SVMMaj - Binary SVM

Lucas van Montfort, Tomas Miskov, Valentino Hägg, Ali Abedi

10/01/2022

1. Introduction

2. Data

3. Method

3.1 Support Vector Machine

In a binary classification problem, one of the well-known models is the support vector machine (SVM). A (non)linear combination of predictors is used to predict the class of the target variable. In order to do so, the SVM algorithm constructs a hyperplane that acts as a separator of the two classes. In a linear SVM, such hyperplane is built in the m-dimensional space where m is the number of input features. In its non-linear counterpart, input features are fist projected onto a higher dimensional space, where, if the projection is chosen wisely, the constructed hyperplane acts as a better binary classifier. This hyperplane is then projected back into the original m-dimensional space where it creates a non-linear classification boundary between the two classes (Vapnik, 2000).

The linear combination of input variables in the SVM classifier can be represented as:

$$q_i = c + \sum_{j=1}^{m} x_{ij} w_j = c + \mathbf{x}_i^\top \mathbf{w}$$

,in which c is an intercept, $\mathbf{x}_i$ is ith row of the matrix of predictor variables (X_{n*m}), and w is the $m*1$ vector of weights used to make a linear combination of the regressors. In essence, q_i becomes a latent variable based on which the category of the target variable will be determined. A latent variable is a variable that's not directly observed but that can be inferred from our input features.

The error for each prediction in SVM is considered as:

$$\xi_i = max(0, 1 - y_i q_i)$$

, where q_i is the linear combination of the regressors and y_i is +1 or -1, corresponding to the actual class. This error is called the absolute hinge error. The loss function for the SVM is expressed as a combination of the error term and a ridge penalty term:

$$L_{\text{SVM}}(c, \mathbf{w}) = \sum_{i=1}^{n} \max\left(0, 1 - y_i q_i\right) + \lambda \mathbf{w}'\mathbf{w}.$$

Here, λ is the penalty parameter limiting the size of the coefficients $\mathbf{w}$. Overall, the model parameters will be determined by the minimization procedure for the loss function.

Generally we can expand the loss function into:

$$
\begin{aligned}
L_{\mathrm{SVM}}(c, \mathbf{w}) &= \sum_{i \in G_{-1}} \max(0, q_i + 1) + \sum_{i \in G_{+1}} \max(0, 1 - q_i) + \quad \lambda \mathbf{w}^\top \mathbf{w} \\
&= \text{ Class } (+1) \text{ errors } \quad + \text{ Class } (-1) \text{ errors } \quad + \quad \text{Penalty for nonzero } \mathbf{w} \\
&= \sum_{i \in G_1} f_1(q_i) \quad + \sum_{i \in G_{-1}} f_{-1}(q_i) \quad + \quad \lambda \mathbf{w}' \mathbf{w}.
\end{aligned}
$$

Separating the error of prediction from the ridge penalty allows for application of different error functions. In addition to the absolute hinge error, SVM can also use for example a quadratic hinge error $max(0, 1 - y_i q_i)^2$, which is a smooth variation on the original absolute hinge.

3.2 Majorizing Algorithm for SVM

Although the quadratic approach for finding the optimized parameters by minimizing L_SVMis used commonly, its computational costs are high; especially with big data sets. Because the loss function in question is convex, applying an iterative majorization (IM) method can guarantee a solution close enough to the optimum global minimum (Groenen, Nalbantov & Bioch, 2009).

In each iteration of the IM procedure, we first try to find a quadratic majorizing function ($g(x, y) = ax^2 - 2bx + c$) of the hinge error. Then, we sum all the hinge errors and calculate the loss function by adding the ridge penalty term to the sum of errors. Finally, we try to update the weights to minimize the loss function. We continue this procedure until reaching our desired level of precission. The majorization function for two different hinge error functions are given in Table 1.

