Evidential Classification

Thierry Denoeux

2023-08-15

Transportation mode

Question 1

The classes are unbalanced, the class `carpool` being underrepresented in the dataset:

```
library(Ecdat)
data(Mode)
table(Mode$choice)
```

```
##
##     car carpool     bus    rail
##     218      32      81     122
```

We thus remove observations of this class, as well as predictors `cost.carpool` and `time.carpool`:

```
y <- as.numeric(Mode[Mode$choice!='carpool',1])
y <- as.numeric(as.factor(y))
x <- scale(Mode[Mode$choice!='carpool',c(2,4,5,6,8,9)])
```

Question 2

We randomly split the data into training and test sets:

```
set.seed(2023)
n <- length(y)
train <- sample(1:n,round(2*n/3))
x.train <- x[train,]
x.test <- x[-train,]
y.train <- y[train]
y.test <- y[-train]
ntrain <- length(y.train)
```

Question 3

We train the EKNN classifier with $K = 5$ neighbors, and we compute the test error rate:

```
library(evclass)
fit <- EkNNfit(x.train,y.train,K=5,
               options = list(maxiter = 300, eta = 0.1,
                              gain_min = 1e-06, disp = FALSE))
```

```r
test <- EkNNval(x.train, y.train, x.test, K=5, y.test, fit$param)
test$err
```

```
## [1] 0.3285714
```

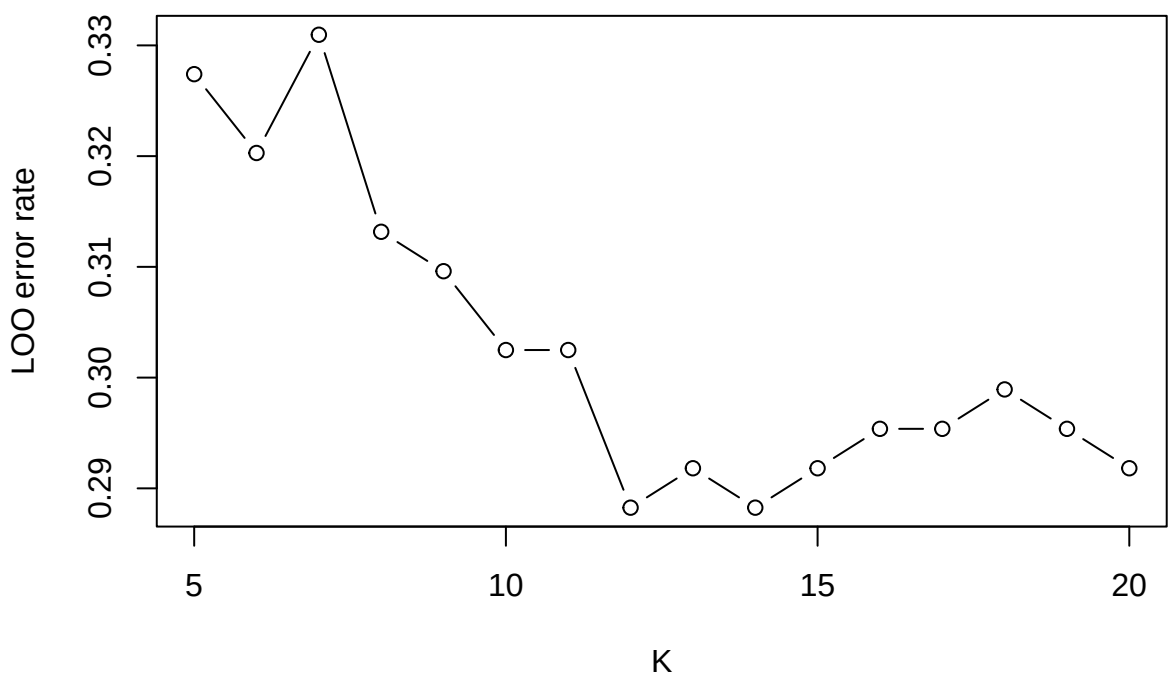
```r
table(as.numeric(y.test),test$ypred)
```

```
##
##      1  2  3
##   1 56  2 11
##   2  8 11 13
##   3  7  5 27
```

Question 4

We determine the obtimum number of K of neighbors by leave-one-out (LOO) cross-validation:

```r
K <- 5:20
N <- length(K)
err<-rep(0,N)
for(i in 1:N){
  fit <- EkNNfit(x.train,y.train,K[i],
                 options=list(maxiter=200,eta=0.1,gain_min=1e-5,disp=FALSE))
  err[i] <- fit$err
}
plot(K,err,type="b",xlab='K',ylab='LOO error rate')
```



We obtain the best value of K and run again the algorithm on the training set:

```r
Kopt <- K[which.min(err)]
print(Kopt)
```

```
## [1] 12
```

```r
fit <- EkNNfit(x.train,y.train,Kopt,
               options=list(maxiter=200,eta=0.1,gain_min=1e-5,
                            disp=FALSE))
test <- EkNNval(x.train, y.train, x.test, Kopt, y.test, fit$param)
test$err
```

```
## [1] 0.3142857
```

```r
table(as.numeric(y.test),test$ypred)
```

```
##
##      1  2  3
##   1 56  2 11
##   2  7 13 12
##   3  7  5 27
```

Question 5

We train the ENN classifier with 6 prototypes:

```r
param0 <- proDSinit(x.train, y.train, nproto=6)
fit <- proDSfit(x.train, y.train, param=param0)
```

```
## [1]  1.0000000  0.3033742 10.0000000
## [1] 11.0000000  0.2915428  3.4939118
## [1] 21.000000  0.267029  1.233739
## [1] 31.0000000  0.2735892  0.4551484
## [1] 41.0000000  0.2142243  0.2060261
## [1] 51.00000000  0.20151800  0.08850549
## [1] 61.00000000  0.19011173  0.04194725
## [1] 71.00000000  0.18275254  0.02365538
## [1] 81.00000000  0.18539621  0.01035022
## [1] 91.000000000  0.178524712  0.005292558
## [1] 1.010000e+02 1.771346e-01 2.751622e-03
## [1] 1.110000e+02 1.767349e-01 1.386058e-03
```

```r
test <- proDSval(x.test,fit$param,y.test)
test$err
```

```
## [1] 0.2785714
```

```r
table(y.test,test$ypred)
```

