Evidential Classification

Thierry Denoeux

2023-08-10

Transportation mode

Question 1

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

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

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

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

```r
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 intro training and test sets:

```r
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:

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

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

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

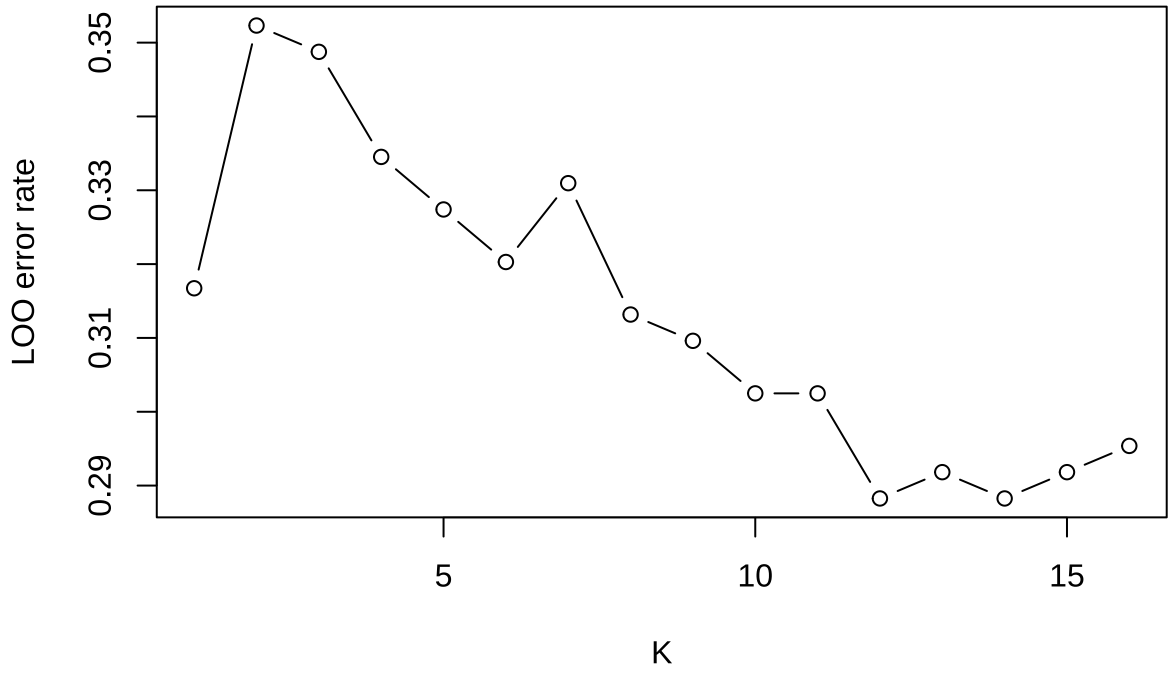
```
table(as.numeric(y.test),test$ypred)
```

```
##
##      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-on-out (LOO) cross-validation:

```
K <- seq(5:20)
N <- length(K)
err<-rep(0,N)
i<-0
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:

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

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

```
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
```

```
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:

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

