Ensembles of classifiers based on Kernel density estimators

Edgar Acuna*
Alex Rojas

Department of Mathematics
University of Puerto Rico,
Mayaguez Campus

April 26, 2000

*This research was supported by grant N00014-00-1-0360 from the Office of Naval Research

Abstract

People in statistics, computer science and engineering are conducting a lot of research on combining classification rules (classifiers) to produce a single one, known as an *ensemble*, which in general is more accurate than the individual classifiers making up the *ensemble*. Three popular methods for creating *ensembles* are *Bagging* (Bootstrap Aggregating), Ada*Boosting (*Adaptive Boosting*)* and *Arcing* (Adaptively resample and combine). These methods rely on resampling techniques to obtain different training sets for each of the classifiers. *Stacking* is another method to combine classifiers but it does not use resampling. Previous work has demonstrated that combining techniques are very effective for unstable classifiers. In this paper will present some results in application of combining techniques to classifiers where the class conditional density is estimated using Kernel density estimators. These type of classifiers are unstable due to singularities presented in the log-likelihood function. Also, we will explore the mixed kernel estimators and adaptive kernel density estimators.

1. Introduction

In the classification problem with J classes and M predictors, one has a learning (training) set of data $\mathcal{L}$: $\{(y_k,x_k)$, k=1,….N$\}$ consisting of measurements on N cases, where each case consists of a categorical response variable y_k that takes values in $\{1,….,J\}$ and a prediction or feature vector $x_k=(x_{1k}, ….x_{Mk})$. The goal of classification is to use $\mathcal{L}$ to construct a function C(x, $\mathcal{L}$), defined for x such that C(. , $\mathcal{L}$) gives accurate classification of future data. More specifically suppose that we have a large test set $\mathcal{T}=\{(y'_k,x'_k)$ k=1, 2,… N'$\}$ sampled from the same distribution as $\mathcal{L}$, then we want C(x'_k,$\mathcal{L}$) to be an accurate estimate of y'_k for all (y'_k,x'_k) in $\mathcal{T}$.

Type of classifiers (according to Statlog)

1) Linear Discriminant and its extensions: Linear Discriminant, Logistic discriminant, Quadratic discriminant, Multilayer Perceptron (Neural network), Projection Pursuit.

2) Decision Trees and rule-based methods : NewID, AC^2, Cal5, CN2, C4.5, CART, IndCART, Bayes Rule, Itrule

3) Density estimates: k-NN, kernel density estimates, Naïve Bayes, Radial basis function (Neural Network), Polytrees, Kohonen self-organizing net (neural network), LVQ(Neural Network)

4) Support Vector Machines

2. The misclassification error and the Bias-Variance tradeoff.

In the classification problem one has a learning (training) set of data $\mathcal{L}$ consisting of measurements on N cases, where each case consists of a categorical response variable y_k that takes values in $\{1,….,K\}$ and a prediction or feature vector $\mathbf{x}_k=(x_{1k}, ….x_{Mk})$. That is, the learning set consists of data $\{(y_k,x_k)$, k=1,….N$\}$. The goal of classification is to use $\mathcal{L}$ to construct a function C($\mathbf{x}$, $\mathcal{L}$), defined for $\mathbf{x}$ such that C(. , $\mathcal{L}$) gives accurate classification of future data. More specifically suppose that we have a large test set $\mathcal{T}=\{(y'_k,\mathbf{x}'_k)$ k=1, 2,… N'$\}$ sampled from the same distribution as $\mathcal{L}$. Then we want C($\mathbf{x}'_k$,$\mathcal{L}$) to be an accurate estimate of y'_k for all (y'_k,$\mathbf{x}'_k$) in $\mathcal{T}$.

Let each (y,$\mathbf{x}$) case in $\mathcal{L}$ be independently drawn from a distribution F. Take Y,$\mathbf{X}$ to be random variables having the distribution F and independently of $\mathcal{L}$. Then the misclassification error of the classifier C is defined as

$$ME(C)=E_{\mathcal{L}}[P_{\mathbf{X},\mathbf{Y}}(C(\mathbf{X},\mathcal{L})\neq Y)]$$

which is minimum when C is chosen as the Bayes classifier C*, where $C^*(x)=\text{argmax}_j P(Y=j/\mathbf{X}=\mathbf{x})$

Let U be the set of all **x** at which C is unbiased and let B be the complement of U. The bias of a classifier C is

$$\text{Bias}(C)=P_{\mathbf{X},Y}(C^*(\mathbf{X})=Y,\ \mathbf{X}\in B) - E_L(P_{\mathbf{X},Y}(C(X,\mathcal{L})=Y,\ \mathbf{X}\in B)$$

An its variance is

$$\text{Var}(C)=P_{\mathbf{X},Y}(C^*(\mathbf{X})=Y,\ \mathbf{X}\in U) - E_L(P_{\mathbf{X},Y}(C(X,\mathcal{L})=Y,\ \mathbf{X}\in U)$$

This leads to the fundamental decomposition

$$\text{ME}(C)=\text{ME}(C^*)+\text{Bias}(C)+\text{Var}(C)$$

The above definitions are due to Breiman (1998). Other definitions of bias and variance of a classifier are given by Kohavi and Wolpert (1996) and Tibshirani (1996). Geman, Bienenstock and Doursat (1992) introduced the bias plus variance decomposition in a regression context.
Using the bootstrap method (Efron and Tibshirani, 1993) we can derive a sample based estimate of the misclassification error fundamental decomposition. The classifier may either overfit the data and in this case the bias is low but the variance is high or underfit the data in this case the bias is high but the variance is low.

Define the aggregated classifier as

$$C_A(\mathbf{x})=\text{argmax}_j P_{\mathcal{L}}[C(\mathbf{x},\mathcal{L})=j]$$

This is aggregation by voting. Consider many independent replicas $\mathcal{L}_1$, $\mathcal{L}_2$,…. ; construct the classifiers $C(\mathbf{x},\mathcal{L}_1)$, $C(\mathbf{x},\mathcal{L}_2)$,…. ; and at each **x** determine the classification $C_A(\mathbf{x})$ by having these multiple classifiers vote for the most popular class.
The classifier $C(\mathbf{x},\mathcal{L})$ is unbiased at **x** if $C_A(\mathbf{x})=C^*(\mathbf{x})$, this means that over the replications of $\mathcal{L}$, $C(x,\mathcal{L})$ picks the right class more often than any other class.
Note that aggregating a classifier and replacing C with C_A reduces the variance to zero, but there is not guarantee that it will reduce the bias

Breiman (1996) heuristically defines a classifier as unstable if a small change in the data $\mathcal{L}$ can make large changes in the classification. Unstable classifiers have low bias but high variance, meanwhile the opposite occurs for stable classifiers. CART and neural networks are not stable classifiers, linear discriminant analysis and K-nearest neighbor classifiers are stable. Several strategies have been proposed for choosing a suitable compromise between bias and variance (Hand, 1997), one of which is to combine multiple classifiers.