Error Type	Original f	a	b	c
Absolute	$f_{-1}(x) = \max(0, x + 1)$ $f_{+1}(x) = \max(0, 1 - x)$	$(4\lvert y + 1\rvert)^{-1}$ $(4\lvert 1 - y\rvert)^{-1}$	$-(a + 1/4)$ $(a + 1/4)$	$a + 1/2 + \lvert y + 1\rvert/4$ $a + 1/2 + \lvert 1 - y\rvert/4$
Quadratic	$f_{-1}(x) = \max(0, x + 1)$	1	$\begin{cases} y & \text{if } y \le -1 \\ -1 & \text{if } y > -1 \end{cases}$	$\begin{cases} 1 - 2(y+1) + (y+1)^2 & \text{if } y \le -1 \\ 1 & \text{if } y > -1 \end{cases}$
	$f_{+1}(x) = \max(0, 1 - x)$	1	$\begin{cases} 1 & \text{if } y \le 1 \\ y & \text{if } y > 1 \end{cases}$	$\begin{cases} 1 & \text{if } y \le 1 \\ 1 - 2(1-y) + (y-1)^2 & \text{if } y > 1 \end{cases}$

Overall, steps for SVM-Majorization algorithm are expressed in Algorithm 1 (Groenen, Nalbantov & Bioch (2009)).

Algorithm 1: SVM-Majorization algorithm

Choose with some initial $c^{(0)} \in \mathbb{R}^1$ and $\mathbf{w}^{(0)} \in \mathbb{R}^p$
Matrix $\mathbf{X}$ is $n \times (p+1)$ and has $\mathbf{1}$ as first column
Compute $L_{\text{SVM}}^{(0)} = L_{\text{SVM}}\left(c^{(0)}, \mathbf{w}^{(0)}\right)$
Set $k \leftarrow 1$
while $k = 1$ *or* $\left(L_{SVM}^{(k-1)} - L_{SVM}^{(k)}\right) / L_{SVM}^{(k-1)} > \epsilon$ **do**
    $k \leftarrow k + 1$
    Compute $\boldsymbol{A}$ with elements a_{ii} depending on the chosen hinge
    Compute $\mathbf{b}$ with elements b_i depending on the chosen hinge
    Update $\mathbf{v}^+$solves the linear system $\left(\mathbf{X}^\top \mathbf{A}\mathbf{X} + \lambda \mathbf{P}\right)\mathbf{v} = \mathbf{X}^\top \mathbf{b}$
    Set $\left[c^{(k)}, \mathbf{w}^{(k)^\top}\right] = \left(\mathbf{v}^+\right)^\top$
    As a check, print $k, L_{\text{SVM}}^{(k)}$, and $L_{\text{SVM}}^{(k-1)} - L_{\text{SVM}}^{(k)}$
end

$\mathbf{A}$ $n \times n$ diagonal matrix with elements $\{a_i\}$
$\mathbf{b}$ is an n-vector with elements b_i
$c_m = \sum_{i=1}^{n} c_i$
$\mathbf{P}$ is an identity matrix with $p_{11} = 0$, $\mathbf{P} = \begin{bmatrix} 0 & \mathbf{0}^\top \\ \mathbf{0} & \mathbf{I} \end{bmatrix}$

4. Results

In this section we will discuss the results of the SVM Maj algorithm and we will compare the performance of the implemented SVM Maj algorithm to the SVM Maj algorithm as seen in the standard library. The goal is to predict y, i.e. whether the client subscribed a term deposit. The size of the training-set is 2000, whereas the size of the test-set is 1000. We find that for $\lambda \in \{1, 5, 10\}$, and hinge losses 'absolute' and 'quadratic', we obtain results as follows in Table 1, Table 2 and Table 3 in the Appendix.

For $\lambda = 1$, we find that the implemented version of the algorithm slightly outperforms the library version in terms of hit rate when the hinge loss of interest is absolute. Furthermore, the implemented version converges faster than the library version, as it merely takes the implemented version 60 iterations while the library takes 285 iterations to converge. Notice, however, that the value of the loss function of the implemented algorithm is slightly larger than the library's algorithm. When the hinge loss is quadratic, we find that the hit rate of the implemented version is similar to the library's one, however, the algorithm of the package seems to converge quicker, as well as the fact that the value of the loss function is slightly less for the algorithm of the package vs. the implemented algorithm. Note that for $\lambda \in \{5, 10\}$ we observe similar results as for $\lambda = 1$.

