Exercises of SVM classifier

Eva Cernadas
FMLCV Course
CITIUS: Centro de Tecnoloxías Intelixentes da USC
Universidade de Santiago de Compostela

12 de diciembre de 2023

We practice the use of Support Vector Machine (SVM) classifier on the datasets `wine.data` (with 2 classes) and `hepatitis.data` (with 3 classes). We use the functionality provided by several libraries:

1. `LibSVM` library[1], accessible from C++, Octave/Matlab, Python, Weka/Java.
2. Class `SVC` in the `svm` module of Python `scikit-learn`[2].
3. Function `ksvm` in the `kernlab` package in programming language `R`[3]

The tasks can be done using some of these programming languages (`R`, `octave`, `Matlab`, `C++` or `Python`). To use the LibSVM from `octave` or `matlab`, do the following steps: 1) download the library from the web page; and 3) compile the library using the command `make`. If you use `octave`, go to the folder `matlab`, enter in `octave` and run `make`. Then, exit `octave`. The main functions of `libsvm` are:

1. `svm= svmtrain(c, x, opt)`, where the input argument `x` is the matrix with the patterns of the training set, `c` is the desired output of the patterns of the training set and `opt` is a string with the configuration options of the SVM (see the `libsvm README` file). This function returns the SVM trained.
2. `z=svmpredict(tc, tx, svm)`, where the input argument `tx` is the matrix with the test patterns, `tc` is a vector with the desired output of the test patterns and `svm` is the trained SVM. This function returns the predicted output for the test patterns.

The code to apply linear and radial SVM to a classification dataset using the whole set to train and test the SVM (this evaluation methodology is not normally used due to

[1] https://www.csie.ntu.edu.tw/∼cjlin/libsvm/
[2] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
[3] https://www.rdocumentation.org/packages/kernlab/versions/0.9-29/topics/ksvm

provide very optimistic results, but it is only to know the use of the functions `svmtrain()` and `svmpredict()`). Note that the data must be pre-processed to be mean 0 and standard desviation 1.

```
clear all;
warning off all
addpath("libsvm-3.24/matlab")
% 3 classes
%dataset='wine';x=load('wine.data');
% 2 classes
dataset='hepatitis';x=load('hepatitis.data');
c=x(:,1);x(:,1)=[];[N,I]=size(x);
cl=unique(c);C=numel(cl);
% preprocessing: mean 0, desviation 1
mx=mean(x); stdx=std(x);
x=bsxfun(@rdivide,bsxfun(@minus,x,mx),stdx);
% x=(x-mean(x))./std(x);  #matlab
% SVM with linear kernel
% s is the SVM type (0 for classification)
% t is the kernel type (0 for linear and 2 for radial)
% c is the tuned parameter lambda
% g is the tuned parameter kernel spread
opt=sprintf('-s 0 -t 0 -c %g -q',100);
svm=svmtrain(c,x,opt);
y=svmpredict(c,x,svm);
[kappa, accu, cm]=evaluate(c, y, C);
disp('Confusion matrix=');disp(cm);
fprintf('SVM lineal: dataset %s: accuracy=%.2f%%\n',dataset,accu)
fprintf('SVM lineal: dataset %s: kappa=%.2f%%\n',dataset, kappa)
% SVM with radial base (Gaussian) kernel
opt=sprintf('-s 0 -t 2 -c %g -g %g -q',100,1/I);
svm=svmtrain(c,x,opt);
y=svmpredict(c,x,svm);
[kappa, accu, cm]=evaluate(c, y, C);
disp('Confusion matrix=');disp(cm);
fprintf('SVM radial: dataset %s: accuracy=%.2f%%\n',dataset,accu)
fprintf('SVM radial: dataset %s: kappa=%.2f%%\n',dataset, kappa)
```

Use the linear SVM using cross-validation with 4 folds. In this case, the lambda parameter must be tuned using the validation set. The functions `standarized()` and `normalize()` are used to standarized the data to be mean zero and standard desviation 1:

```
% standarized: return the mean, standard desviation of data x and
% the data x with mean 0 and standard desviation 1.
% x : a matrix of number of patterns by number of inputs
```

```
function [mx, stdx, x]=standarized(x)
  % preprocessing: mean 0, desviation 1
  mx=mean(x); stdx=std(x);
  x=bsxfun(@rdivide,bsxfun(@minus,x,mx),stdx);
  % x=(x-mean(x))./std(x);  #matlab
end
```

```
% normalize: return the data x normalized
% inputs : the data x and the mean and std to normalize
function x=normalize(x, meant, stdt)
  x=bsxfun(@rdivide,bsxfun(@minus,x,meant),stdt);
  % x=(x-meant)./stdt;  #matlab
end
```