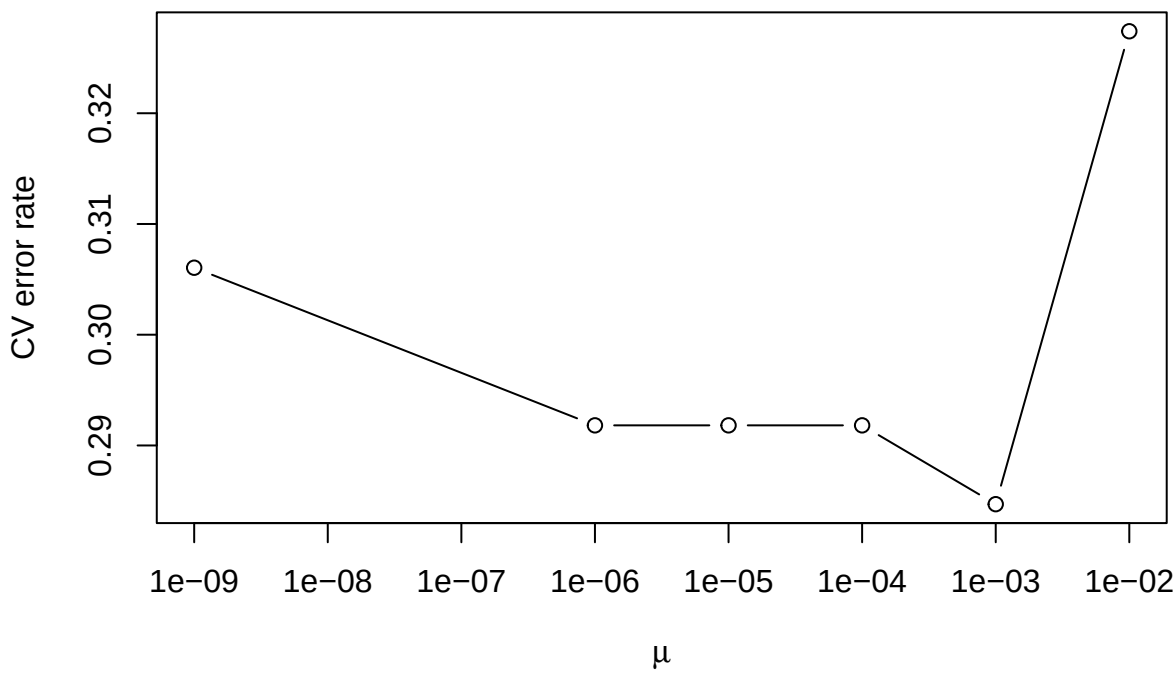
```
##
## y.test  1  2  3
##      1 63  2  4
##      2  8 13 11
##      3  8  6 25
```

The error rate is lower than that of the EKNN classifier.

Question 6

We use 5-fold cross-validation. The number of prototypes is set to 30.

```r
Kfold <- 5
folds <- sample(1:Kfold,ntrain,replace=TRUE)
MU <- c(1e-9,1e-6,1e-5,1e-4,0.001,0.01)
N <- length(MU)
CV <- rep(0,N)
nproto <- 30
param0 <- proDSinit(x.train, y.train, nproto=nproto,
                    nprotoPerClass = FALSE, crisp = FALSE)
options <- list(maxiter = 500, eta = 0.1, gain_min =1e-04,
                disp = 0)
for(i in (1:N)){
  for(k in (1:Kfold)){
    fit <- proDSfit(x.train[folds!=k,], y.train[folds!=k],
                    param=param0,mu=MU[i],options=options)
    val <- proDSval(x.train[folds==k,],fit$param,
                    y.train[folds==k])
    CV[i] <- CV[i]+ length(which(folds==k))*val$err
  }
  CV[i] <- CV[i]/ntrain
}
plot(MU,CV,type='b',xlab=expression(mu),ylab='CV error rate',log="x")
```



We train the best classifier on the training data, and evaluate it on the test data:

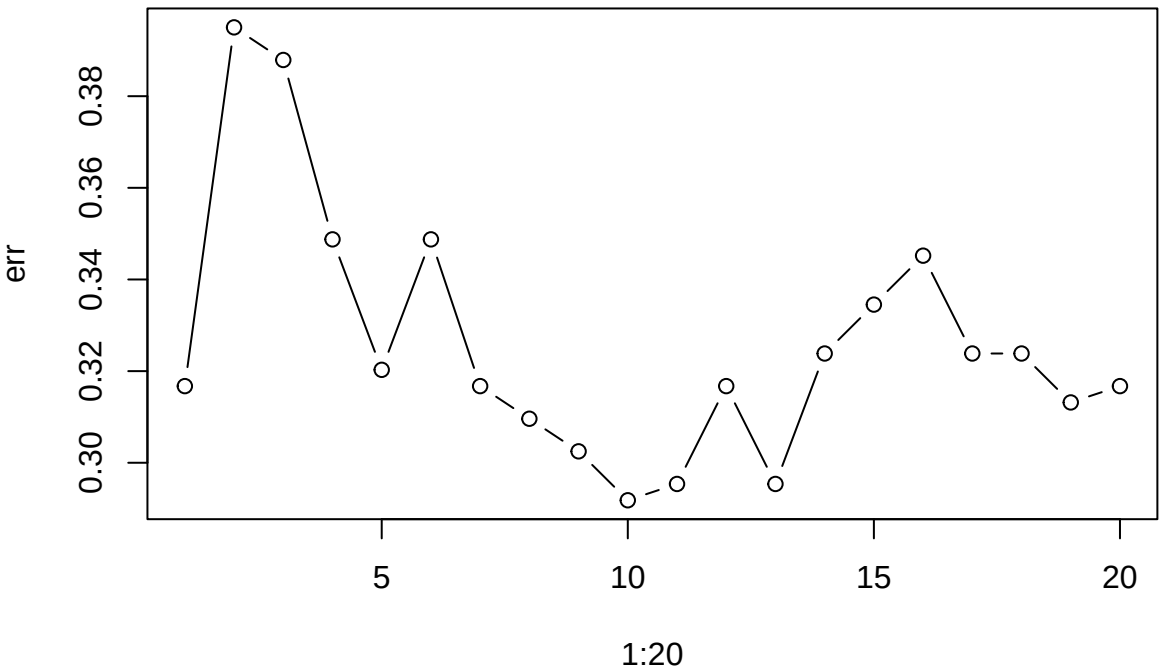
```r
muopt <- MU[which.min(CV)]
param0 <- proDSinit(x.train, y.train, nproto=nproto)
fit <- proDSfit(x.train, y.train, param=param0,mu=muopt,
                options=options)
test <- proDSval(x.test,fit$param,y.test)
test$err
```

```
## [1] 0.2714286
```