3. Combining (Bundling) classifiers

Many researches have investigated the technique of combining the predictions of multiple classifiers to produce a single classifier: Wolpert (1992), Breiman (1996, 1998), Quinlan (1996), Freund and Schapire (1996), Maclin and Optiz (1997), Bauer and Kohavi (1999), etc. The resulting classifier, also known as an Ensemble, is generally more accurate than the individual classifiers making up the ensemble.

Three popular methods for creating ensembles are: Bagging (Bootstrap aggregating) introduced by Breiman, (1996), AdaBoosting by Freund and Schapire (1996) and Arcing (Adaptively resampling and combining) by Breiman (1998). These methods rely on resampling techniques to obtain different training sets for each of the classifiers.

Stacking (Wolpert, 1992) is another method to combine classifers but it does not use resampling, in this case the learning data set is divided in v parts similar to v-fold cross validation and then a classifier is estimated in each of the part. The stacked classifier is a linear combination of the single classifiers.

The Bagging Algorithm: Breiman (1996)

Input: learning set L , classifier C, integer T (number of bootstrap samples)

1. For i=1 to T {
2. B_i= Bootstrap sample from L (i.i.d. sample with replacement)
3. $C_i=C(B_i)$
4. }
5. $C_A(x)= \arg\max_{j\in\{1,....J\}} \sum_{i:C_i(x)=j} 1$ (The class most voted)

Output: Ensemble C_A

Table 1. Results of previous experiments in Bagging

Reference	Classifier	Relative Error Reduction (%)	# of datasets (w-l-t)
Breiman (1996)	CART	29.0	7 (7-0)
Freund & Schapire (1996)	C4.5	20.0	27(22-2-3)
Quinlan (1996)	C4.5	10.0	27(24-3)
Maclin & Opitz (1997)	C4.5	18.5	23(21-1-1)
Maclin & Opitz (1997)	Neural Nets	13.3	23(22-0-2)
Breiman (1998)	CART	36.0	11(11-0)
Bauer & Kohavi (1999)	MC4	14.5	14(14-0)

The Adaboosting Algorithm: Freund and Schapire[1996]

Input: learning set L of size N, classifier C, integer T (number of bootstrap samples)

1. B= L with weights $w_1(x_j)=1/N$. For j=1,…..N

2. For i=1 to T {

3. $C_i=C(B)$

4. $e_i = \sum_{x_j \in B: C_i(x_j) \neq y_j} w_i(\mathbf{x}_j)$ (This is the weighted error on learning set).

5. If either $e_i>1/2$ or $e_i=0$ then restart assigning equal weights (Breiman's variation).

6. Set $b_i = e_i /(1-e_i)$

7. Udpate the weights: for each $x_j \in B$, if $C_i(x_j)=y_j$ $w_{i+1}(x_j)=w_i(x_j)\beta_i$ and $w_{i+1}(x_j)=w_i(x_j)$ otherwise.

8. Normalize the weights.}

9. $C^*(x)= \arg\max_{j\in\{1,....J\}} \sum_{i:C_i(x)=j} \log\frac{1}{b_i}$

Output: Ensemble C*

Table 2. Results of previous experiments with Adaboosting

Reference	Classifier	Relative Error Reduction (%)	# of datasets (w-l-t)
Freund & Schapire (1996)	C4.5	24.8	27(22-4-1)
Quinlan (1996)	C4.5	15.0	27(21-6)
Maclin & Opitz (1997)	C4.5	22.0	23(21-2)
Maclin & Opitz (1997)	Neural Nets	17.1	23(19-4)
Breiman (1998)	CART	48.4	11(10-1)
Bauer & Kohavi (1999)	MC4	27.0	14(10-4)

The Arcing (Arc-x4) Algorithm: Breiman [1998]

Input: Learning set L of size N, classifier C, integer T (number of bootstrap samples)

1. B= L with weights $w_1(x_j)=1/N$. For j=1,…..N

2. For i=1 to T {

3. B_i= Bootstrap sample from L with weight for x set to

$w(x_j)=(1+e(x_j)^4)/\Sigma(1+e(x_j)^4)$, where $e(x_j)$ is the number of misclassifications made on x_j by classifiers C_1 to C_{i-1}.

4. $C_i=C(B_i)$

5. }

6. $C_A(x)=\underset{j\in\{1,.....J\}}{\arg\max}\sum_{i:C_i(x)=j}1$ (The class most voted)

Output: Ensemble C_A

Table 3. Results of previous experiments with Arcing (Arc-x4)

Reference	Classifier	Relative Error Reduction (%)	# of datasets (w-l-t)
Maclin & Opitz (1997)	Neural Nets	19.4	23(20-0-3)
Breiman (1998)	CART	45.4	11(11-0)
Bauer & Kohavi (1999)	MC4	27.0	14(10-4)

Comparison of Bagging, Boosting and Arcing classifiers

The three methods are very effective for unstable classifiers such as decision trees: Breiman (B96, B98), Quinlan (Q96), Freund and Schapire (FS96), Bauer and Kohavi (BK99) and neural networks: Maclin and Optiz (MacO97).

Adaboosting and Arcing apply to decision trees and Naïve-Bayes performs generally better that Bagging, but not uniformly better, sometimes they degraded compared to the single classifier. The same conclusions were obtained for neural networks classifiers (MacO97).

When tree classifiers are used, Bagging mainly reduces the variance, whereas Arcing and boosting reduces, both the bias and the variance (B98 and BK99).

Bagging can be easily be implemented in parallel, but Adaboosting and Arcing are essentially sequential and parallelized versions have not been implemented so far (BK99).

Table 4. Comparison of results of experiments on Bagging, Boosting and Arcing with the best single classifier in several datasets.

