Supervised Machine Learning HW 4

Group 3: Maja Nordfeldt, Jakob Rauch, Timo Schenk, Konstantin Sommer

25-1-2020

Introduction

For a variety of organizations, ranging from businesses to NGO's, cold-calling is a key component driving sales, recruitment and promotion. The payoffs of such labour-intensive efforts may vary greatly however, and figuring out who to target or not - and how - is of great interest for telemarketers all over the world. In this paper we aim to identify what predictors that explain telemarketers outcome representing a Portugese retail bank, and how well we can predict whether a call will be successful or not. In particular, we aim to predict whether a client subscribed a term deposit. We do so by employing a support vector machine approach, suitable for the large set of predictors and binary outcome of each observation. Although results being case-specific, the general findings may be of interests for all sectors.

Data

The data utilized is originally provided by Moro et al. (2014) and is available to retrieve online[1] http://www3.dsi.uminho.pt/pcortez/bib/2014-dss.txt]], and covers the telemarketing phone calls selling long-term deposits of a Portugese retail bank. It includes 4119 clients are a sample of 10% of the original sample.

The binary outcome variable y, represents whether the called client subscribed to a term deposit and takes the value "yes" or "no". We transform this variable to -1 for "no" and 1 for "yes". The dataset is not balanced, as a majority of the outcomes are "no".

The predictor variables include characeristics of the telemarketing, the product sold and the client targeted. Additionally, predictors covering economic and social circumstances retrieved from the central bank of the Portuguese Republic[2] are included.

More precisely, client attributes include *age*, *job* - being "admin.","blue-collar","entrepreneur","housemaid","management","retir
employed","services","student","technician","unemployed" or "unknown", *marital* representing marital status of "divorced" (divorced or widowed),"married","single" or "unknown", *education* representing "basic.4y","basic.6y","basic.9y","high.school","illiterate","professional.course","university.degree"or "unknown", $default$ being having credit in default - "no","yes" or "unknown", *housing* being having housing loan - "no","yes" or "unknown" as well as *loan* - having a personal loan - "no","yes" or "unknown". We find that there are no observations in education illiterate, but model matrix command creates dummy with NANs for it.

Further, variables related with the last contact of the current campaign incude *contact* communication type - "cellular" or "telephone", *month* denoting last contact month of year and day_of_week being last contact day of the week.

Other attributes captured are *campaign* - number of contacts performed during this campaign and for this client, *pdays*: number of days that passed by after the client was last contacted from a previous campaign, *previous*: number of contacts performed before this campaign and for this client *poutcome*: outcome of the previous marketing campaign - "failure","nonexistent" or "success".

Further, social and economic context attributes include *cons.price.idx* as consumer price index and *nr.employed*, the number of employees by a quarterly indicator.

[1] Available at: [pdf] http://dx.doi.org/10.1016/j.dss.2014.03.001 bib

[2] Available at https://www.bportugal.pt/estatisticasweb

We exclude a few variables - *euribor*3*m* being theeuribor 3 month rate and *emp.var.rate* being a quaterly indicator of employment variation rate because they both contain numerical and date entries. Further, we drop variable *duration* being last contact duration as it is always 0 when the outcome is *no*. All categorical variables are transformed to dummies, so we end up with $p = 50$ predictors in addition to a constant. All variables are scaled to z-scores. Finally, we draw a random sample of 1000 observations for computational speed.

Method

```
## Warning: package 'SVMMaj' was built under R version 3.5.3
```

The question that we are asking is a classification type question. We are trying to classify based on some available characterisitics if a customer will chose yes or no. We therefore use a method called support vector machine. Its purpose is to predict in a linear regression (can also be extended to nonlinear) type estimation the value of the endogeneous variable, in our case y, if the client subscribed a term deposit. In order to that we are maximizing a loss function, that minimizes the likelihood of being far of the right solution, so if clients with a predicted value that is far from the actual decission is predicted wrong, that influences the loss heavier than closer ones. Additionally we include a penalizing term, that in this case does not only full fill a parameter shrinkage function, but also determines the width of an intervall in which we will not give a prediction about the choice of the client.\ This loss function that we are aiming to minimize is given by

$$L_{Quad-SVM(c;w)} = \sum_{i \in G_{-1}} \max(0; q_i + 1)^2 + \sum_{i \in G_1} \max(0; 1 - q_i)^2 + \lambda w^T w$$