Question 7

Let us start with the voting KNN classifier. We first use function `knn.cv` from package `FNN` to determine the LOO error rate for different values of the number K of neighbors:

```
library(FNN)
err<-rep(0,20)
for(K in 1:20){
  pred<-knn.cv(x.train,y.train,K)
  err[K] <- mean(pred != y.train)
}
plot(1:20,err,type="b")
```




```
Kopt<-which.min(err)
```

We then use function `knn` to compute the test error rate for the best value of K:

```
test<-knn(x.train,x.test,y.train,Kopt)
err<-mean(test != y.test)
print(err)
```

```
## [1] 0.3285714
```

The error rate is not significantly different from that of the EKNN classifier.

Let us now try the MLP classifier. We use function `nnet` from package `nnet`. Let us start with a neural network with 5 hidden units:

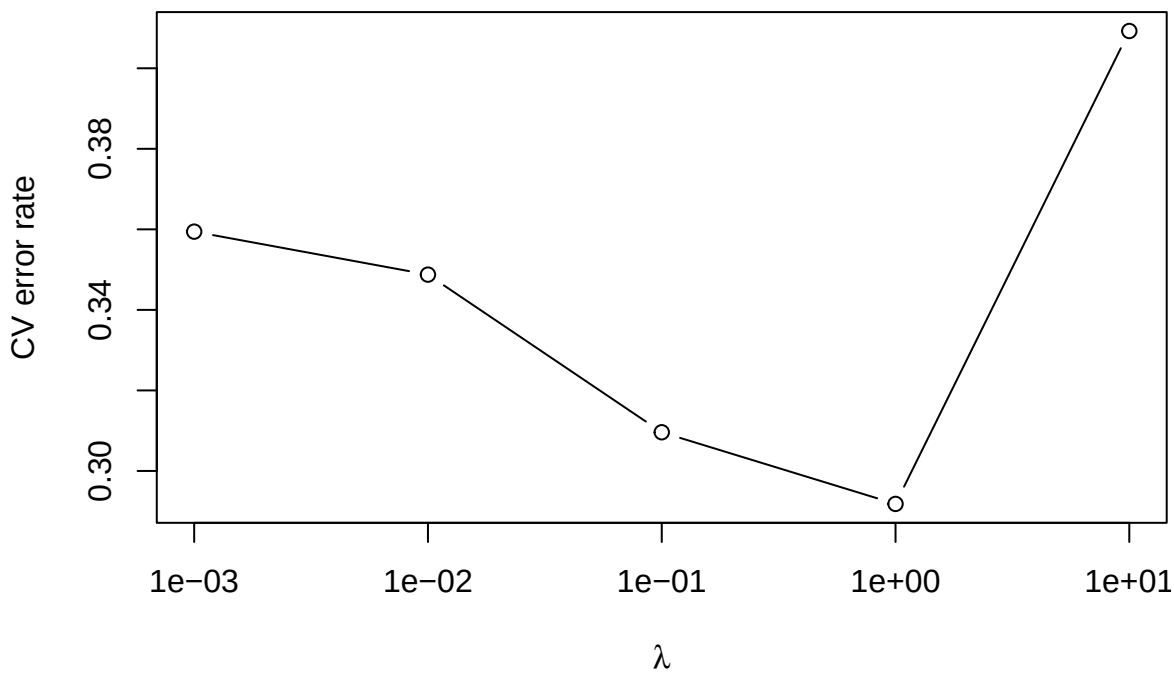
```
library(nnet)
Id <- diag(3)
Y.train <- Id[y.train,]
fit <- nnet(x.train,Y.train,entropy=TRUE,softmax=TRUE,maxit=1000,size=5,trace=FALSE)
pred <- predict(fit,newdata=x.test)
ypred <- max.col(pred)
err<-mean(ypred!=y.test)
```

```
print(err)
```

```
## [1] 0.3642857
```

As with the ENN model, we now tune the weight decay hyperparameter by 5-fold cross-validation, using 30 hidden units:

```
Lambda <- c(0.001,0.01,0.1,1,10)
N <- length(Lambda)
CV <- rep(0,N)
size <- 30
for(i in (1:N)){
  for(k in (1:Kfold)){
    fit <- nnet(x.train[folds!=k,],Y.train[folds!=k,],
              entropy=TRUE,softmax=TRUE,decay=Lambda[i],
              maxit=1000,size=size,trace=FALSE)
    val <- predict(fit,x.train[folds==k,])
    yval <- max.col(val)
    CV[i] <- CV[i]+ sum(yval != y.train[folds==k])
  }
  CV[i] <- CV[i]/ntrain
}
plot(Lambda,CV,type='b',xlab=expression(lambda),ylab='CV error rate',log="x")
```