Dataset	Cases	Attributes	Classes	Q96 C4.5	FS96 C4.5	Mac097 C4.5	B96 CART	B98 CART	Mac0N
Iris	150	4c	3	5.13b	5.0ab	4.6b	N	N	2.9x
Labor	57	8c, 8d	2	13.86a	11.3b	13.2a	N	N	3.2x
Glass	214	9c	6	23.55a	22.7a	28.4b	24.9b	21.6x	31.5x
Hepatitis	155	6c, 13d	2	17.68a	16.3a	13.8a	N	N	18.1b
Heart-c	303	8c, 5d	2	21.39a	20.9b	17.4a	N	N	16.7b
Diabetes[1]	768	8c	2	23.63b	24.4b	21.9b	18.8b	23.9b	22.8a
Breast-w	699	9c	2	4.09a	3.2b	3.1a	4.2b	3.2a	3.2x
Credit-a[1]	690	6c, 8d	2	14.13b	13.6b	12.1b	N	N	14.1b
Ionosphere	351	34c	2	N	5.8a	6.0ab	8.6b	6.3x	7.0x
Segment[1]	2310	19c	7	1.87a	1.4a	2.3a	N	N	3.7x
Credit-g[1]	1000	7c, 13d	2	25.81b	24.6b	22.8b	N	N	24.3b

1: Statlog Project
b: bagging
a: Ada Boosting
x: Arcing

4. Classification based on kernel density estimation

From a Bayesian point of view, supervised classification is equivalent to compare estimates of the probabilities of belonging to each class with each other, assigning an object with measurement vector **x** to the class with the largest $\hat{f}$ (j/**x**), j=1,2….J. In order to obtain such estimates, one can estimate them indirectly via the class conditional density f(**x**/j) using the Bayes' theorem. Kernel density estimators can be used to carry out that task. For a given class j and a random sample $\mathbf{X}_1,\mathbf{X}_2,\ldots..\mathbf{X}_n$ of the p-dimensional random vector **X** with continuos components, the product kernel of the class conditional density at the point **x** is given by

$$\hat{f}(\mathbf{x}) = \frac{1}{nh_1h_2....h_p}\sum_{i=1}^{n}\prod_{v=1}^{p} K(\frac{x_v - \mathbf{X}_{iv}}{h_v})$$

where the kernel K will be usually a radially symmetric unimodal density function, for instance the multivariate normal density, and h_v represents the bandwidth of the *v*-th predictor. There are several approaches to select the optimal bandwidth. Hand (1982) is a good reference for application of kernel density estimators in classification.

In the Statlog Project (Michie, et. al. 1994), where 23 classifiers are compared in 22 datasets, classifiers based on kernel density estimators (ALLOC80) performed very well. In the Statlog Project (Michie, et. al. 1994), where 23 classifiers are compared in 22 datasets, classifiers based on kernel density estimators (ALLOC80) performed better than CART (a 13-8 victory) and tied with C4.5 (11-11). However, ALLOC80 appeared as the top 5 classifier for 11 datasets whereas C4.5 and CART appeared only for 5 and 3 datasets respectively. Classifiers based on kernel density estimation are unstable due to singularities presented in the log-likelihood function, and the selection of the bandwidth is affected by the presence of outliers. For categorical predictors we have followed the Titterington's proposal.

Table 5. Comparison of error rates for Kernel classifiers with the best classifier

Dataset	Cases	Attributes	Classes	Kernel Classifier	Best Single Classifier
Iris	150	4c	3	3.33	2.0 LDA
Glass	214	9c	6	27.1	23.8 1-NN
Hepatitis	155	6c, 13d	2	30.0	19.9 NN
Heart-c	303	8c, 5d	2	25.33	18.2 NN
Diabetes	768	8c	2	28.25	22.3 Log
Breast-w	699	9c	2	4.43	3.3 NN

5. Improving the Kernel density estimator using Gaussian mixtures

To produce kernel estimators that are better able to adapt to local features of the data it seems natural to attempt to use estimators whose bandwidths change locally as a function of the point estimation, they are called Adaptive kernel estimates. Marchette, et. al. (1996) proposed a kernel estimator that uses a small number of bandwidths and named the filtered kernel estimator, defined as

$$\hat{f}(x) = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m}\frac{r_j(x_i)}{h_j}K(\frac{x-x_i}{h_j})$$

The filtered functions $\{r_j\}_{j=1}^{m}$ satisfy $0 \le r_j(x) \le 1$ and $\sum_{j=1}^{m} r_j(x) = 1$ for all x. If the density is estimated first by a finite mixture $f(x) = \sum_{i=1}^{m} p_j f_j(x)$ then $r_j(x) = p_j f_j(x) / f(x)$. However they only considered the univariate case.

Hjort and Glad (1995) also considered a version of kernel estimator that is closely related to the above discussed, their idea is to use a parametric density as a start on the kernel estimator. They proposed a multivariate version but they did not go further.

Priebe and Marchette (1999) proposed an alternating kernel and mixture estimator, where an iterative procedure is used, alternating parametric and nonparametric estimates. Again only the univariate case it was considered.

6. Combining classifiers based on kernel density estimation.

Classifiers based on Kernel density estimators and Gaussian mixtures are unstable due to singularities presented in the log-likelihood function, therefore it is expected that combining them will produce a better classifier.

Smyth and Wolpert (1998) used Stacking for combining classifiers based on kernel density estimators and in gaussian mixtures.

6.1 Experimental Setup

We chose 8 datasets coming from the Machine Learning Database at UCI to evaluate the effect of combining KDE classifiers considering mixed type of predictors: continuous and categorical and fixed as well as adaptive bandwidths. For each dataset we have performed the following procedure: The dataset is randomly divided in 10 parts, the first one is taken as the test sample and the remaining is considered as the learning sample. Next, 50 bootstrapped samples are taking from the learning sample and a KDE classifier is constructed with each of them. Finally, each instance of the test sample is assigned to a class by voting using the 50 classifiers previously constructed. The proportion of instances incorrectly assigned will be the bagged misclassification error. We repeat the steps considering now the second part as the test set and in this way we continue until the tenth part is considered as the test set. The procedure is repeated 10 times. The misclassification error of a single classifier is estimated by a 10-fold crossvalidation and averaged over 10 runs. The bagged misclassification error is averaged on 100 repetitions. We also computed the relative error reduction (Improv.). A S-plus function (see Appendix) has been developed to carry out all our tasks. The results are shown in the table 7.

Table 6. Comparison of Bagging using classical and adaptive kernel classifiers

Dataset	Classical Kernel			Adaptive Kernel		
	Single	Bagged	Improv	Single	Bagged	Improv
Iris	4.00	3.33	16.75	4.67	4.00	14.34
Glass	44.97	40.52	9.90	35.20	33.25	5.54
Heart-C	22.09	20.05	9.23	23.60	19.80	16.10
Breast-W	4.34	4.10	5.53	4.88	4.53	7.17
Diabetes	25.43	25.05	1.49	26.46	25.92	2.04
Vehicle	35.62	32.54	8.65	37.17	33.58	9.66
Credit-G	37.00	35.21	4.84	36.17	34.21	5.42
Segment	15.76	14.25	9.58	13.33	12.54	5.92

The average of the error reduction for the 8 datasets after Bagging using the classical Kernel was 8.25% whereas for the Adaptive Kernel was 8. 27%. When C4.5 classifer was bagged (Quinlan. 1997) the average error reduction for the same datasets was 9.07%, quite similar to our result. Presently we are carrying out feature selection to deal with the curse of dimensionality and so far we are getting good results.