```
## [1]  1.0000000  0.3031168 10.0000000
## [1] 11.0000000  0.2857723  3.4974446
## [1] 21.0000000  0.2555994  1.2391578
## [1] 31.0000000  0.2283450  0.4988377
## [1] 41.0000000  0.2092642  0.1893629
## [1] 51.00000000  0.19210266  0.08312631
## [1] 61.00000000  0.18430039  0.03767964
## [1] 71.00000000  0.18054868  0.01717119
## [1] 81.000000000  0.178689825  0.007966855
## [1] 91.000000000  0.178176184  0.003709229
## [1] 1.010000e+02 1.772777e-01 1.770324e-03
```

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

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

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

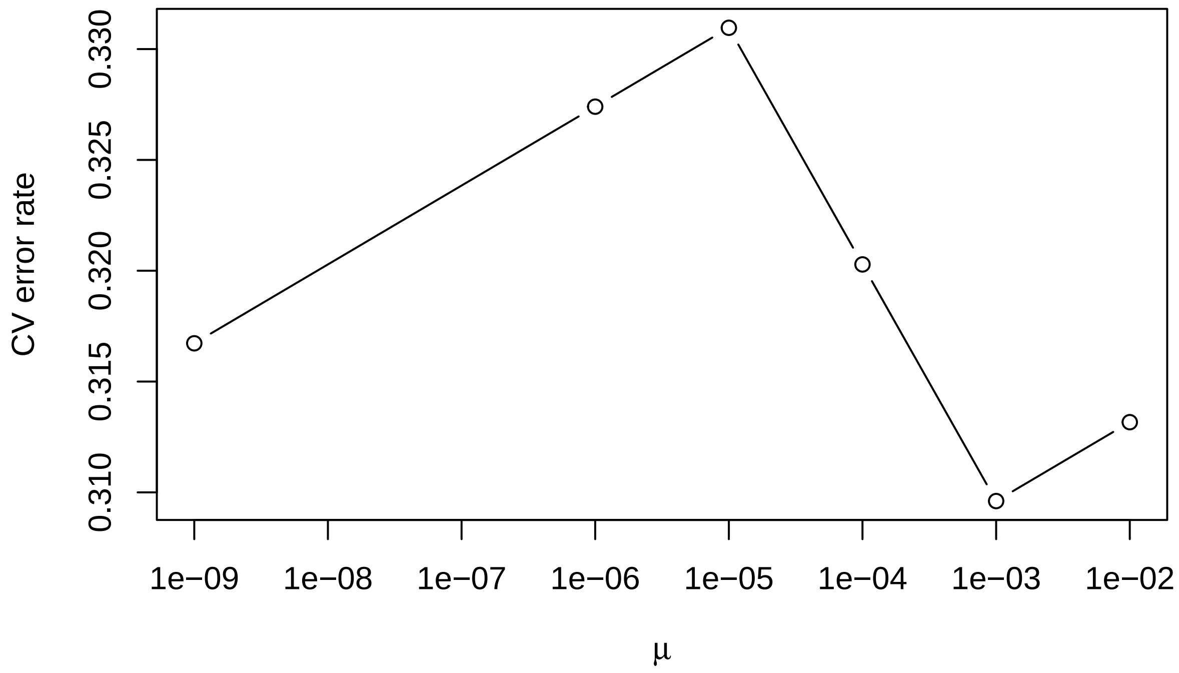
```
##
## y.test  1  2  3
##      1 63  3  3
##      2 10 12 10
##      3  8  5 26
```

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.

```
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 on the test data:

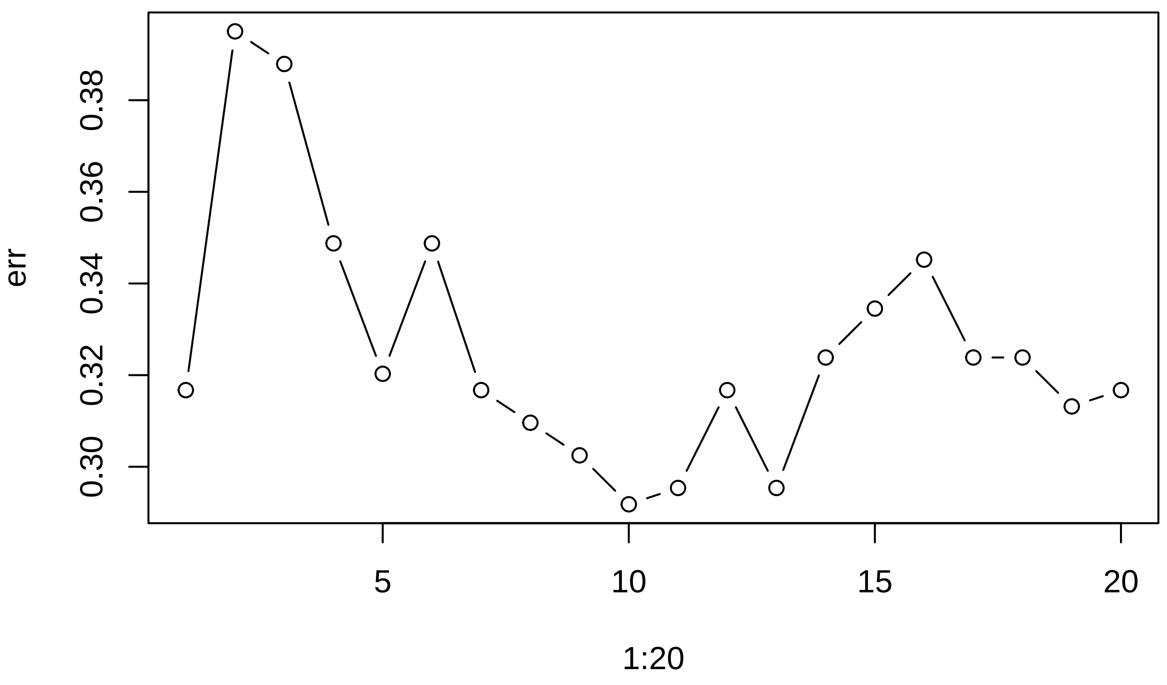
```
muopt <- MU[which.min(CV)]
param0 <- proDSinit(x.train, y.train, nproto=nproto)
fit <- proDSfit(x.train, y.train, param=param0,mu=muopt,
                options=options)
test <- proDSval(x.test,fit$param,y.test)
test$err
```

```
## [1] 0.2571429
```

