same test sample throughout the experiments, the performances of the models are directly comparable, it is a form of pairing.

6.1 Program for the experiment

We combine various values of **n.train = (250, 500, 1000, 2000, 3000, 5000)** and the 5 linear machine learning methods (naive bayes, linear discriminant analysis, logistic regression, perceptron, linear svm). We repeat K = 100 times the experiment for each combination. We detail the R program below.

```
#function for the calculation of the error rate
#pred is the prediction of a classifier
error.rate <- function(data.test,pred){
  mc <- table(data.test$y,pred)
  err.rate <- 1-sum(diag(mc))/sum(mc)
  return(err.rate)
}

#one experiment for a learning sample of size "n"
#the function returns the test error rate for each classifier
experiments <- function(n,test.set){

  #generation of a learning sample (size n)
  learning <- generate.data(n,p,noise)
  print(nrow(learning))

  #preparation of the vector gathering the results
  #5 learning algorithms to evaluate
  result <- numeric(5)

  #naive bayes classifier
  model.nb <- naiveBayes(y ~ ., data = learning)
  result[1] <- error.rate(test.set,prediction.nb(model.nb,test.set))

  #linear discriminant analysis
  model.lda <- lda(y ~ ., data = learning)
  result[2] <- error.rate(test.set,prediction.lda(model.lda,test.set))

  #logistic regression
  model.glm <- glm("y ~ .", data = learning, family = binomial)
  result[3] <- error.rate(test.set,prediction.glm(model.glm,test.set))

  #single layer perceptron
  model.nn <- nnet(y ~ ., data = learning,skip=TRUE,size=0)
  result[4] <- error.rate(test.set,prediction.nn(model.nn,test.set))

  #linear support vector machine
  model.svm <- svm(y ~ ., data = learning,kernel="linear")
  result[5] <- error.rate(test.set,prediction.svm(model.svm,test.set))

  print(result)
  return(result)
}

#various learning set size to evaluate
size.training <- c(250,500,1000,2000,3000,5000)

#generation of the unique test set
#used during the whole experiment
set.seed(25032003)
other.data.test <- generate.data(100000,p,noise)

#load the package needed for the learning algorithms
library(MASS)
library(e1071)
library(nnet)

#experiment for a given learning set size: “size.learning”
one.expe.session <- function(size.learning){
 results <- mapply(experiments,size.learning,MoreArgs=list(test.set=other.data.test))
  return(results)
}

#K: number of repetition for each experiment
K <- 100
set.seed(05092008)
all.results <- replicate(K,one.expe.session(size.learning=size.training),simplify="matrix")

#all.results is a table with K = 100 columns
#and 30 rows (5 learning methods x 6 learning set sizes)

#preparing the results for a new table:
```

```
#in rows, 5 methods; in columns, 6 learning set sizes
#the summary measure is the mean of the K = 100 trials
mean.results <- matrix(0,nrow=5,ncol=length(size.training))
colnames(mean.results) <- size.training
rownames(mean.results) <- c("naive.bayes","lda","log.reg","perceptron","svm.linear")
for (i in 1:5){
  for (j in 1:length(size.training)){
    mean.results[i,j] <- mean(all.results[i+(j-1)*5,])
  }
}
print(mean.results)


#the same calculations but using the median as summary measure
med.results <- mean.results
for (i in 1:5){
  for (j in 1:length(size.training)){
    med.results[i,j] <- median(all.results[i+(j-1)*5,])
  }
}
print(med.results)
```

The calculations are slow. We really need to do several tests to calibrate the experiment, to make sure that the results are generated correctly, and that they are collected adequately.

6.2 Discussion of the results

Here are the curves of the error according to the learning sample size.