In general, notice that the quadratic hinge loss, even though it has a larger value for the loss function than the absolute loss, converges much quicker to the steady state. Also keep in mind that in order to minimize the loss, finding the optimal penalty-parameter λ can be found through (K-fold) cross-validation. Noticeable is that the intercept coefficient estimated by the library is 0.00. This might be due to the fact that the package automatically adds an constant to the regressors.

5. Conclusion

References:

Groenen, P. J., Nalbantov, G., & Bioch, J. C. (2008). SVM-Maj: a majorization approach to linear support vector machines with different hinge errors. Advances in data analysis and classification, 2(1), 17-43.

Vapnik, V., & Chapelle, O. (2000). Bounds on error expectation for support vector machines. Neural computation, 12(9), 2013-2036.

6. Appendix

Table 1: Performance measures of the implemented SVM Maj algorithm vs. the SVM Maj of the standard package using absolute or quadratic hinge losses for $\lambda = 1$.

	Loss function type	Loss function	Hit rate	Misclassification rate	Iterations
SVM Maj (impl.)	abs.	345.368	0.908	0.092	60
SVM Maj (impl.)	quadr.	435.8463	0.91	0.09	64
SVM Maj (lib)	abs.	345.3117	0.906	0.094	285
SVM Maj (lib)	quadr.	435.8276	0.91	0.09	42

Table 2: Performance measures of the implemented SVM Maj algorithm vs. the SVM Maj of the standard package using absolute or quadratic hinge losses for $\lambda = 5$.

	Loss function type	Loss function	Hit rate	Misclassification rate	Iterations
SVM Maj (impl.)	abs.	350.2345	0.912	0.088	53
SVM Maj (impl.)	quadr.	437.1179	0.91	0.09	63
SVM Maj (lib)	abs.	350.1478	0.911	0.089	126
SVM Maj (lib)	quadr.	437.1011	0.91	0.09	41

Table 3: Performance measures of the implemented SVM Maj algorithm vs. the SVM Maj of the standard package using absolute or quadratic hinge losses for $\lambda = 10$.

	Loss function type	Loss function	Hit rate	Misclassification rate	Iterations
SVM Maj (impl.)	abs.	354.2543	0.91	0.09	80
SVM Maj (impl.)	quadr.	438.6325	0.911	0.089	61
SVM Maj (lib)	abs.	354.1733	0.908	0.092	149
SVM Maj (lib)	quadr.	438.6151	0.911	0.089	40