7. Future Work

A) To make a large study including 24 datasets. All of them have been analyzed already in a combining setup and nine of these datasets were analyzed in the Statlog Project (Michie, et. al. 1994).

B) To make an analysis of the bias-variance decomposition for the misclassification error when classifiers based in kernel density estimators and finite gaussian mixtures are combined.

C) To make an study of the training time of the ensembles.

D) To implement algorithms in parallel for combining classifiers based in kernel density estimators and finite gaussian mixtures.

References

Bauer, E. and Kohavi, R. (1999), "An Empirical comparison of voting classification algorithms: Bagging Boosting and variants". Machine Learning, 36, 105-139.

Blake, C. and Merz, C. (1998), "UCI repository of machine learning databases", Department of Computer Science and Information. University of California, Irvine.

Breiman, L. (1996b), "Bagging Predictors", Machine Learning, 26,123-140.

Breiman, L. (1998), "Arcing Classifiers". The Annals of Statistics, 26, 801-849.

Freund, Y and Schapire, R. (1996), "Experiments with a new boosting algorithm". In Machine Learning, Proceedings of the Thirteenth International Conference, 148-156., San Francisco, Morgan Kaufman..

Hand, D. (1982), Kernel Discriminant Analyis, Research Studies, New York.

Hastie, T. and Tibshirani, R. (1996), "Discriminant analysis by gaussian mixtures", Journal of the Royal Statistical Society, Ser. B, 58, 155-176.

Hjort, N.L, and Glad,I.K, (1995), "Nonparametric density estimation with a parametric start", The Annals of Statistics, 23, 882-904.

Maclin, R. and Opitz, D. (1997), "An Empirical evaluation of Bagging and Boosting". Proceedings of the Fourteenth National Conference on Artificial Intelligence, AAAI/MIT Press.

MacLachlan, G.J. (1992) Discriminant Analysis and Statistical Pattern Recogniton. New York; Wiley,.

Marchette, D.J, Priebe,C.E., Rogers,G.W, and Solka,J.L, (1996), "Filtered kernel density estimation". Computational Statistics, 11, 95-112.

Michie, D., Spigelhalter, D.J. and Taylor, C.C. (1994), Machine Learning, Neural and Statistical Classification. London: Ellis Horwood,.

Ormoneit, D. and Tresp, V. (1996) Improved Gaussian mixture density estimates using Bayesian penalty term and network averaging. Advances in Neural Information Processing, 8, 542-548. MIT Press.

Priebe, C. E. and Marchette, D.J. (1999), "Alternating kernel and mixture density estimates". Technical Report, Department of Mathematical Sciences, Johns Hopkins University.

Quinlan, J.R. (1996), "Bagging, Boosting and C4.5", Proceedings of the Thirteenth National Conference on Artifiicial Intelligence, 725-730. AAAI/MIT Press.

Smyth, P., and Wolpert, D. (1999), "An Evaluation of lineally combining density estimators via stacking", Machine Learning, 36, 59-83.

Taxt, T., Hjort, N. and Eikvil, L. (1991), "Statistical classification using a linear mixture of multinormal probability densities". Pattern Recognition Letters, 12, 731-737.

Titterington, D.M. (1980). A comparative study of kernel-based density estimates for categorical data. *Technometrics*, 22, 259-268.

Titterington, D.M. , Smith, A.F.M., and Makov, U.E. (1985), Statistical analysis of finite mixture distributions. New York,Wiley.

Wolpert, D. (1992), "Stacked Generalization". Neural Networks, 5 , 241-259.