where q_i is the predicted value from $c + x_i^T w$, with x_i the observation of individual i, c the constant and w the weights or coefficient vector. G_{-1} and G_1 describe the set of individuals for which y is either -1 or 1 so that an observation counts towards the loss function for only of the two terms.\ The decision rule is then when a predicted value is smaller than 0 it will be predicted as $y = -1$ and for $q_i > 0$ vice versa. In the loss function however, values in between minus and plus 1 do still count towards the error, since only the ones that are either below minus one or above one are here considered as being correct and all other count towards the loss. The width of this interval is determined by the size of the coefficients since they determine the value of q_i. This is where the penalty term comes into play, so by chosing different values for λ one can determine how much the parameters are shrinked and therefore how large the described interval is. The intuition for the two maxima functions are that an observation either adds zero to the loss function, if it would be predicted correctly or it adds a number that is increasing in the distance to the actual correct value, -1 or 1. Such that values that are only predicted slightly wrong, but will still be classified incorrectly, in the loss function, counts less towards the loss than an observation that is preicted far of, but still gets classified the same as the one that is only a bit of.\ One can instead of minimizing over the squared maximum terms also maximizize over the absolute values or over what is known as a Hinge error term. The advantage of the squared expression is, that there must be a blobal minimum, since the Loss function is strictly convex and that there exists algorithms that can find this minimum in a reasonable time. We are using the so called majoorizing function algorithm for it.\ In order to find the optimal value of the hyperparameter, we are again relying on the k-fold cross validation. In contrast to other typical regression cases one must now chose a different measure of loss instead of the RMSE. We decided to base our deccission based on the λ that minimizes the misclassification rate (defined as $1 - Hitrate$), which is defined as one minues the sum over all correctly specified observations divided by the number of observations. This loss definition has a straight forawrd intuition since one wants to maximize the number of correctly predicted values.

- alternative kernels used

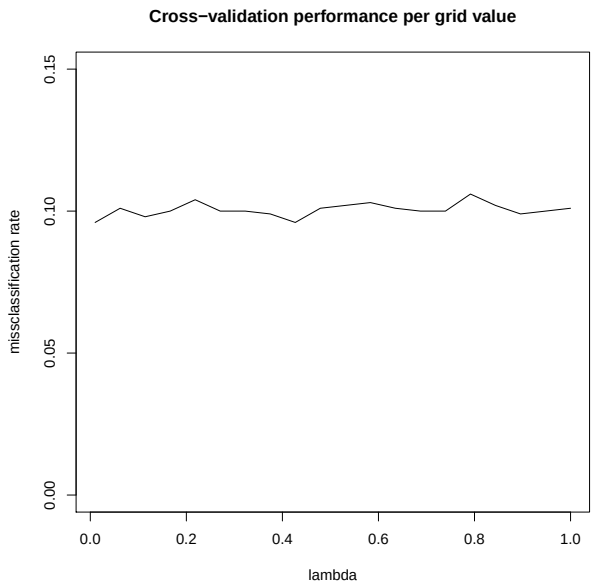

Figure 1: Cross-validating the penalty term

Results

Optimal: $\lambda = 0.427$ with out-of-sample misclassification of 9.6%. Now check most important predictors and in-sample diagnostics for the optimal lambda:

contacttelephone	monthaug	monthjun	cons.conf.idx	nr.employed
-0.2002	-0.1339	0.1221	0.2028	-0.15

```
##     ypred
## y     -1   1
##   -1 880  13
##   1   75  32
```

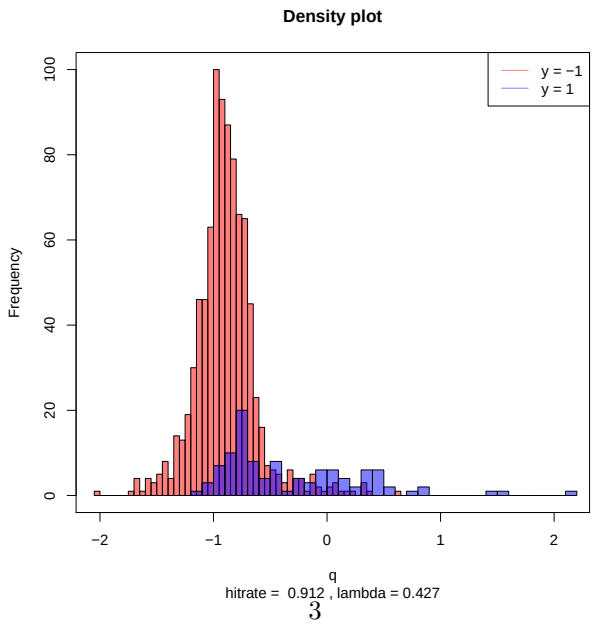

Figure 2: Density plot of in-sample predictions

Future research could focus on predicting succesful outcomes, as this is of great business interest. It may also use different hinge functions and examine whether results vary, as we in this paper focus on the quadratic one. Further, we restrict the analyiss to a sample of 1000 observations due to limited computational capability. Using the full dataset and comparing the outcomes would be another interesting future application.