The code to apply linear SVM using cross-validation is provided:

```
clear all;more off
warning off all
addpath("libsvm-3.24/matlab")
% 3 classes
% dataset='wine';x=load('wine.data'); % first column is the
    output
% 2 classes
dataset='hepatitis';x=load('hepatitis.data'); % first column is
    the output
c=x(:,1);x(:,1)=[];[N,I]=size(x);
cl=unique(c);C=numel(cl);
K=4 % number of folds
[tx,tc,vx,vc,sx,sc]=createFolds(x, c, K);
vL= 2.^(-5:2:15);nL=length(vL); % lambda values
best_kappa=-100;bestL=100;
for l=1:nL
  L=vL(l);
  for i=1:K
    opt=sprintf('-s 0 -t 0 -c %g -q',L);
    [mx, stdx, x]=standarized(tx{i});
    svm=svmtrain(tc{i},x,opt);
    xv=normalize(vx{i}, mx, stdx);
    y=svmpredict(vc{i},xv,svm);
    [kappa(i), acc(i)]=evaluate(vc{i}, y, C);
  end
  kappa_mean=mean(kappa);acc_mean=mean(acc);
  fprintf('lambda=%.1g: ',L);
  fprintf('kappa=%.1f%% accuracy=%.1f%%\n',kappa_mean,acc_mean)
```

```
    if kappa_mean>best_kappa
      best_kappa=kappa_mean;
      bestL=L;
    end
end
printf('best_config='); fprintf('Lambda= %g\n',bestL);
cmt=zeros(C); % confusion matrix
kappa=zeros(1,K); acc=zeros(1,K);
for i=1:K
    opt=sprintf('-s 0 -t 0 -c %g -q -b 1',bestL);
    [mx, stdx, x]=standarized([tx{i}; vx{i}]);
    svm=svmtrain([tc{i}; vc{i}],x,opt);
    xv=normalize(sx{i}, mx, stdx);
    y=svmpredict(sc{i},xv,svm);
    [kappa(i), acc(i), cm]=evaluate(sc{i}, y, C);
    fprintf('fold %i: kappa=%.1f%% accuracy=%.1f%%\n',i,kappa(i),
        acc(i))
    cmt = cmt + cm;
end
kappa_mean=mean(kappa); acc_mean=mean(acc); cmt=cmt/K;
disp('Final confusion matrix='); disp(cmt);
fprintf('dataset %s: kappa=%.1f%% accuracy=%.1f%%\n',dataset,
    kappa_mean,acc_mean)
```

The code to apply radial SVM using cross-validation is also provided. In this case the tuned parameters are the regularization parameter λ and the kernel spread σ:

```
clear all; more off
warning off all
addpath("libsvm-3.24/matlab")
% 3 classes
dataset='wine'; x=load('wine.data'); % first column is the output
% 2 classes
% dataset='hepatitis'; x=load('hepatitis.data'); % first column is
    the output
c=x(:,1); x(:,1)=[]; [N,I]=size(x);
cl=unique(c); C=numel(cl);
K=4 % number of folds
[tx,tc,vx,vc,sx,sc]=createFolds(x, c, K);
vL= 2.^(-5:2:15); nL=length(vL); % lambda values
vG=2.^(-7:2:7); nG=length(vG); % kernel spread values
best_kappa=-100; bestL=100; bestG=0;
for l=1:nL
  L=vL(l);
```

```
17    for j=1:nG
18      G=vG(j);
19      for i=1:K
20        opt=sprintf('-s 0 -t 2 -c %g -g %g -q',L,G);
21        [mx, stdx, x]=standarized(tx{i});
22        svm=svmtrain(tc{i},x,opt);
23        xv=normalize(vx{i}, mx, stdx);
24        y=svmpredict(vc{i},xv,svm);
25        [kappa(i), acc(i)]=evaluate(vc{i}, y, C);
26      end
27      kappa_mean=mean(kappa);acc_mean=mean(acc);
28      fprintf('lambda=%.1g, radial=%.1g: ',L,G);
29      fprintf('kappa=%.1f%% accuracy=%.1f%%\n',kappa_mean,acc_mean)
30      if kappa_mean>best_kappa
31        best_kappa=kappa_mean;
32        bestL=L; bestG=G;
33      end
34    end
35  end
36  printf('best_config='); fprintf('Lambda= %g, Radial spread=%g\n',
      bestL, bestG);
37  cmt=zeros(C); % confusion matrix
38  kappa=zeros(1,K);acc=zeros(1,K);
39  for i=1:K
40    opt=sprintf('-s 0 -t 2 -c %g -g %g -q',bestL,bestG);
41    [mx, stdx, x]=standarized([tx{i}; vx{i}]);
42    svm=svmtrain([tc{i}; vc{i}],x,opt);
43    xv=normalize(sx{i}, mx, stdx);
44    y=svmpredict(sc{i},xv,svm);
45    [kappa(i), acc(i), cm]=evaluate(sc{i}, y, C);
46    fprintf('fold %i: kappa=%.1f%% accuracy=%.1f%%\n',i,kappa(i),
        acc(i))
47    cmt = cmt + cm;
48  end
49  kappa_mean=mean(kappa);acc_mean=mean(acc);cmt=cmt/K;
50  disp('Final confusion matrix='); disp(cmt);
51  fprintf('dataset %s: kappa=%.1f%% accuracy=%.1f%%\n',dataset,
      kappa_mean,acc_mean)
```