7. Code

```
#------
# Imports
#------
rm(list=ls())
if (!require("pacman")) install.packages("pacman")
if (!require("dplyr")) install.packages("dplyr")
if (!require("caret")) install.packages("caret")
pacman::p_load(SVMMaj)
source("SVMMaj_lib.R")
#------
# Functions
```

```r
#------
#' Performance of the predictions
#'
#' Compute the performance of the predictions
#'
#' @param vY Vector of outcome values (2 categories, -1 & 1) of the observed class
#' @param vYHat Vector of outcome values (2 categories, -1 & 1) of the predicted class

fPredPerf <- function(vY, vYHat){
  lConMatrix <- confusionMatrix(data = factor(vYHat),
                                reference = factor(vY),
                                positive = "1")
  dMissClassRate <- sum(vY != vYHat)/length(vY)
  dHitRate <- 1-dMissClassRate

  return(list("ConMatrix" = lConMatrix, "MissClassRate" = dMissClassRate, "HitRate" = dHitRate))
}

#' Print the performance of the SVMMaj package and the own implementation
#'
#' Print the performance of the SVMMaj package and the own implementation, and
#' compare the results of both methods.
#'
#' @param lPredPerfImplement List of performance indicators of the implementation
#' @param lPredPerfPackage List of performance indicators of the SVMMaj package
#' @param modelPackage Model estimated with the SVMMaj package
#' @param sHinge String indicating the type of hinge error
#' @param modelImplement Model results of the implementation

fPrintPredPerf <- function(lPredPerfImplement, lPredPerfPackage, modelPackage, sHinge, modelImplement){
  cat("===== Estimation properties SVMMaj package ===== \n")
  print(lPredPerfPackage)
  cat("=== Estimation properties own implementation === \n")
  cat("Confusion matrix:")
  print(lPredPerfImplement$ConMatrix)
  cat(".\n")
  cat("Misclassification rate: " , lPredPerfImplement$MissClassRate, ".\n")
  cat("Hit rate: " , lPredPerfImplement$HitRate, ".\n")
  cat("============ Comparison methods (implementation vs package) ==============\n")
  cat("Coefficients: ")
  dfResults <- data.frame(implementation = round(modelImplement$v,5), package = round(modelPackage$beta
  print(dfResults)
  cat("Iterations: ", modelImplement$iter, modelPackage$iteration,".\n")
  cat("Loss function: ", modelImplement$endLoss, modelPackage$loss,".\n")
  cat("Type of loss function: ", sHinge,".\n")
}

#' Prepare the data
#'
#' Clean up the data, by the addition of dummies and scaling of the columns.
#' Obtain a train and a test set.
#'
#' @param dfData Dataframe containing the data
```

```r
#' @param vInd Vector of indices of the sample
#' @param iNTrain Int indicating the size of the training set

fPrepData <- function(dfData, vInd, iNTrain){
  dfData <- dfData[vInd,]
  dfData <- select(dfData, -c('emp.var.rate', 'euribor3m'))

  vY <- ifelse(dfData$y == 'yes', 1, -1)
  mX <- model.matrix(y ~., data = dfData)

  mX <- mX[,!colnames(mX) %in% c('educationilliterate', 'defaultyes', 'monthdec')]
  mXTrain <- mX[1:iNTrain,]
  mXTest <- mX[(iNTrain+1):length(vInd),]
  mXTrainStd <- scale(mXTrain, center = TRUE, scale = TRUE)
  mXTestStd <- scale(mXTest, center = attr(mXTrainStd,"scaled:center"),
                     scale = attr(mXTrainStd,"scaled:scale"))
  mXTrainStd[,1] = 1
  mXTestStd[,1] = 1

  return(list('XTrain' = mXTrainStd, 'YTrain' = vY[1:iNTrain],
              'XTest' = mXTestStd, 'YTest' = vY[(iNTrain+1):length(vInd)]))
}
#------------------------------------------------------------------------------
# Main
#------------------------------------------------------------------------------
#------
# Magic numbers
#------
dLambda <- 1
dEpsilon <- 10^-5
bSilent <- TRUE
iNTrain <- 2000
iNTest <- 1000
iSeed <- 235167
sHinge <- 'quadratic'                                  # 'absolute' or 'quadratic'
#------
# Initialisation
#------
set.seed(iSeed)
load('bank.RData')
vInd <- sample(1:nrow(bank),size = iNTrain + iNTest, replace = FALSE)