References

Moro et al., 2014] S. Moro, P. Cortez and P. Rita. A Data-Driven Approach to Predict the Success of Bank Telemarketing. Decision Support Systems, In press, http://dx.doi.org/10.1016/j.dss.2014.03.001

Code

```
knitr::opts_chunk$set(echo = F)
set.seed(123456)
library(SVMMaj)

### functions

## SVM loss
loss.svm.quad= function(y,q,w,lambda){
  # returns SVM loss given weights, predicted qs
  grp0err = sum(((q+1)[(y==-1)&(q>-1)])^2)
  grp1err = sum(((1-q)[(y==1)&(q<1)])^2)
  return(grp0err+grp1err+lambda*sum(w^2))
}



## quadratic hinge -- notation as in slides
v.update = function(y,X,vold,lambdaP){
  # returns updated v+
  qtilde = X%*%vold
  b = qtilde*((qtilde<=-1)&(y==-1)) - ((qtilde>-1)&(y==-1)) +
    ((qtilde<=1)&(y==1)) + qtilde*((qtilde>1)&(y==1))
  # A = diag(1,nrow(X)) # skip for computational simplicity
  vnew = solve(t(X)%*%X + lambdaP)%*%t(X)%*%b

  return(vnew)
}

## SVMMaj MM algorithm

SVMMaj_manual = function(y,X,lambda,v0,epsilon = 1e-8){
  # returns optimal v = (c,w)' given data y, X, penalty lambda
  # by running the MM algorithm with initial level v0 and stopping criterion
  # epsilon

  lambdaP = diag(lambda,ncol(X))
  lambdaP[1,1] = 0
```

```
  Lold = loss.svm.quad(y,q = X%*%v0,w = v0[-1],lambda)
  #cat('LO =',Lold,"\n")
  k = 1
  Lnew = Lold
  v = v0
  while(k==1|((Lold-Lnew)/Lold)>epsilon) {
    k = k+1
    Lold = Lnew
    vnew = v.update(y,X,v,lambdaP)
    Lnew = loss.svm.quad(y,q = X%*%vnew,w = vnew[-1],lambda)
    v = vnew
    #cat('after iteration ',k," L = ",Lnew,"\n")
  }
  #cat("stopping crit achieved\n")
  return(vnew)
}
### k-fold
k_fold_crossval = function(y,X,k,lambda,v0,reshuffled_indices){
  # Function for k-fold crossvalidation, returns RMSE for given alpha and lambda
  # and shuffled indices

  n = length(y) # Number of observations
  p = (dim(X)[2]) # Number of parameters

  # Reshuffle data and split into k groups of equal size
  reshuffled_indices = sample(1:n, n, replace=FALSE) # Shuffle indices of our n observations

  n_test = floor(n/k) # n=77 and k=10 would give 7 obs. in the test set the rest of the observations
  # in the training set. For n=77, using k=11 or k=7 is advisable to get evenly sized groups.

  loss=rep(0,k) # Vector to hold the loss for each fold

  # Loop over the k folds and save MSEs
  for (i in 1:k){

    # Divide data into test and training
    y_test = y[c(reshuffled_indices[(1+(i-1)*n_test):(i*n_test)])]
    y_train = y[-c(reshuffled_indices[(1+(i-1)*n_test):(i*n_test)])]
    X_test = X[c(reshuffled_indices[(1+(i-1)*n_test):(i*n_test)]),]
    X_train = X[-c(reshuffled_indices[(1+(i-1)*n_test):(i*n_test)]),]

    beta = SVMMaj_manual(y_train,X_train,lambda,v0) # get beta estimates for training data from majoriz
    fitted_val = (X_test)%*%beta # get qhat for test data
    TP=sum(fitted_val>0&y_test==1)
    TN=sum(fitted_val<0&y_test==(-1))
    loss[i]=1-(TP+TN)/n_test # save average missclassification for test data
    cat('.')
  }
  loss=mean(loss) ##take average over found losses
  cat('mean loss for lambda =',lambda,':',loss)
  cat('\n')

  return(loss)
```

```
}

min_loss = function(y,X,k=10,lambda_values=10^seq(-3, 4, length.out = 10),v0){
  # Tunes hyperparameters lambda via k-fold crossvalidation. Returns optimal lambda
  # as well as the resulting loss

  loss = array(0,length(lambda_values)) # create vector to hold loss for each lambda

  # Loop over hyperparameter combinations
  for (j in 1:length(lambda_values)){
    # cat('lambda = ',lambda_values[j])
    loss[j] = k_fold_crossval(y,X,k,lambda_values[j],v0) # Fill the matrix with losses
  }


  min_index = which.min(loss) # Find index of lowest loss