1. Programas en Python

1. Use the following code to create a program `svc.py` that implements SVC using the object `SVC` of `sklearn.svm`, tuning the λ and $\gamma = 1/2\sigma^2$ hyper-parameters using 4-fold cross-validation with the `createFolds()` function used with ANN:

```
model=SVC(C=L,kernel='rbf',gamma=G,verbose=False).fit(tx[k],
    ty[k])
z=model.predict(vx[k])
```

The whole program is:

```
from numpy import *
from sklearn.svm import *
from sklearn.metrics import *

dataset='wine';  # hepatitis (2 class), wine (3 class)
nf='%s.data'%dataset;x=loadtxt(nf)
y=x[:,0]-1;x=delete(x,0,1);C=len(unique(y))
print('SVC dataset %s'%dataset)

def createFolds(x,y,K):
        from numpy.random import shuffle,seed
        seed(100)
        [N,n]=x.shape;C=len(unique(y));ntf=K-2;nvf=1
        ti=[[]]*K;vi=[[]]*K;si=[[]]*K
        for i in range(C):
                t=where(y==i)[0];npc=len(t);shuffle(t)
                npf=int(npc/K);ntp=npf*ntf
                nvp=npf*nvf;nsp=npc-ntp-nvp;start=0
                for k in range(K):
                        p=start;u=[]
                        for l in range(ntp):
                                u.append(t[p]);p=(p+1)%npc
                        ti[k]=ti[k]+u;u=[]
                        for l in range(nvp):
                                u.append(t[p]);p=(p+1)%npc
                        vi[k]=vi[k]+u;u=[]
                        for l in range(nsp):
                                u.append(t[p]);p=(p+1)%npc
                        si[k]=si[k]+u;start=start+npf
        tx=[];ty=[];vx=[];vy=[];sx=[];sy=[]
        for k in range(K):
                i=ti[k];tx.append(x[i,:]);ty.append(y[i])
```

```
                i=vi[k];vx.append(x[i,:]);vy.append(y[i])
                i=si[k];sx.append(x[i,:]);sy.append(y[i])
        return [tx,ty,vx,vy,sx,sy]

K=4;
tx,ty,vx,vy,sx,sy=createFolds(x,y,K)

# pre-processing: mean 0, desviation 1----------------------
for k in range(K):
        med=mean(tx[k],0);dev=std(tx[k],0)
        tx[k]=(tx[k]-med)/dev
        vx[k]=(vx[k]-med)/dev
        sx[k]=(sx[k]-med)/dev
# tuning the hyper-parameters-----------------------------
kappa_mellor=-100;kappa=zeros([1,K]);
vL=2.**arange(-5,16,2);nL=len(vL);  # regularization (lambda)
vG=2.**arange(-10,11,2);nG=len(vG); # gaussian spread (gamma)
vkappa=zeros([nL,nG]);kappa=zeros(K);kappa_mellor=-inf;
print('%10s %15s %10s %10s'%('Lambda','Gamma','Kappa','Best'))
for i in range(nL):
    L=vL[i]
    for j in range(nG):
        G=vG[j]
        for k in range(K):
            modelo=SVC(C=L,kernel='rbf',gamma=G,verbose=False).fit(tx[k],ty[k])
            z=modelo.predict(vx[k])
            kappa[k]=100*cohen_kappa_score(vy[k],z)
        kappa_med=mean(kappa);vkappa[i,j]=kappa_med
        if kappa_med>kappa_mellor:
            kappa_mellor=kappa_med;L_mellor=L;G_mellor=G
        print('%10i %15g %10.1f %10.1f'%(L,G,kappa_med,kappa_mellor))
print('L_best=%g G_best=%g kappa=%.1f%%'%(L_mellor,G_mellor,kappa_mellor))
from pylab import *
# graph with the tuning of the two hyperparameters L,G-------
figure(1);clf();u=ravel(vkappa);plot(u);grid(True)
axis([1,len(u),-5,100])
xlabel('Configuration');ylabel('Kappa (%)')
title('Kappa (%%) tuning for SVC %s'%dataset)
show()
savefig('tuning_svc_%s.eps'%dataset);show()
#3D  plot ------------------------------------------------
from mpl_toolkits.mplot3d import Axes3D
fig=figure(2);clf();ax=Axes3D(fig)
```

```
[X,Y]=meshgrid(log2(vL),log2(vG))
ax.plot_surface(X,Y,vkappa,rstride=1,cstride=1,cmap='hot')
xlabel('$log_2 \lambda$');ylabel('$log_2 \gamma$')
title('Kappa (%%) tuning SVC 3D %s'%dataset);colorbar()
show()
# mapa de calor-----------------------------------------
figure(3);clf();imshow(vkappa);colorbar()
xlabel('Regularization ($log_2 \lambda$)')
ylabel('gaussian spread ($log_2 \gamma$)')
title('Tuning SVC calor map %s'%dataset)
show()
# test--------------------------------------------------
mc=zeros([C,C])
if C==2:
        pre=zeros(K);re=zeros(K);f1=zeros(K)
for k in range(K):
        x=vstack((tx[k],vx[k]));y=concatenate((ty[k],vy[k]))
        modelo=SVC(C=L_mellor,kernel='rbf',gamma=G_mellor,\
        verbose=False).fit(x,y)
        z=modelo.predict(sx[k]);y=sy[k]
        kappa[k]=100*cohen_kappa_score(y,z)
        mc+=confusion_matrix(y,z)
        if C==2:
                pre[k]=precision_score(y,z)
                re[k]=recall_score(y,z)
                f1[k]=f1_score(y,z)
kappa_med=mean(kappa);mc/=K
print('SVC dataset=%s L=%g G=%g kappa=%.1f%%'%(dataset,\
    L_mellor,G_mellor,kappa_med))
print('Confusion matrix:'); print(mc)
```