We then retrain the network with the best weight decay parameter:

```
lambdaopt <- Lambda[which.min(CV)]
fit <- nnet(x.train,Y.train,entropy=TRUE,softmax=TRUE,
          maxit=1000,size=size,decay=lambdaopt,trace=FALSE)
pred <- predict(fit,newdata=x.test)
ypred <- max.col(pred)
err <- mean(ypred!=y.test)
print(err)
```

```
## [1] 0.2714286
```

Credit Approval

Question 1

We start by reading the data file:

```
credit<-read.csv("/Users/Thierry/Documents/R/Data/Economics/credit_approval/crx.data",
                 sep=",",header=FALSE)
```

We then remove observations with missing values (coded as "?" in the data file):

```
n <- nrow(credit)
missing <- rep(FALSE,n)
for(i in 1:n) if(any(credit[i,]=="?")) missing[i] <- TRUE
credit1 <- credit[missing==FALSE,]
```

Next, we declare categorical variables as factors, and quantitative variables as numeric. (I do it using as many instructions as variables; there must be a smarter way to do it, but I didn't find it).

```
credit1[,1]<-as.factor(credit1[,1])
credit1[,4]<-as.factor(credit1[,4])
credit1[,5]<-as.factor(credit1[,5])
credit1[,6]<-as.factor(credit1[,6])
credit1[,7]<-as.factor(credit1[,7])
credit1[,9]<-as.factor(credit1[,9])
credit1[,10]<-as.factor(credit1[,10])
credit1[,12]<-as.factor(credit1[,12])
credit1[,13]<-as.factor(credit1[,13])
credit1[,16]<-as.factor(credit1[,16])
credit1[,2]<-as.numeric(credit1[,2])
credit1[,3]<-as.numeric(credit1[,3])
credit1[,8]<-as.numeric(credit1[,8])
credit1[,11]<-as.numeric(credit1[,11])
credit1[,14]<-as.numeric(credit1[,14])
credit1[,15]<-as.numeric(credit1[,15])
```

Finally, we create an input matrix anda vector of class labels, to be used by the EKNN and ENNreg classifiers:

```
X <- model.matrix(V16 ~.,credit1)[,-1]
x <- scale(X)
y <- as.numeric(credit1$V16)
```

Question 2

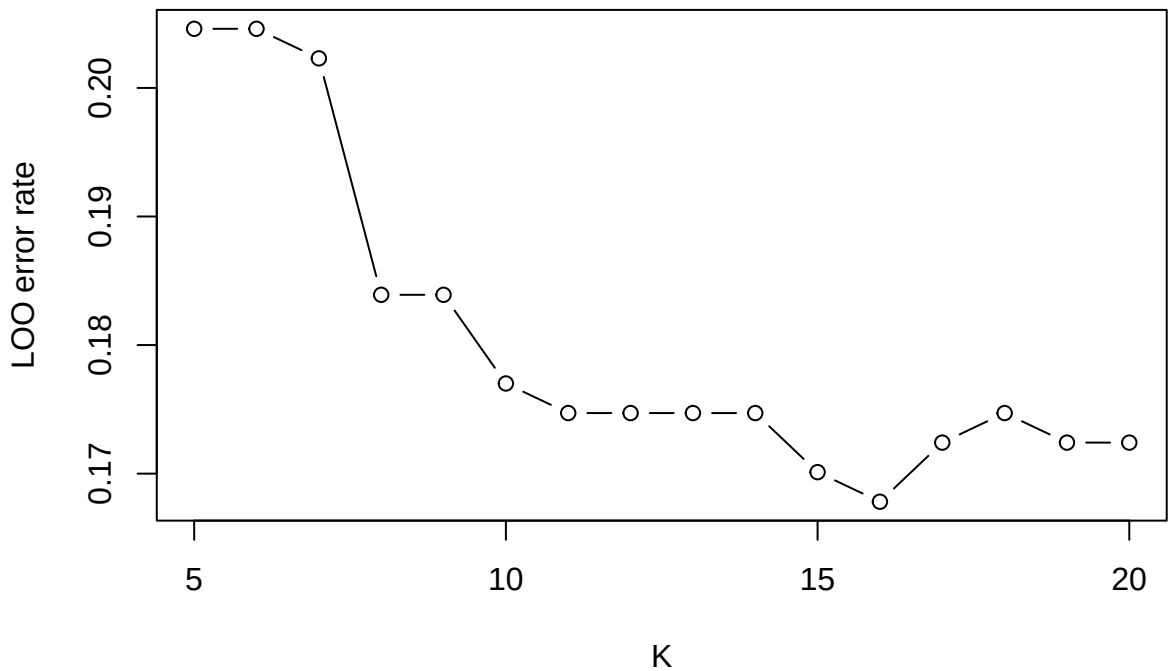
```
set.seed(2023)
n <- length(y)
train <- sample(1:n,round(2*n/3))
x.train<-x[train,]
x.test<-x[-train,]
y.train<-y[train]
```

```
y.test<-y[-train]
ntrain=length(y.train)
```

Question 3

Let us first determine the best value of K for the EKNN classifier using LOO cross-validation:

```
library(evclass)
err <- rep(0,15)
K <- 5:20
N <- length(K)
for(i in 1:N){
  fit <- EkNNfit(x.train,y.train,K[i],options=list(maxiter=200,eta=0.1,gain_min=1e-5,disp=FALSE))
  err[i] <- fit$err
}
plot(K,err,type="b",xlab='K',ylab='LOO error rate')
```




```
Kopt <- K[which.min(err)]
print(Kopt)
```

```
## [1] 16
```