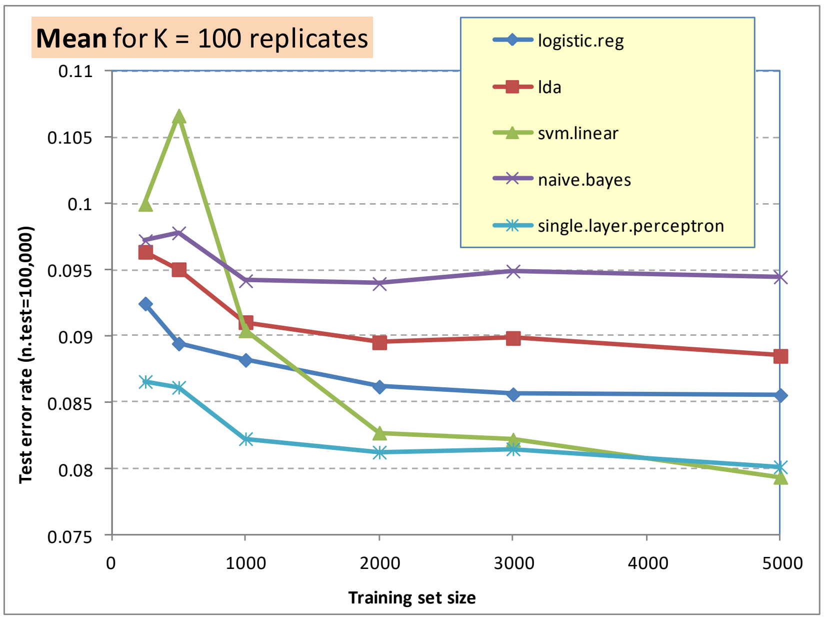


Figure 22 – Mean of the error rate according to learning sample size

At a first glance, if we use the mean as summary indicator (Figure 22):

1. Clearly, the methods based on too restrictive assumptions are not relevant (naive bayes, linear discriminant analysis and, in a lesser extent, the logistic regression).
2. SVM and Perceptron are therefore the best on our data, quite similarly.

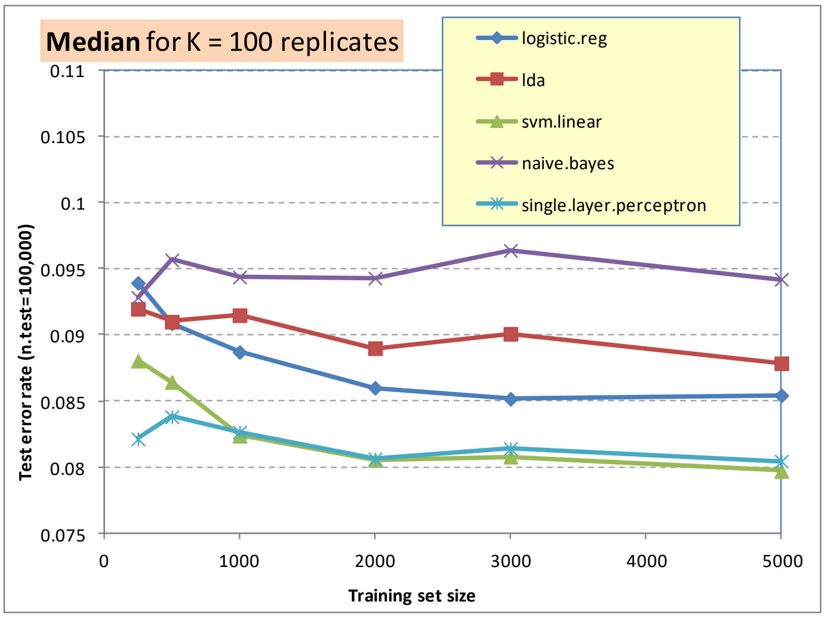


Figure 23 - Median of the error rate according to the learning sample size

But some additional conclusions are also of interest:

3. Linear SVM seems to have a catastrophic behavior on small learning samples (n.train ≤ 500) (Figure 22). In fact, the computational library has failed on a large proportion of these samples. The curve is quite different – for SVM – when we use the median as summary measure (Figure 23). This kind of problems sometimes happens in experiments. It is necessary to check, re-check, and still check the results before thinking about publishing them.
4. The test error rate decreases when the learning sample size increases. Fortunately. The contrary would have been counterintuitive. But, from “n.train = 2000”, the improvement is insignificant. I think, it is due to the simplicity of the concept to learn (linear frontier with only two descriptors).
5. However, none of the methods converge on optimal performance (5%), even with a large learning sample. This is the consequence of the noise added to the labels**. When we generate data without noise, all methods, except naive bayes and linear discriminant analysis which are constrained by their restrictive assumptions, find the theoretical frontier**.

7 Experiment 2 – Number of descriptors “p”

In this section, we set “n.train = 500”, and we check the influence of the number of descriptors on the quality of the inferred classifiers. We know that only (X1, X2) are relevant. The additional descriptors are thus irrelevant. They may be considered as another kind of noise added to the data.