2. Exercises to do by the students

The lab work for the students is:

1. Calculate the accuracy, Cohen kappa and confusion matrix for datasets studied (`wine`, `hepatitis`, `heart-disease-cleveland` and `annealing`) using the SVM classifier with linear kernel using the whole dataset as training and test set.

2. Calculate the accuracy, Cohen kappa and confusion matrix for all datasets using the SVM classifier with Gaussian kernel using the whole dataset as training and test set and using the default configuration for the hyper-parameters ($\lambda = 100$ and $\sigma = 1/n$,

which n is the number of inputs). Compare the performance with the SVM classifier with linear kernel.

3. Repeat the process using cross-validation with 4 folds. In this case, we must tune the hyper-parameters for the SVM with Gaussian kernel: the regularization parameter λ with values $\lambda = 2^{-5} \dots 2^{15}$ and the kernel spread σ of the Gaussian kernel, $\sigma = 2^{-7} \dots 2^{7}$. For the SVM with linear kernel, it is only need to tune the λ parameter. So, you must use the validation set to tune the hyper-parameters and select the best configuration to train the SVM with the training and validation sets and test the SVM over the test set.

4. Use the SVM classifier to the classification of one texture dataset, using 4-fold cross validation and hyper-parameter tuning. Compare the results using the SVM classifier with linear and Gaussian kernel.

5. For the problems with more than two classes, compare the results provided by the SVM using the OVO (one-versus-one) and OVA (one-versus-all) approaches.

6. **Compare classifiers**: 1) use the Wilcoxon-Signed Rank Test to compare the SVM classifier with linear and RBF kernel; and 2) use the Friedman rank and box plots to compare the classifiers studied (KNN, LDA, MLP, ELM and SVM). What classifier is the best in your experimentation?.

7. **Optional task**: calculate the performance of **Fast Support Vector Classification (FSVC)**[4] on the `arabic` dataset in the UCI Machine Learning Repository. The `arabic` dataset has 16800 patterns with 1024 atributes and 28 classes. Althought, it is quite large, it can be also processed by SVM classifier and be loaded in memory, but the code allows to process big datasets, which can be saved in different files. You can download the dataset and code from `https://doi.org/10.24433/CO.8733864.v1`. The code is in `octave/matlab` programming language.

[4] `https://doi.org/10.1109/TPAMI.2021.3085969`