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:

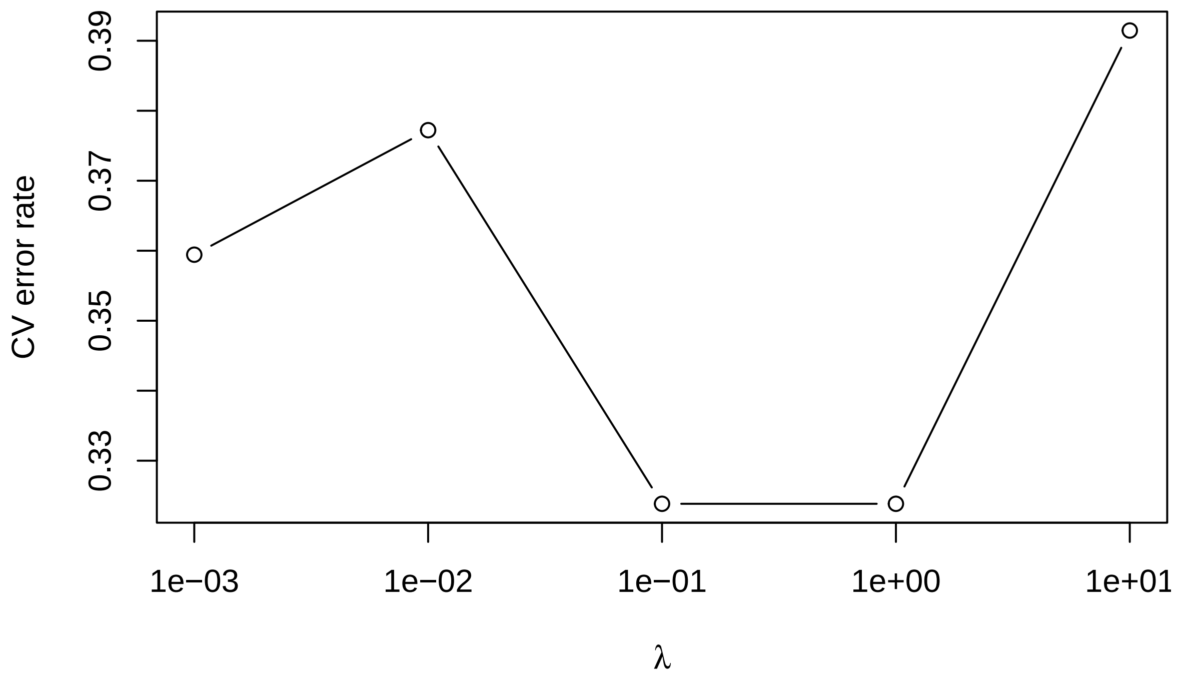
```
library(nnet)
Id <- diag(3)
Y.train <- Id[y.train,]
fit <- nnet(x.train,Y.train,entropy=TRUE,softmax=TRUE,maxit=1000,size=5,trace=FALSE)
pred <- predict(fit,newdata=x.test)
ypred <- max.col(pred)
err<-mean(ypred!=y.test)
```

```r
print(err)
```

```
## [1] 0.35
```

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

```r
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:

```r
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] 0.3071429
```