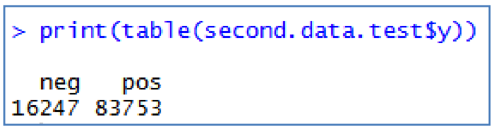
7.1 Program of the experiment

We combine the values of "p" [**p = (2, 5, 10, 25, 50, 70)** – knowing that (p-2) are irrelevant] and the 5 learning algorithms. We repeat each combination K = 100 times. We detail below the program for R. It has strong similarities to the source code in the previous section.

```
#Experiment: influence of the number of descriptors p
#learning sample size = 500
experiments.dimension <- function(p,test.set){

  #learning set
  learning <- generate.data(500,p,noise)

  #vector containing the results
  result <- numeric(5)

  #naive bayes classifier
  model.nb <- naiveBayes(y ~ ., data = learning)
  result[1] <- error.rate(test.set,prediction.nb(model.nb,test.set))

  #linear discriminant analysis
  model.lda <- lda(y ~ ., data = learning)
  result[2] <- error.rate(test.set,prediction.lda(model.lda,test.set))

  #logistic regression
  model.glm <- glm("y ~ .", data = learning, family = binomial)
  result[3] <- error.rate(test.set,prediction.glm(model.glm,test.set))

  #single layer perceptron
  model.nn <- nnet(y ~ ., data = learning,skip=TRUE,size=0)
  result[4] <- error.rate(test.set,prediction.nn(model.nn,test.set))

  #linear support vector machine
  model.svm <- svm(y ~ ., data = learning,kernel="linear")
  result[5] <- error.rate(test.set,prediction.svm(model.svm,test.set))

  print(result)
  return(result)
}

#generate the test samples with
#n.test = 100000 instances and p = 100 descriptors
set.seed(25032003)
second.data.test <- generate.data(100000,100,noise)
```

```
#print
print(colnames(second.data.test))
print(table(second.data.test$y))

#various dimension size
size.p <- c(2,5,10,25,50,70)

#one experiment for various dimensionality
one.expe.dimension <- function(size.dimension){
  results <- mapply(experiments.dimension,size.dimension,MoreArgs=list(test.set=second.data.test))
  return(results)
}

#repeat K times the experiments
K <- 100
set.seed(21102011)
all.results <- replicate(K,one.expe.dimension(size.dimension=size.p),simplify="matrix")

#summary measure: mean
mean.results <- matrix(0,nrow=5,ncol=length(size.p))
colnames(mean.results) <- size.p
rownames(mean.results) <- c("naive.bayes","lda","log.reg","perceptron","svm.linear")

for (i in 1:5){
  for (j in 1:length(size.p)){
    mean.results[i,j] <- mean(all.results[i+(j-1)*5,])
  }
}
print(mean.results)

#summary measure: median
med.results <- mean.results
for (i in 1:5){
  for (j in 1:length(size.p)){
    med.results[i,j] <- median(all.results[i+(j-1)*5,])
  }
}
print(med.results)
```

The test sample “second.data.test” is generated with 100,000 instances and 100 descriptors. It is operable for the various values of descriptors we try p = (2, 5, 10, 25, 50, 70).



```
> print(table(second.data.test$y))

  neg   pos 
16247 83753 
```

The test sample has 16.25% of negative instances, and 83.75% of positive ones. That is an important information. It means that the **error rate of the default classifier** (predicting systematically the most frequent class) is **16.25%**. We will see that some classifiers do not fare better when we increase the number of irrelevant descriptors.