Appendix

1) S-Plus program to compute the bandwidth

```
# Computation of the bandwidth for continuous predictors
# using Gaussian Kernel and Epanechnikov Kernel. Silverman(1986, pp 86-87)
# data       :matrix of data containing only continuous predictors.
# convari    :number of continuous  predictors.
# ndatagru   :group size
# ker        :type of kernel; gus:gaussian, epa:Epanechnikov


hnormul<-function(data,convari,ndatagru,ker="gus")
{
       if(convari==1)
       {
              data<-as.matrix(data)
       }
       sd <- sqrt(apply(data, 2, var))
       if(ker=="gus")
       {
              hh <-
sd*((4/(convari+2))^(1/(convari+4)))/(ndatagru^(1/(convari+4)))
       }
       if(ker=="epa")
       {
              Cd<-gamma(1+(convari/2))/(pi^(convari/2))
              hh <-
sd*((8*Cd*(convari+4)*((2*(pi^0.5))^convari))^(1/(convari+4)))/(ndatagru^(1/(co
nvari+4)))
       }
       hh
}


# Computation of bandwidth for binary variables, using two
# approaches: Kullback-Liebler and MMSE, Titterington(1980)
# The program compute one lambda for each variable and the average of them
# in case that only lambda one is going  to be used.
# data       :matrix of data containing only binary variables
# binvari    :number of binary variables
# ndatagru   :group size
# meto       :method; k:Kullback-Liebler, m:MMSE

hbina<-function(data,binvari,ndatagru,meto="k")
{
       data<-as.matrix(data)
       h<-matrix(0.5,1,binvari)
       if(meto=="m")
       {
              for(gf in 1:binvari)
              {
                     r<-mean(data[,gf])
                     Z<-((1-2*r)^2)*ndatagru
                     h[gf]<-1-(0.5*(ndatagru-Z)/(ndatagru+(ndatagru-1)*Z))
              }
       }
       else
       {
              for(gf in 1:binvari)
              {
                     r<-mean(data[,gf])
                     a1<-(ndatagru*(1-2*r))
```

```
                    a2<-(ndatagru*(2*r-1))
                    if(r==0.5)
                    {
                          h[gf]<-0.5
                    }
                    else
                    {
                          hprov<-((((1-r)^2)*a1)+((r^2)*a2))*ndatagru/(a1*a2)
                          h[gf]<-max(0.5,-hprov)
                    }
                    if(h[gf]>1)
                    {
                          h[gf]<-0.5
                    }
             }
       }
       hf<-matrix(mean(h),1,binvari)
       salida<-list(hf,h)
       salida
}


# Calculating the proportion of observations in the dataset
# that belong to a particular category for each categorical variable
#
# data       :matrix of data containing only categorical variables
# catvar     :number of categorical variables
# catego     :vector containing the number of categories of each
#             categorical variable
# ndatagru   :group size

proporc<-function(data,catvar,catego,ndatagru)
{
       salida<-list(0)
       data<-as.matrix(data)
       for(oi in 1:catvar)
       {
             proq<-rep(0,catego[oi])
             for(kl in 0:(catego[oi]-1))
             {
                   proq[kl+1]<-mean(as.numeric(data[,oi]==rep(kl,ndatagru)))
             }
             salida[oi]<-list(proq)
       }
       salida
}


# Computation of the bandwidth for nominal and ordinal variables,
# using the MMSE approach, see Titterington(1980).
# The program compute one lambda for each variable and the average of them
# in case that only lambda one is going  to be used.
# data       :matrix of data containing only categorical variables
# catvari    :number of categorical variables
# ndatagru   :group size
# tipv       :type of variable: ordinal o nominal



hcat<-function(data,catvar,catego,ndatagru,tipv="nom")
{
       data<-as.matrix(data)
       P<-proporc(data,catvar,catego,ndatagru)
```

```
	h<-matrix(0.5,1,catvar)
	for(fg in 1:catvar)
	{
		Pr<-matrix(unlist(P[fg]),ncol=1)
		V<-(diag(unlist(P[fg]))-(Pr%*%t(Pr)))/ndatagru
		G<-lambd(catego[fg],tipv)
		h[fg]<-
1+(sum(diag(V%*%G))/sum(diag((t(G)%*%Pr%*%t(Pr)%*%G)+(V%*%G%*%t(G)))))
	}
	hf<-matrix(mean(h),1,catvar)
	salida<-list(hf,h)
	salida
}



# Distancias y kernels para variables categoricas


# Calcula las distancias entre las observaciones y
# los datos de prueba por variable (continua)
# Tambien divide por la amplitud de la ventana para cada variable
#
# data       :matriz de los datos originales (continuos)
# prueba     :matriz de los datos a discriminar


distancia<-function(data, prueba, h,acomo)
{
	data<-as.matrix(data)
	prueba<-as.matrix(prueba)
	dis <- list(0)
	n<-length(data[,1])
	for(i in 1:length(h)) {
		dik<-matrix(unlist(acomo[i]),ncol=n)
		dis[i] <- list((t(outer(data[, i], prueba[, i], "-"))*dik)/h[i])
	}
	dis
}

# Kernel para variables binarias, usando un lambda para todas las
# variables (binarias), es un caso particular de la funcion
# distacat ( a describir mas adelante), pero esta es mas eficiente
# para este tipo de variables.
#
# Notese que varias de los parametros de la funcion se pueden
# hallar facilmente, pero como se esta haciendo el programa como un
# todo, estos parametros ya se tienn luego se ahorra un tiempo
#
# data       :matriz de los datos originales
# prueba     :matriz de los datos a discriminar
# npru       :cantidad de individuos de prueba
# ndata      :cantidad de individuos originalmente
# binvari    :cantidad de variables binarias
# lam        :es el ancho de la ventana ( uno para todas las variables)

distanbin<-function(data, prueba,npru,ndata,binvari,lam)
{
	if(binvari==1)
	{
		data<-as.matrix(data)
		prueba<-as.matrix(prueba)
```

```
	}
	if(npru==1)
	{
		prueba<-t(as.matrix(prueba))
	}
	dist<-matrix(0,npru,ndata)
	for(mn in 1:npru)
	{
		for(np in 1:ndata)
		{
			dif<-data[np,]-prueba[mn,]
			dist[mn,np]<-t(dif)%*%dif
		}
	}
	ker<-(lam^(binvari-dist))*((1-lam)^dist)
	ker
}

# Kernel para variables categoricas, usando un lambda para cada
# variable (categorica)
#
# Notese que varias de los parametros de la funcion se pueden
# hallar facilmente, pero como se esta haciendo el programa como un
# todo, estos parametros ya se tienn luego se ahorra un tiempo
#
# data       :matriz de los datos originales
# prueba     :matriz de los datos a discriminar
# npru       :cantidad de individuos de prueba
# ndata      :cantidad de individuos originalmente
# catvari    :cantidad de variables categoricas
# cat        :es un vector con el numero de categorias para cada
#             variable categorica
# lam        :es el ancho de la ventana ( uno por cada variable)
# tipv       :tipo de variable categorica con la que se trabaja,
#             nominal o ordinal, ya que los lambdas dependen de esta
#             escogencia

distacat<-function(data,prueba,npru,ndata,catvari,cat,lam,tipv="nom")