  lambda_min = lambda_values[min_index] # lambda value at lowest loss
  loss_min = loss[min_index] # lowest misclassification error

  return(list(loss,lambda_min,loss_min))
}
set.seed(123456)
library(SVMMaj)
library("pander") # for tables

# load data
load('./bank.RData')
smp = sample(1:dim(bank)[1],size = 1000,replace = F)
bank = bank[smp,]
y = 2*as.numeric(bank$y)-3   # scale to -1,1 instead of 1,2

# transform categorical variables to dummies
# omit euribor3m, weird entries.
X = model.matrix(y~.-euribor3m -emp.var.rate -duration -1 ,data = bank)
# no observations in education illiterate, but model matrix command creates dummy with NANs for it:

X = scale(X)
idx.missing = which(colSums(is.na(X))>0) # identify index of column containing NAns
X = X[,-idx.missing]
X = cbind(1,X)

# linear SVM
p = ncol(X)-1

# settings
v0 = rep(0,p+1)
lambda_values=seq(0.01, 1, length.out = 20)
epsilon = 1e-8 # the default of the package
# ---- Cross Validation -----------
# manual cross validation of lambda using quadratic hinge loss
man_results=min_loss(y,X,10,lambda_values,v0) # find optimal lambda
loss_vector=man_results[[1]]
opt_lambda=man_results[[2]]
```

```r
pdf('cvalplot.pdf')
plot(lambda_values,loss_vector,type='l',
     ylim=c(0,0.15),xlab='lambda',ylab='missclassification rate',
     main='Cross-validation performance per grid value') # plot average missclassification for differen
dev.off()
# compare cross validation to package
pack_results=svmmajcrossval(X,y,ngroup=10,search.grid = list(lambda=lambda_values)) ##Find optimal lamb
# cbind(opt_lambda,pack_results$param.opt)
knitr::include_graphics('./cvalplot.pdf')
# ----  confusion matrix and hitrate ----
lambda_compare = 0.427 # or set to opt_lambda from CV
vhat=SVMMaj_manual(y,X,lambda_compare,v0,epsilon) # manual estimates
pander((vhat[rank(-abs(vhat))<=6,])[-1],
       caption='largest 5 coefficients') # largest 5 coefficients except constant
q = X%*%vhat
ypred = 1*(q>0) -1*(q<0)
table(y,ypred)

hitrate = mean(ypred==y)
# ---- density plot ----
qlim = c(min(q),max(q))
cols = c(rgb(1,0,0,0.5),rgb(0,0,1,0.5))
pdf('densityplot.pdf')
hist(q[y==-1],breaks = 50, col =cols[1],
     main="Density plot",
     sub= paste("hitrate = ",hitrate,', lambda =',round(lambda_compare,3)),
     xlim=qlim,xlab='q')
hist(q[y==1],breaks = 30, col =cols[2],add=T)
legend('topright',legend = c('y = -1','y = 1'),
       col = cols, lty=1)
box()
dev.off()
knitr::include_graphics('./densityplot.pdf')

# ---- Compare with package for a fixed value of lambda ----
pack_vhat=svmmaj(X[,-1],y,lambda = lambda_compare,hinge="quadratic",scale="none") # get coefficients fo
print(cbind(vhat[-1],pack_vhat$beta)) # compare our solution to package
print(mean(abs(vhat[-1]-pack_vhat$beta))) # mean absolute deviation of the estimates
sum(ypred!=((pack_vhat$q>0)-(pack_vhat$q<0))) # predictions are the same

# Results different Kernels
library(kernlab)
pack_vhat_rbf <- svmmaj(X,y,lambda = lambda_compare,hinge="quadratic",
                        scale="none", kernel=rbfdot, kernel.sigma=1)  # RBF kernel, sigma = 1
pack_vhat_poly <- svmmaj(X,y,lambda = lambda_compare,hinge="quadratic",
                         scale="none", kernel=polydot,kernel.degree=2,
                         kernel.offset =1,kernel.scale=1) # Poly kernel degree 2, added term = 1
pack_vhat_laplace <- svmmaj(X,y,lambda = lambda_compare,hinge="quadratic",
                            scale="none", kernel=laplacedot,kernel.sigma=1)  # Laplace kernel sigma = 1

# Comparing kernels performance
summary(pack_vhat_rbf)[19]
summary(pack_vhat_poly)[19]
```

```r
summary(pack_vhat_laplace)[19]
```