7.2 Discussion of the results

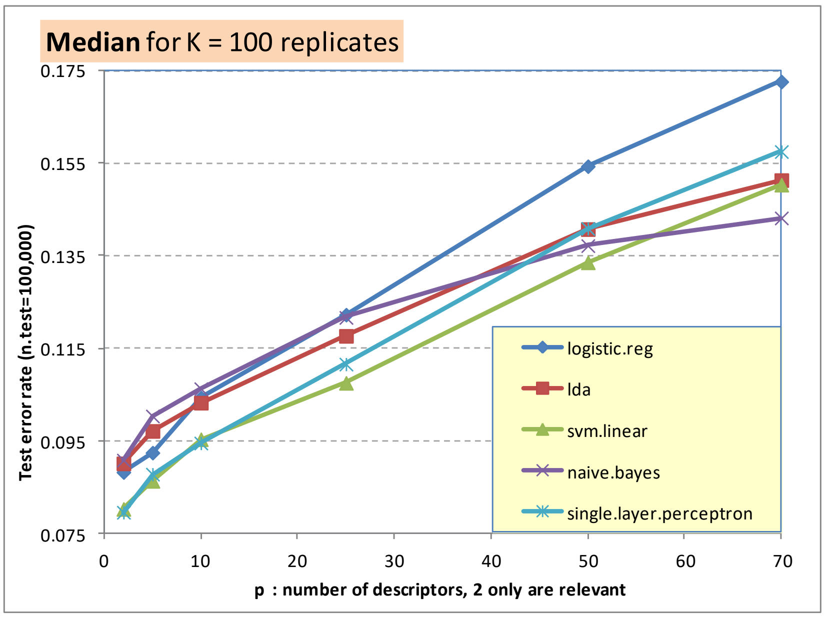


Figure 24 - Median of the error rate according to the number of descriptors

We present the median of error rates for K = 100 trials (Figure 24). We can make several comments:

1. The curse of dimensionality is not a myth. All the approaches fails when we add in a high proportion the irrelevant variables.
2. But not in the same way however. For instance, the Perceptron and SVM, which are the best when we use only the two relevant descriptors, evolve differently. Clearly, SVM resists better to the addition of noisy variables than the perceptron. This last one is close to the default classifier in the worst case.
3. The good surprise is that the discriminant analysis and the naive bayes, previously dominated, are rather robust towards the dimensionality (knowing that we are far from the optimal error anyway). But is this really a surprise? A restrictive search bias becomes beneficial when we present erratic datasets to the learning process.
4. The Naive Bayes stands out because the number of parameters to be estimated is very low (conditional mean and standard deviations of descriptors simply). It even surpasses the SVM when the representation space is extremely noisy (for our dataset).
5. The logistic regression is really disturbed when the dimension increases. It is not better than the default classifier when "p = 70" (68 descriptors are irrelevant).

All this shows above all that the variable selection is an essential aspect of supervised learning, both for the interpretation of models and for their predictive qualities, including on an easy to

learn concept that we used to generate the data in this tutorial (linear separator in a two-dimensional representation space).

8 Conclusion

Our initial goal was to show and compare the behavior of the most popular linear classifiers. We have first detailed the working of the methods by describing the boundaries induced on an artificial data set (sections 3 et 4). There is no doubt that linear methods subdivide the area of representation into regions by using straight lines (or hyperplane if we are in higher than 2-dimensions representation spaces). We have also seen that some techniques, because of their underlying assumptions, are struggling to infer the right solutions when they are placed in situations that disadvantage them.

In a second step, to give better viability to the results, we expand experiments, by trying to analyze the impacts of the learning sample size and dimensionality on the quality of the results. The value of using artificial data is that we fully control the evaluation process. We know the characteristics of the data generated which can explain the nature of the obtained results. Among our main results, we observe that some methods are more robust than the others when they are placed in a difficult context.

Finally, as a prospective, we could explore the influence of the level and the kind of noise on the behavior of the learning algorithms. To achieve this, only few changes are needed to adapt the program accompanying this tutorial.

9 References

Bardos M., « Analyse Discriminante – Application au risque et scoring financier », Dunod, 2001; chapter 2, « Fisher Discriminant Analysis », pp. 29 à 59; chapter 3, « Logistic Discrimination », pp. 61 à 79 [in French].

Bishop C., « Pattern Recognition and Machine Learning », Springer, 2006; Chapter 4, « Liner Models for Classification », pp. 179 à 224.

Duda R., Hart P., Stork D., « Pattern Classification », John Wiley and Sons, 2001; chapter 5, « Linear Discriminant Functions », pp. 215 à 281.

Hastie T., Tibshirani R., Friedman J., « Elements of Statistical Learning », 10th printing, Janvier 2013, http://www-stat.stanford.edu/~tibs/ElemStatLearn/ ; chapter 4, « Linear Methods for Classification », pp. 101 à 137.

Theodoridis S., Koutroumbas K., « Pattern Recognition », Elsevier Inc., 2009; chapter 3, « Linear Classifiers », pp. 91 à 150.