{
	if(catvari==1)
	{
		data<-as.matrix(data)
		prueba<-as.matrix(prueba)
	}
	if(npru==1)
	{
		prueba<-t(as.matrix(prueba))
	}
	est<-matrix(1,npru,ndata)
	for(mn in 1:npru)
	{
		for(pq in 1:catvari)
		{
			lambdas<-diag(cat[pq])+(1-lam[pq])*lambd(cat[pq],tipv)
			for(np in 1:ndata)
			{
				prov<-lambdas[prueba[mn,pq]+1,data[np,pq]+1]
				est[mn,np]<-est[mn,np]*prov
			}
		}
	}
	est
}
```

```
# Calculo de la matriz necearia para calcular los lambdas, G
# para variables nominales y ordinales, como en el articulo de
Titteringtон(1980)
# La matriz de lambdas sera C=I-(1-lambda)*G.
# k    :es la cantidad de categorias de la variable
# tvar :es el tipo de variable, ya que existen diferencias al ponderar

lambd<-function(k,tvar="nom")
{
	mlamb<-matrix(0,k,k)
	mlambn<-matrix(0,k,k)
	for(i in 1:k)
	{
		for(j in 1:k)
		{
			if(i==j)
			{
				mlamb[i,j]<- -1
				mlambn[i,j]<- -1
			}
			else
			{
				mlambn[i,j]<-1/(k-1)
				if(i>j)
				{
					mlamb[i,j]<-(2*j/((k-1)*i))
				}
				else
				{
					mlamb[i,j]<-(2*(k+1-j)/((k-1)*(k+1-i)))
				}
			}
		}
	}
	if(tvar=="ord")
	{
		mlambn<-mlamb
	}
	mlambn
}



# Calcula el Kernel univariado para cada una de las variables
# continuas, sin dividir por n, ni por el ancho de banda,
# pero si por las ponderaciones usadas para kernel variable o adaptativo
# Hay tres posibilidades para el kernel: gau:gaussiano
# epa:epanechnikov y biw: biweight
# dist       :distancias {(x-xi)/h}
# ker        :tipo de kernel a usar

kerdist<-function(dist, ker = "gau", acomov)
{
	indprue <- length(dist[, 1])
	inddata <- length(dist[1,  ])
	if(ker == "gau") {
		kercon <- matrix(dnorm(dist), ncol = inddata)
	}
	else if(ker == "epa") {
		verdad <- as.numeric(abs(dist) < (5^0.5))
		dd <- verdad * dist
```

```
            kercon <- (3/(4 * (5^0.5))) * (1 - (dd^2)/5) * verdad
      }
      else if(ker == "biw") {
            verdad <- as.numeric(abs(dist) < 1)
            dd <- verdad * dist
            kercon <- (15/16) * ((1 - (dd^2))^2) * verdad
      }
      kercon <- kercon * acomov
      kercon
}


# Realiza el producto de los kernels univariados, sin dividir
# por el numero total de individuos ni por los anchos de banda.
# distan     :lista con las distancias {(x-xi)/h} para
#             cada una de las variables (continuas)
# vari       :cantidad de variables continuas
# indid      :cantidad total de individuos en la base de datos
# ker        :tipo de kernel a usar

kerdis<-function(distan, vari, indid, ker = "gau", acomo)
{
      prod <- kerdist(matrix(unlist(distan[1]), ncol = indid), ker,
matrix(unlist(acomo[1]), ncol = indid))
      if(vari > 1) {
            for(nn in 2:vari) {
                  prod <- prod * kerdist(matrix(unlist(distan[nn]), ncol =
indid), ker, matrix(unlist(
                        acomo[nn])^(1/vari), ncol = indid))
            }
      }
      prod
}


# Calcula el kernel ("normal", variable y adapatativo) para cualquier tipo de
variable.
# Si no hay algun tipo de variable, asigne 0 a var* y a categ*
# data       :matriz con los datos de entrenamiento
# pruebav    :matriz con los datos de prueba, incluyendo su clasificacion
# varcon     :vector que indica las columnas donde estan las variables
continuas
# varbin     :vector que indica las columnas donde estan las variables binarias
# varnom     :vector que indica las columnas donde estan las variables
nominales
# categn     :vector con el numero de categorias por variable nominal
# varord     :vector que indica las columnas donde estan las variables
ordinales
# catego     :vector con el numero de categorias por variable ordinal
# ngrupos    :cantidad de grupos
# hgru       :vector de ancho de ventana para cada variable continua
#            Puede ser estimado con ucv,bcv,hsj,hcv
# lam1       :vector de ancho de ventana para cada variable binaria o puede ser
un solo valor
# lam2       :vector de ancho de ventana para cada variable nominal
# lam3       :vector de ancho de ventana para cada variable ordinal
# ker        :tipo de kernel que se quiere usar, hay tres posibilidades:
#             gau:gaussiano epa:epanechnikov y biw: biweight
# prop       : 1 si se desea ponderar por las proporciones, 0 si no
# variable   : 1 si se desea que el kernel sea variable( Silverman, 1986)
#             y 0 si no.
# knn        : vector que contiene el parametro k (vecinos mas cercanos) pra
#             cada grupo
```

```
# adapt      : 1 realiza el kernel adaptativo, 0 no

kertot<-function(data, pruebav, varcon, varbin, varnom, categn, varord, catego,
ngrupos, ker = "gus", prop = 1,
      variable = 0, knve = 0,adapt=0, hgru, lam1, lam2, lam3)
{
# Encuentra diferentes caracteristicas del conjunto de datos
      salida <- list(0)                       # inicializa la lista de salida
      pruebav <- as.matrix(pruebav)
      grupo <- length(pruebav[1,  ])          # columna en la que esta la
clasificacion
      totvari <- grupo - 1                           # numero total de
variables predictoras
      nprueba <- length(pruebav[, 1])         # numero de individuos de prueba
      binvari <- length(varbin)               # numero de variables binarias
      nomvari <- length(varnom)               # numero de variables nominales
      ordvari <- length(varord)               # numero de variables ordinales
      convari <- length(varcon)               # numero de variables continuas

# Separa los datos "originales" por clases, y los datos de prueba por tipo de
variable

      porgru <- separar(data, grupo, ngrupos) # separa la base de datos por
clases
      prueba <- pruebav[,  - grupo]
      clases <- pruebav[, grupo]

# separa el archivo de prueba por tipo de variable

      portipop <- tipo(prueba, varcon, varbin, varnom, varord, totvari)

# Inicializa las matrices que contiene el kernel final y las proporciones

      kerfin <- matrix(0, nprueba, ngrupos)
      proporc <- matrix(0, 1, ngrupos)

###
###   Realiza la estimacion de la densidad por grupos
###

      for(kk in 1:ngrupos) {
            porgrupo <- matrix(unlist(porgru[kk]), ncol = totvari)
            ndatagru <- length(porgrupo[, 1])# numero de individuos en el
grupo
            proporc[1, kk] <- ndatagru/length(data[, 1])

# separa el archivo de datos por tipo de variable

            portipod <- tipo(porgrupo, varcon, varbin, varnom, varord,
totvari)
##
## Kernel por tipo de variable
##

## Continuas
            if(varcon[1] > 0) {