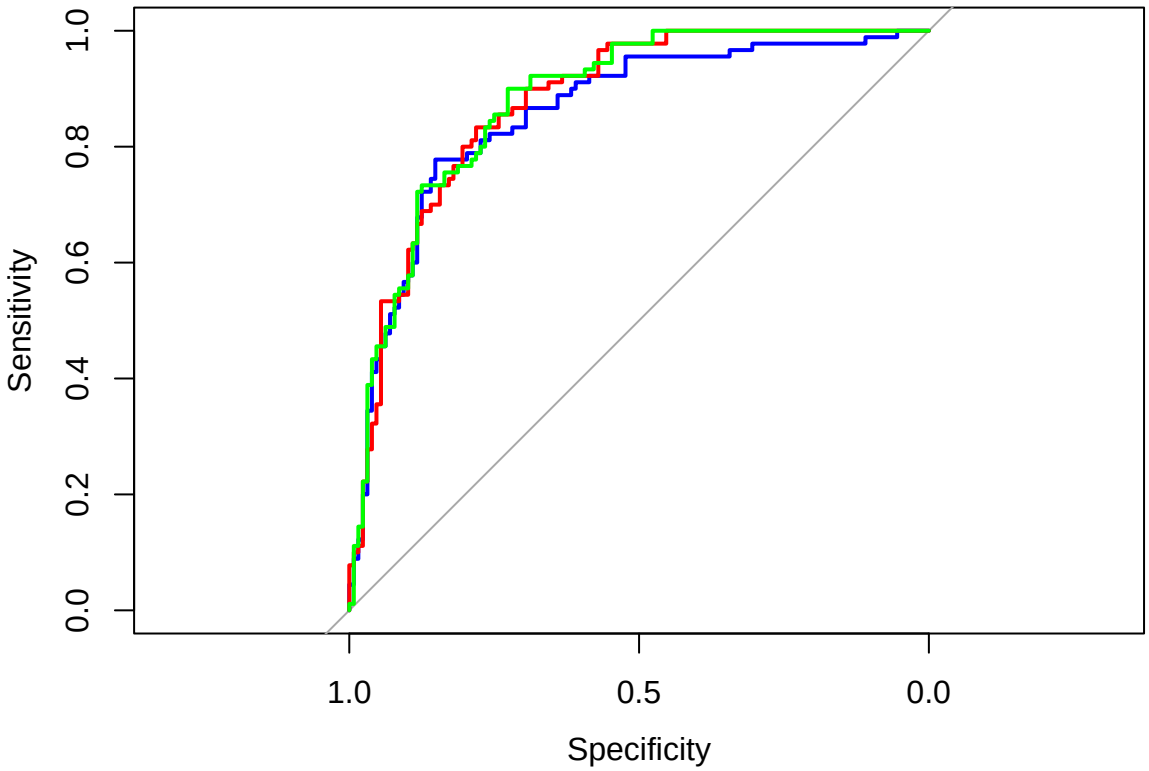
We fit again the parameters with the best value of K, and classify the test data:

```
fit <- EkNNfit(x.train,y.train,Kopt,options=list(maxiter=200,eta=0.1,gain_min=1e-5,disp=FALSE))
test <- EkNNval(x.train,y.train,x.test,Kopt,y.test,fit$param)
```

To plot the ROC curve, we use function `roc` in package `pROC`. We use the degree of belief, the degree of plausibility and the pignistic probability of class 2 (positive) as discriminant functions:

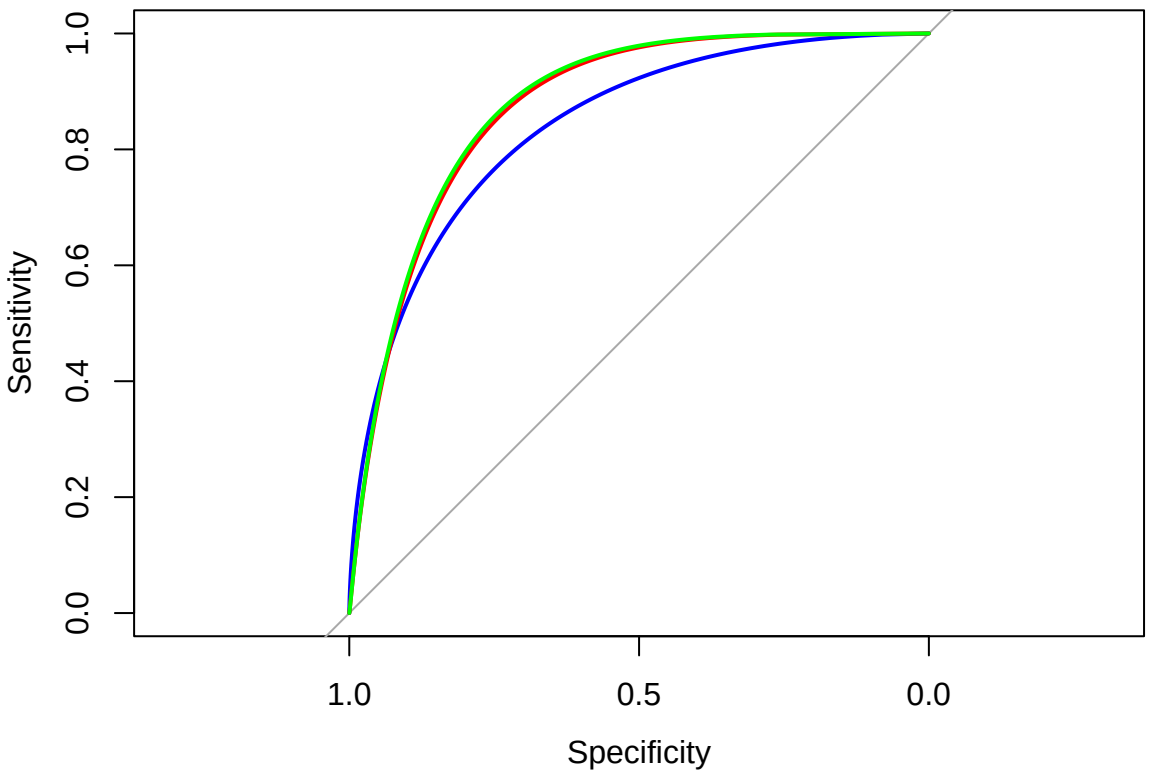
```
library(pROC)
curve1<-roc(y.test,test$m[,2])
curve2<-roc(y.test,test$m[,2]+test$m[,3])
curve3<-roc(y.test,test$m[,2]+test$m[,3]/2)
```

```r
plot(curve1,col="blue")
plot(curve2,add=TRUE,col="red")
plot(curve3,add=TRUE,col="green")
```