lData <- fPrepData(bank, vInd, iNTrain)
#--------------
# Estimation
#--------------
#--------------
# SVMMaj package
#--------------
modelPackage <- svmmaj(lData$XTrain, lData$YTrain, hinge = sHinge,
                       convergence = dEpsilon, lambda = dLambda, scale = "none",verbose = TRUE )
lPredPerfPackage <- predict(modelPackage, lData$XTest, lData$YTest)
#--------------------
# Our implementation
```

```r
#-------------------
modelImplement <- MajSVM(lData$XTrain, lData$YTrain, dEpsilon = dEpsilon, sHinge = sHinge, bSilent = FA
vQPred <- lData$XTest %*% modelImplement$v
vYPred <- ifelse(vQPred > 0, 1, -1)
lPredPerfImplement <- fPredPerf(lData$YTest, as.numeric(vYPred))
#--------------
# Output
#--------------
fPrintPredPerf(lPredPerfImplement, lPredPerfPackage, modelPackage, sHinge, modelImplement)
```

```r
# Name: SVMMaj_lib
# Author: "Lucas van Montfort, Tomas Miskov, Valentino Hägg, Ali Abedi"
# Date: 08/01/2022
# Version: 1.0
# Purpose: Functions librarY for minimizing SVM loss using majorization
#-------------------------------------------------------------------------

#' SVM Loss Function
#'
#' Compute the SVM loss function
#'
#' @param mX Matrix of regressors including a column of 1s
#' @param vY Vector of outcome values (2 categories, -1 & 1)
#' @param vV Vector of weights (+ a constant)
#' @param dLambda Ridge penalization parameter lambda
LossSVM <- function(mX, vY, vV, dLambda){
  vQ <- mX %*% vV
  dLoss <- sum(pmax(0, 1 - vY * vQ)) + dLambda * (t(vV[-1]) %*% vV[-1])
  return(dLoss)
}

#' SVM Quadratic Loss Function
#'
#' Compute the SVM loss function for quadratic hinge
#'
#' @param mX Matrix of regressors including a column of 1s
#' @param vY Vector of outcome values (2 categories, -1 & 1)
#' @param vV Vector of weights (+ a constant)
#' @param dLambda Ridge penalization parameter lambda
LossSVMQuadr <- function(mX, vY, vV, dLambda){
  vQ <- mX %*% vV
  dLoss <- sum(pmax(0, 1 - vY * vQ)**2) + dLambda * (t(vV[-1]) %*% vV[-1])
  return(dLoss)
}
#' SVM majorization
#'
#' Solve the primal SVM problem using majorization
#'
#' @param mX Matrix of regressors including a column of 1s
#' @param vY Vector of outcome values (2 categories, -1 & 1)
#' @param dC Initial value of the constant
#' @param vW Vector of weights
#' @param dLambda Ridge penalization parameter lambda
```

```r
#' @param dEpsilon Accuracy parameter
#' @param sHinge Type of hinge error (absolute or quadratic)
#' @param bSilent Boolean argument, if FALSE, function prints the iterations
#' @export
MajSVM <- function(mX, vY, dC = 1, vW = rep(1, ncol(mX) - 1),
                   dLambda = 1, dEpsilon = 10^(-6),
                   sHinge = 'absolute', bSilent = TRUE){
  iN <- length(vY)
  iP <- ncol(mX)
  vV0 <- c(dC, vW)
  dStartL <- LossSVM(mX, vY, vV0, dLambda)          #starting SVM loss value
  mP <- diag(iP)
  mP[1,1] <- 0

  if(sHinge == 'quadratic'){
    mZ <- solve(t(mX) %*% mX + dLambda * mP) %*% t(mX)
    dStartL <- LossSVMQuadr(mX, vY, vV0, dLambda)          #starting SVM loss value
  }

  dL0 <- dStartL
  dDecrease <- 0
  iK <- 1
  while((iK == 1) | dDecrease > dEpsilon){
    vQ <- mX %*% vV0

    if(sHinge == 'absolute'){
      vA <- ifelse(abs(1 - vY * vQ) > dEpsilon, 1/(4 * abs(1 - vY * vQ)), 1/(4*dEpsilon))
      vB <- vY * (vA + 0.25)
      mA <- diag(as.vector(vA))
      vV1 <- solve(t(mX) %*% mA %*% mX  + dLambda * mP, t(mX) %*% vB)
      dL1 <- LossSVM(mX, vY, vV1, dLambda)
    }

    if(sHinge == 'quadratic'){
      vB <- ifelse((vY == -1 & vQ > -1) | (vY == 1 & vQ < 1), vY * 1, vQ)
      vV1 <- mZ %*% vB
      dL1 <- LossSVMQuadr(mX, vY, vV1, dLambda)
    }

    dDecrease <- (dL0 - dL1)/dL0

    if (bSilent == FALSE) {
      cat("Iteration: ", iK, "Loss SVM: ", dL1,
          "Relative difference loss SVM: ", dDecrease,"\n")
    }
    dL0 <- dL1
    vV0 <- vV1
    iK <- iK + 1
  }
  return(list('v' = vV0, 'startLoss' = dStartL, 'endLoss' = dL0,
              'iter' = iK))
}
```