                  ccdatos <- matrix(unlist(portipod[1]), ncol = convari)
                  ccprueba <- matrix(unlist(portipop[1]), ncol = convari)
                  if(missing(hgru)) {
                        hgru <- hnormul(ccdatos, convari, ndatagru, ker)
                  }
                  denomina <- (exp(rowSums(matrix(log(hgru), 1, convari))))
                  if(adapt==0){
```

```
                        acomo <- varia(ccdatos, ndatagru,
nprueba,adap=0,lambdaa=0,knve, variable)
                        disgruco <- distancia(ccdatos, ccprueba, hgru, acomo)
                        denomina <- (exp(rowSums(matrix(log(hgru), 1,
convari))))
                        kercont <- kerdis(disgruco, convari, ndatagru, ker =
"gau", acomo)/denomina
                  }
                  else{

# Kernel Piloto
                        acomo <- varia(ccdatos, ndatagru,
ndatagru,adap=0,lambdaa=0,k=0, v=0)
                        disgruco <- distancia(ccdatos, ccdatos, hgru, acomo)
                        kercont <- kerdis(disgruco, convari, ndatagru, ker =
"gau", acomo)/denomina
                        kerpilot<- rowSums(kercont)/ndatagru

# kernel adaptativo

                        medgeo<-exp(sum(log(kerpilot))/ndatagru)
                        lambb<-((kerpilot/medgeo)^(-0.5))
                        acomo2 <- varia(ccdatos, ndatagru,
nprueba,adap=1,lambdaa=lambb,k=0,v=0)
                        disgruco2 <- distancia(ccdatos, ccprueba, hgru,
acomo2)
                        kercont<-kerdis(disgruco2, convari, ndatagru, ker =
"gau", acomo2)/denomina
                  }
            }
            else {
                  kercont <- matrix(1, nprueba, ndatagru)
            }
            if(varbin[1] > 0) {
                  cdatos <- matrix(unlist(portipod[2]), ncol = binvari)
                  cprueba <- matrix(unlist(portipop[2]), ncol = binvari)
                  if(missing(lam1)) {
                        lam1 <- unlist(hbina(cdatos, binvari, ndatagru)[2])
                        categor <- rep(2, length(lam1))
                        kerbin <- distacat(cdatos, cprueba, nprueba,
ndatagru, binvari, categor, lam1,"nom")
                  }
                  else {
                        if(length(lam1) == 1) {
                          kerbin <- distanbin(cdatos, cprueba, nprueba,
ndatagru, binvari, lam1)
                        }
                        else {
                          categor <- rep(2, length(lam1))
                          kerbin <- distacat(cdatos, cprueba, nprueba,
ndatagru, binvari, categor,
                              lam1, "nom")
                        }
                  }
            }
            else {
                  kerbin <- matrix(1, nprueba, ndatagru)
            }
            if(varnom[1] > 0) {
                  cndatos <- matrix(unlist(portipod[3]), ncol = nomvari)
                  cnprueba <- matrix(unlist(portipop[3]), ncol = nomvari)
                  if(missing(lam2)) {
```

```
                        lam2 <- unlist(hcat(cndatos, nomvari, categn,
ndatagru, "nom")[2])
                  }
                  kernom <- distacat(cndatos, cnprueba, nprueba, ndatagru,
nomvari, categn, lam2, "nom")
            }
            else {
                  kernom <- matrix(1, nprueba, ndatagru)
            }
            if(varord[1] > 0) {
                  codatos <- matrix(unlist(portipod[4]), ncol = ordvari)
                  coprueba <- matrix(unlist(portipop[4]), ncol = ordvari)
                  if(missing(lam3)) {
                        lam3 <- unlist(hcat(codatos, ordvari, catego,
ndatagru, "ord")[2])
                  }
                  kerord <- distacat(codatos, coprueba, nprueba, ndatagru,
ordvari, catego, lam3, "ord")
            }
            else {
                  kerord <- matrix(1, nprueba, ndatagru)
            }
            kerfin[, kk] <- rowSums(kercont * kerbin * kernom *
kerord)/ndatagru
      }

# Pondera la estimacion de densidad por las proporciones, si asi se desea


      if(prop == 1) {
            kerfin <- kerfin * (matrix(1, nprueba, 1) %*% proporc)
      }
      maximo <- matrix(0, nprueba, 1)
      dond <- matrix(0, nprueba, ngrupos)

# Busca la clase a la que pertenece
# Los grupos deben llamarse 1,2,3...

      for(tt in 1:nprueba) {
            maximo[tt, 1] <- max(kerfin[tt,  ])
            dond[tt,  ] <- as.numeric(kerfin[tt,  ] == maximo[tt, 1])
      }
      donde <- dond %*% matrix(seq(1:ngrupos), ncol = 1)
      salida[1] <- list(cbind(clases, donde))                          #
Calculo del error
      salida[2] <- list(1 - mean(as.numeric(clases == donde)))         #
Tabla de Clasificacion
      salida[3] <- list(tabla(clases, donde, ngrupos))                 #
Salida de la clasificacion final y el error
      salida
}



# Sacar muestras bootstrapping

MuesBoo<-function(data,muestras=100,nmuestra)
{
      n<-length(data[,1])
      salida<-list(0)
      if(missing(nmuestra))
      {
            nmuestra<-n
```

```
	}
	for(i in 1:muestras)
	{
		indmues<-c(floor(runif(nmuestra,1,n+1)))
		salida[i]<-list(data[indmues,])
	}
	salida[muestras+1]<-list(data)
	salida
}


# Creacion de los folderes

Folderes<-function(data,nfold=10)
{
	salida<-list(0)
	n<-length(data[,1])
	baraja<-rank(runif(n,0,1))
	azar<-data[baraja,]
	parti<-floor(n/nfold)
	sobra<-n%%nfold
	for(j in 1:nfold)
	{
		cc<-((j-1)*parti+1):(j*parti)
		if(j==nfold)
		{
			cc<-((j-1)*parti+1):(j*parti+sobra)
		}
		salida[2*j]<-list(azar[c(cc),])
		salida[2*j-1]<-list(azar[-c(cc),])
	}
	salida
}

# Error Validacion Cruzada para kernel

ErrorVCk<-function(lista, nfold, ntvar, adt=0,vari, knv)
{
	if(missing(nfold)) {
		nfold <- ask("Digite por favor el numero de folderes")
	}
	vc <- ask("Digite como vector las variables continuas")
	vb <- ask("Digite como vector las variables binarias")
	vn <- ask("Digite como vector las variables nominales")
	vo <- ask("Digite como vector las variables ordinales")
	cn <- ask("Digite como vector el numero de categorias para las variables
nominales")
	co <- ask("Digite como vector el numero de categorias para las variables
ordinales")
	ng <- ask("Digite el numero de clases")
	salida <- matrix(0, 1, nfold)
	for(m in 1:nfold) {
		dato <- matrix(unlist(lista[2 * m - 1]), ncol = ntvar)
		prue <- matrix(unlist(lista[2 * m]), ncol = ntvar)
		prov <- kertot(dato, prue, vc, vb, vn, cn, vo, co, ng, ker =
"gus", adapt=adt,variable = vari,knve = knv)
		salida[m] <- unlist(prov[2])
	}
	print("ESTIMACION DEL ERROR POR VALIDACION CRUZADA")