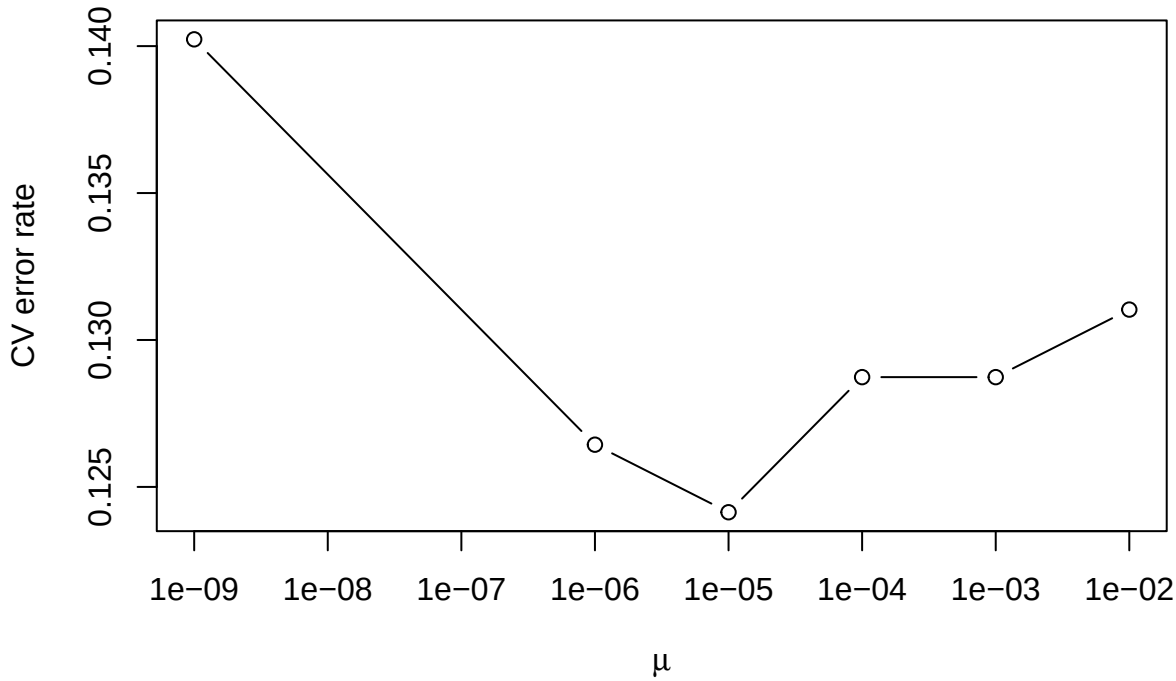
We observe that the plausibility and the pignistic give better results (a higher ROC curve). This can be better seen by plotting the smoothed curves:

```r
plot(smooth(curve1),col="blue")
plot(smooth(curve2),add=TRUE,col="red")
plot(smooth(curve3),add=TRUE,col="green")
```



Let us now use the ENNreg model with 30 prototypes. Hyperparameter μ is tuned by 5-fold cross-validation:

```
Kfold <- 5
folds <- sample(1:Kfold,ntrain,replace=TRUE)
MU <- c(1e-9,1e-6,1e-5,1e-4,0.001,0.01)
N <- length(MU)
CV <- rep(0,N)
nproto <- 30
param0 <- proDSinit(x.train, y.train, nproto=nproto,
                    nprotoPerClass = FALSE, crisp = FALSE)
options <- list(maxiter = 500, eta = 0.1, gain_min =1e-04,
                disp = 0)
for(i in (1:N)){
  for(k in (1:Kfold)){
    fit <- proDSfit(x.train[folds!=k,], y.train[folds!=k],
                  param=param0,mu=MU[i],options=options)
    val <- proDSval(x.train[folds==k,],fit$param,
                  y.train[folds==k])
    CV[i] <- CV[i]+ length(which(folds==k))*val$err
  }
  CV[i] <- CV[i]/ntrain
}
plot(MU,CV,type='b',xlab=expression(mu),ylab='CV error rate',log="x")
```



We train the best classifier on the training data, and evaluate it on the test data:

```
muopt <- MU[which.min(CV)]
param0 <- proDSinit(x.train, y.train, nproto=nproto)
fit <- proDSfit(x.train, y.train, param=param0,mu=muopt,
                options=options)
test <- proDSval(x.test,fit$param,y.test)
```

