Sparse SVM procedure

Sungmin Ji

2023-12-07

```
## Select the optimal model using Minimum CV rule
### model without interaction of perturbation

lambda_seq <- cv_folds$lambda
lambda_ind <- which.min(cv_folds$cvm)
lambda <- cv_folds$lambda[lambda_ind]; cvm_min <- min(cv_folds$cvm)
lambda;cvm_min
```

```
## [1] 0.01940544
```

```
## [1] 0.02692884
```

```
lasso_optim <- sparseSVM(train_mat, Ytr, alpha=1, lambda=lambda_seq)
test_fit <- predict(lasso_optim, test_mat, lambda=lambda)
## test error
1-sum(diag(table(test_fit, Yte)))/length(Yte)
```

```
## [1] 0.04932735
```

```
##Elastic net
num_alpha = 20
num_lambda = 1e2

## Parallel computing
if("doParallel" %in% rownames(installed.packages()) == FALSE)
{install.packages("doParallel")
}
library("doParallel")
```

```
## Warning: package 'doParallel' was built under R version 4.3.2
```

```
## Loading required package: foreach
```

```
## Loading required package: iterators
```

```
## Loading required package: parallel
```

```r
if("parallel" %in% rownames(installed.packages()) == FALSE)
{install.packages("parallel")
}
library("parallel")

K=10 ## Number of folds CV
for_svmcv <- function(alpha_i){
  temp=svmcv(train_mat, Ytr, alpha=alpha_i, folds=K, lambda.min = 0.0005, seeds=1)
  return(c(temp$cvm,temp$cv_se,temp$lambda))
}

alpha_seq <- seq(0.05, 1, length=num_alpha)

ncores = 5
cl = makeCluster(ncores)
registerDoParallel(cl)

cverror_alpha_mat <- foreach(i=1:length(alpha_seq), .combine = rbind,
                                   .packages=c("sparseSVM", "Matrix")) %dopar% {
  alpha_i = alpha_seq[i]
  temp = for_svmcv(alpha_i)
  return(temp)
                                   }

lambda_seq <- round(cverror_alpha_mat[1,(1+2*num_lambda):(3*num_lambda)],5)
rownames(cverror_alpha_mat) <- paste0("a=", alpha_seq)
cverror <- cverror_alpha_mat[,1:num_lambda]
cvse <- cverror_alpha_mat[,(1+num_lambda):(2*num_lambda)]
colnames(cverror) <- paste0("l=", lambda_seq)
colnames(cvse) <- paste0("l=", lambda_seq)


stopCluster(cl)

apply(cverror, 1, min)
```

```
##     a=0.05      a=0.1     a=0.15      a=0.2     a=0.25      a=0.3     a=0.35
## 0.02692884 0.02803995 0.02581773 0.02803995 0.02806492 0.02694132 0.02694132
##      a=0.4     a=0.45      a=0.5     a=0.55      a=0.6     a=0.65      a=0.7
## 0.02581773 0.02694132 0.02694132 0.02581773 0.02578027 0.02690387 0.02806492
##     a=0.75      a=0.8     a=0.85      a=0.9     a=0.95        a=1
## 0.02916355 0.02805243 0.02803995 0.02803995 0.02692884 0.02692884
```

```r
min_ind <- apply(cverror, 1, which.min)
alpha_ind <- which.min(apply(cverror, 1, min))
alpha_optim <- alpha_seq[alpha_ind]
lambda_optim <- lambda_seq[min_ind][alpha_ind]
alpha_optim; lambda_optim
```

```
## [1] 0.6
```

```
## [1] 0.00058
```

```r
### SVM with elastic net penalty
ela_optim <- sparseSVM(train_mat, Ytr, alpha=alpha_optim, lambda=lambda_seq)
test_fit <- predict(ela_optim, test_mat, lambda=lambda_optim)

## test error
confusion <- table(test_fit, Yte)
1-sum(diag(table(test_fit, Yte)))/length(Yte)
```

```
## [1] 0.06278027
```

```r
sum(diag(table(test_fit, Yte)))/length(Yte)
```

```
## [1] 0.9372197
```

```r
## Confusion matrix
confusion
```

```
##          Yte
## test_fit  -1   1
##       -1 115  10
##       1    4  94
```

```r
## the sensitivity, TPR
confusion[2,2]/sum(confusion[,2])
```

```
## [1] 0.9038462
```

```r
## the specificity, TNR
confusion[1,1]/sum(confusion[,1])
```

```
## [1] 0.9663866
```