	print("KERNEL")
	ECV <- mean(salida)
	print(ECV)
	ECV
```

```
}


# Calculo del error por el metodo de refinado de Bootstraping

ErrorBk<-function(lista, B, ntvar, adt,vari, knv)
{
       eb <- matrix(0, 1, B)
       et <- matrix(0, 1, B)
       if(missing(B)) {
              B <- ask("Digite por favor el numero de muestras bootstrap")
       }
       vc <- ask("Digite como vector las variables continuas")
       vb <- ask("Digite como vector las variables binarias")
       vn <- ask("Digite como vector las variables nominales")
       vo <- ask("Digite como vector las variables ordinales")
       cn <- ask("Digite como vector el numero de categorias para las variables
nominales")
       co <- ask("Digite como vector el numero de categorias para las variables
ordinales")
       ng <- ask("Digite el numero de clases")
       orig <- matrix(unlist(lista[B + 1]), ncol = ntvar)
       prov <- kertot(orig, orig, vc, vb, vn, cn, vo, co, ng,variable = vari,
knve = knv, adapt=adt)
       ea <- unlist(prov[2])
       for(m in 1:B) {
              dato <- matrix(unlist(lista[m]), ncol = ntvar)
              eb[m] <- unlist(kertot(dato, dato, vc, vb, vn, cn, vo, co, ng)[2])
              et[m] <- unlist(kertot(dato, orig, vc, vb, vn, cn, vo, co, ng)[2])
              print(et[m] - eb[m])
       }
       print("ESTIMACION DEL ERROR POR BOOTSTRAPPING")
       print("KERNEL")
       Sesgo <- mean(et - eb)
       print("Error por Restitucion")
       print(ea)
       print("SESGO")
       print(Sesgo)
       EBoot <- ea + Sesgo
       print(EBoot)
       EBoot
}


# Manejo de los datos


# Separar el archivo por categorias
# data:      matriz de datos que se desea separar
# colgru:    columna de la matriz en la que se encuentran las clases
# numgru:    cantidad de grupos

separar<-function(data, colgru, numgru)
{
       dagrup <- list(0)
       sobra <- length(data[1,  ]) - 1
       f <- split(data[,  - colgru], data[, colgru])
       for(j in 1:numgru) {
              prov <- list(matrix(unlist(f[j]), ncol = sobra))
              dagrup[j] <- prov
       }
       dagrup
}
```

```
# Separar el archivo por tipo de variables
# Si no hay algun tipo de variable, digite 0 en el parametro correspondiente de
la funcion
# data:      matriz de datos que se desea separar
# varcon:    columnas donde estan las variables continuas
# varbin:    columnas donde estan las variables binarias
# varnom:    columnas donde estan las variables nominales de mas de dos
categorias
# varord:    columnas donde estan las variables ordinales de mas de dos
categorias
# pantalla:  es una variable auxiliar, que se encarga de imprimir un mensaje de
advertencia
#            para cuando no hay algun tipo especifico de variable

tipo<-function(data, varcon, varbin, varnom, varord, totvari)
{
       data <- matrix(data, ncol = totvari)
       if(varcon[1] == 0) {
              datacon <- 0
       }
       else {
              datacon <- matrix(data[, varcon], ncol = length(varcon))
       }
       if(varbin[1] == 0) {
              databin <- 0
       }
       else {
              databin <- matrix(data[, varbin], ncol = length(varbin))
       }
       if(varnom[1] == 0) {
              datanom <- 0
       }
       else {
              datanom <- matrix(data[, varnom], ncol = length(varnom))
       }
       if(varord[1] == 0) {
              dataord <- 0
       }
       else {
              dataord <- matrix(data[, varord], ncol = length(varord))
       }
       tipovar <- list(datacon, databin, datanom, dataord)
       tipovar
}

# Tabla de clasificacion
# En esta aparece la tabla de clasificacion y los totales por grupos
# En la salida aparecen unos NA, que no quieren decir nada.
# origina:   corresponde a la variable que tiene las categorias
# estimado:  corresponde a la variable que tiene las categorias estimadas
# ngrupos:   es la cantidad de grupos que hay

tabla<-function(origina,estimado,ngrupos)
{
       salida<-matrix(0,ngrupos+2,ngrupos+2)
       for(hj in 1:ngrupos)
       {
              CONTRg<-as.numeric(origina==hj)
              salida[hj+1,(ngrupos+2)]<-sum(CONTRg)
              for(jl in 1:ngrupos)
              {
```

```
                    CONTRe<-as.numeric(estimado==jl)
                    salida[hj+1,jl+1]<-sum(as.numeric(CONTRg*CONTRe))
                    salida[(ngrupos+2),jl+1]<-sum(CONTRe)
             }
       }
       salida[1,]<-c(NA,seq(1:ngrupos),NA)
       salida[,1]<-c(NA,seq(1:ngrupos),NA)
       salida[ngrupos+2,ngrupos+2]<-length(origina)
       salida
}
# Calculo de las ponderaciones dependiendo del metodo de kernel

varia<-function(data, ndatagru, nprueba, adap = 0, lambdaa, k, v = 0)
{
       data <- as.matrix(data)
       lambdda <- matrix(lambdaa, ncol = 1)
       salida <- list(0)
       orden <- matrix(0, 1, ndatagru)
       aco <- matrix(1, nprueba, 1)
       defau <- matrix(1, nprueba, ndatagru)
       n <- length(data[1,  ])
       if(v == 0) {
              for(j in 1:n) {
                    salida[j] <- list(defau)
              }
       }
       else {
              for(j in 1:n) {
                    dis <- abs(outer(data[, j], data[, j], "-"))
                    for(i in 1:ndatagru) {
                           orden[i] <- quantile(dis[i,  ], probs = (k/(ndatagru
- 1)))
                    }
                    salida[j] <- list(1/(aco %*% orden))
              }
       }
       if(adap == 1) {
              for(j in 1:n) {
                    salida[j] <- list(1/(aco %*% lambdaa))
              }
       }
       salida
}


Errou<-function(data,ntvar,n)
{
       vc <- ask("Digite como vector las variables continuas")
       vb <- ask("Digite como vector las variables binarias")
       vn <- ask("Digite como vector las variables nominales")
       vo <- ask("Digite como vector las variables ordinales")
       cn <- ask("Digite como vector el numero de categorias para las variables
nominales")
       co <- ask("Digite como vector el numero de categorias para las variables
ordinales")
       ng <- ask("Digite el numero de clases")
       salida <- matrix(0, 1, n)
       for(m in 1:n) {
              dato <- matrix(data[-m,], ncol = ntvar)
              prue <- matrix(data[m,], ncol = ntvar)
              prov <- kertot(dato, prue, vc, vb, vn, cn, vo, co, ng, ker =
"gus", prop = 1, variable = 0,knve = 0)
              salida[m] <- unlist(prov[2])
```

```
        }
        print("ESTIMACION DEL ERROR DEJANDO UNO AFUERA")
        print("KERNEL")
        ECV <- mean(salida)
        print(ECV)
        ECV
}
```