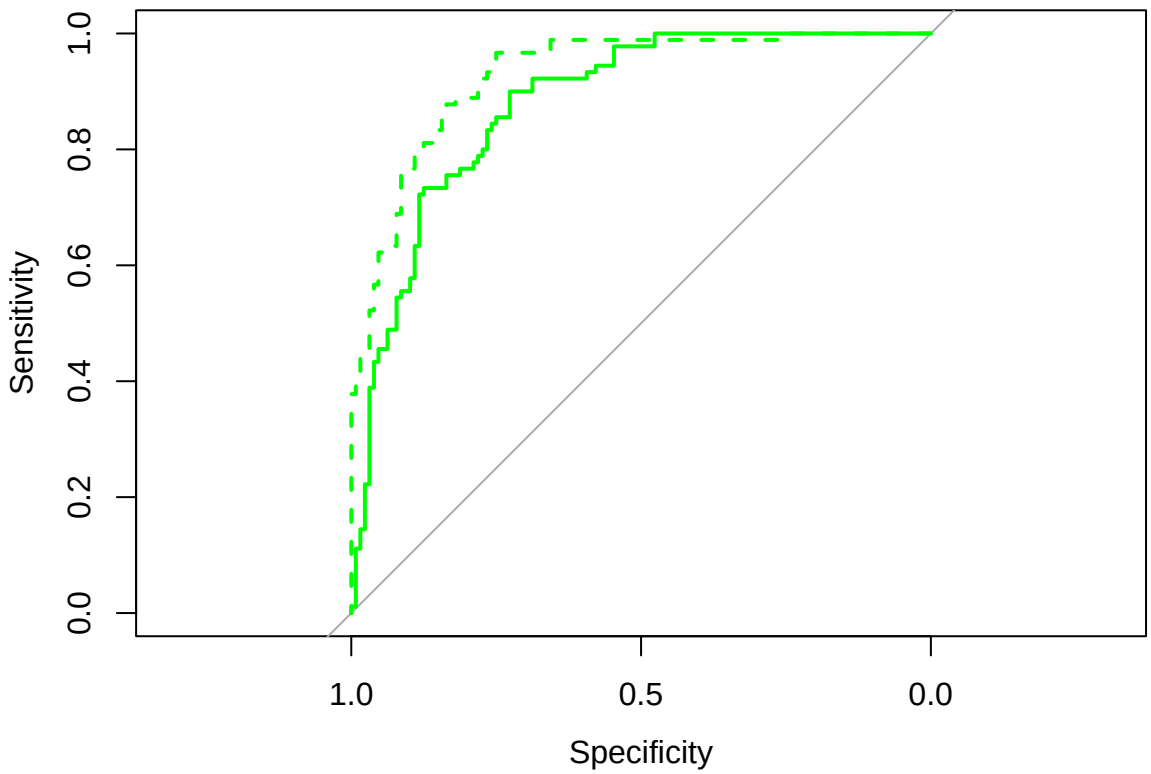
We plot the ROC curve (based on the pignistic probability), and compare it to that of the EKNN classifier:

```
curve4<-roc(y.test,test$m[,2]+test$m[,3]/2)
```

```
## Setting levels: control = 1, case = 2
```

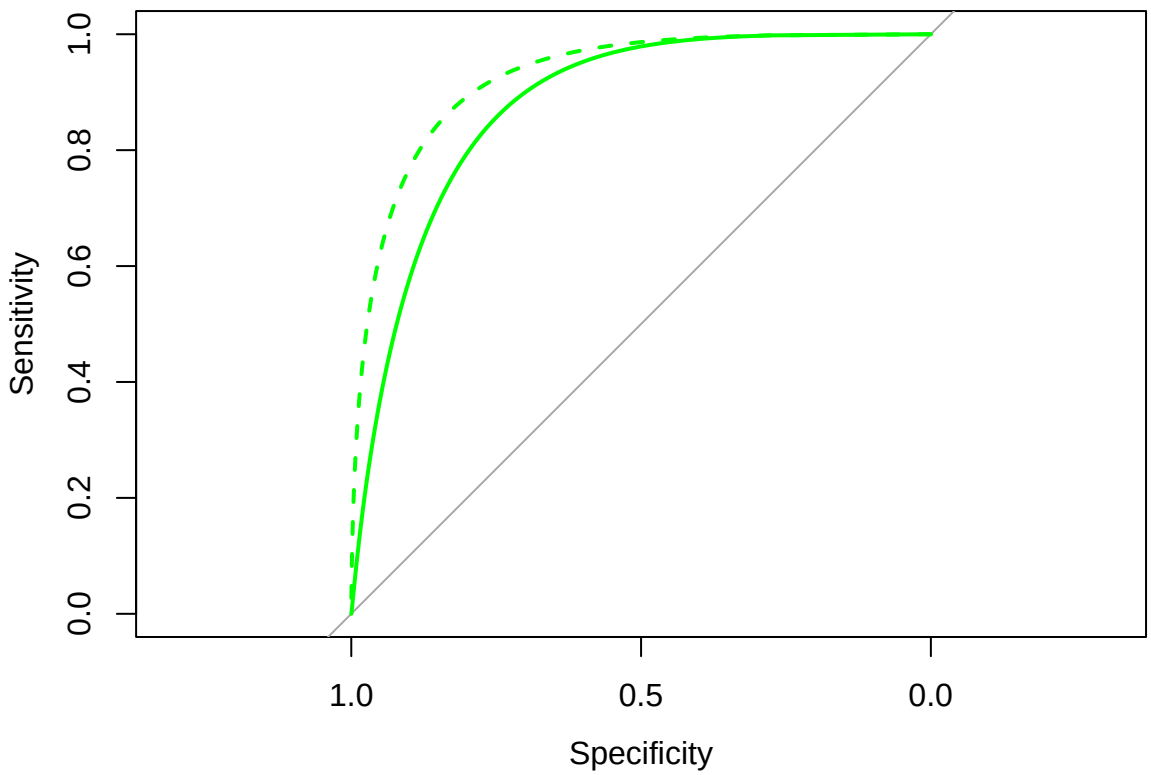
```
## Setting direction: controls < cases
```

```
plot(curve3,col="green")
plot(curve4,add=TRUE,col="green",lty=2)
```



The ENNreg classifier clearly outperforms EKNN. This can also be seen by plotting the smoothed ROC curves:

```
plot(smooth(curve3),col="green")
plot(smooth(curve4),add=TRUE,col="green",lty=2)
```



Question 4

Let us start with a 0-1 loss matrix:

```
L1 <- matrix(c(0,1,1,0),2,2)
print(L1)
```

```
##      [,1] [,2]
## [1,]    0    1
## [2,]    1    0
```

With this loss, matrix, we can compute the decisions using, e.g., the pignistic criterion (the other criteria will give the same results), and compute the confusion matrix:

```
d1<-decision(test$m,L1,rule="pignistic")
table(y.test,d1)
```

```
##       d1
## y.test  1  2
##      1 99 29
##      2  8 82
```

The true positive rate (TPR) is 82/(82+8)=0.91 and the false positive rate (FPR) is 29/(99+29)=0.23. Let us now assume that the loss of classifying a negative example as positive is half that of classifying a positive example as negative, and let us recompute the resulting decision matrix:

```
L2<-matrix(c(0,1,0.5,0),2,2)
print(L2)
```

```
##      [,1] [,2]
## [1,]    0  0.5
## [2,]    1  0.0
```

```
d2<-decision(test$m,L2,rule="pignistic")
table(y.test,d2)
```

```
##       d2
## y.test  1  2
##      1 93 35
##      2  3 87
```

The TPR has increased to 87/(87+3)=0.97, but the FPR has also increased to 35/(93+35)=0.27.

Question 5

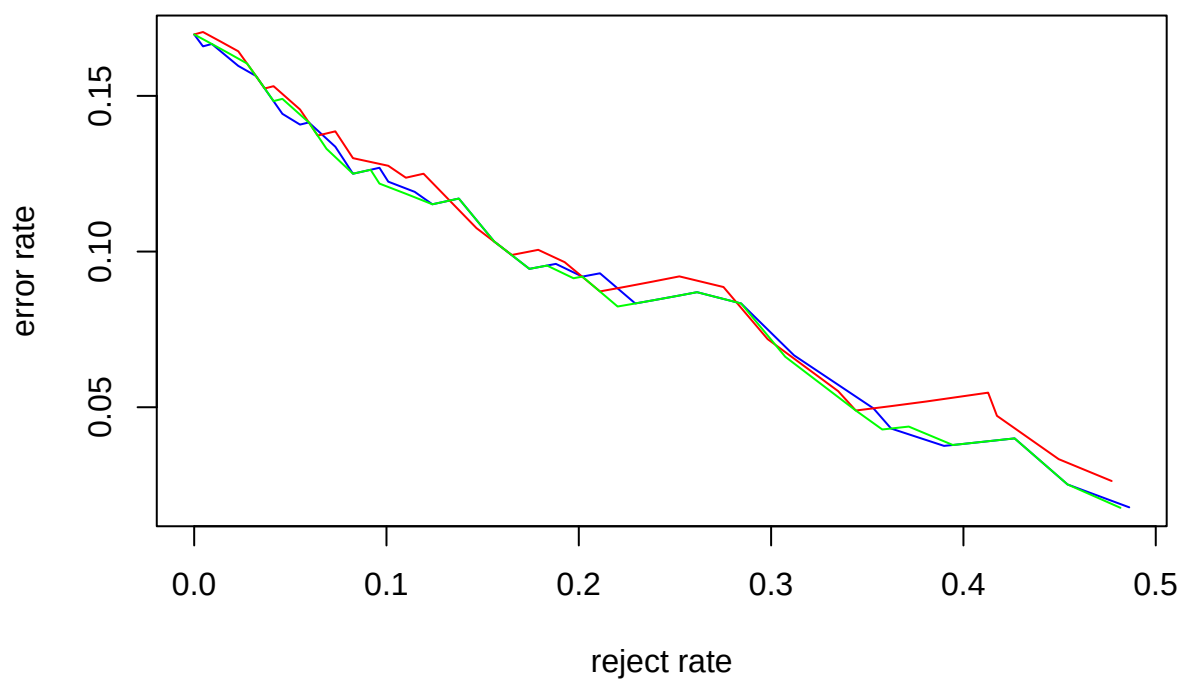
To implement classification with a reject option, we consider 3 acts: assignment to class 1, assignment to class 2, and rejection. The loss matrix passed to function `decision` will have 2 rows and 3 columns:

$$L = \begin{pmatrix} 0 & 1 & \lambda_0 \\ 1 & 0 & \lambda_0 \end{pmatrix}$$

where λ_0 is the loss of rejection. By varying λ_0, we will obtain different error and rejection rates:

```
l0 <- seq(0.1,1,0.01)
N <- length(l0)
nt <-length(y.test)
err1 <- rep(0,N)
err2 <- err1
err3 <- err1
```

```r
rej1 <- rep(0,N)
rej2 <- rej1
rej3 <- rej1
for(i in 1:N){
  L <- matrix(c(0,1,1,0,l0[i],l0[i]),2,3)
  d1 <- decision(test$m,L,rule="upper")
  rej1[i] <- length(which(d1==3))/nt
  ndec<-length(which(d1!=3))
  if(ndec==0) err1[i] <- 0 else
  err1[i] <- length(which((d1!=3)&(d1!=y.test)))/ndec
  d2 <- decision(test$m,L,rule="lower")
  rej2[i] <- length(which(d2==3))/nt
  ndec<-length(which(d2!=3))
  if(ndec==0) err2[i] <- 0 else
    err2[i] <- length(which((d2!=3)&(d2!=y.test)))/ndec
  d3 <- decision(test$m,L,rule="pignistic")
  rej3[i] <- length(which(d3==3))/nt
  ndec<-length(which(d3!=3))
  if(ndec==0) err3[i] <- 0 else
    err3[i] <- length(which((d3!=3)&(d3!=y.test)))/ndec
}
plot(rej1,err1,type="l",col="blue",xlab="reject rate",ylab="error rate")
lines(rej2,err2,col="red")
lines(rej3,err3,col="green")
```



We can see that the thre decision methods have roughly the same performances.