Exercises of LDA classifier

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

30 de noviembre de 2023

1. Programs in Matlab

To practice the Linear Discriminant Analysis (LDA) classifier, i provide the datasets wine.data (with 3 classes) and hepatitis.data (with 2 classes), which were downloaded from the UCI machine learning repository (https://archive.ics.uci.edu/ml/datasets.php). I also share the following Matlab code of the LDA classifier:

```
% lda: implements the lda classifier
% output: y the predicted output
% inputs: x matrix with the training patterns (each pattern one
    row)
%          c vector with the desired output in training set
%          xtest matrix with the test patterns
function y=lda(x, c, xtest)
[N, I]=size(x);
mx=mean(x); stdx=std(x);
% preprocessing: mean 0, desviation 1
x=bsxfun(@rdivide,bsxfun(@minus,x,mx),stdx);
% x=(x-mean(x))./std(x);  #matlab

cl=unique(c);C=numel(cl);
nc=zeros(C,1);  % number of patterns per class
mc=zeros(C,I);  % mean of each class
S=zeros(I); % total covariance
w=zeros(C,I+1); % coeficients of LDA

for i=1:C
        j=(c==cl(i));nc(i)=sum(j);
        u=x(j,:);mc(i,:)=mean(u);
```

```
            S=S+(nc(i)-1)*cov(u)/(N-C);
end
pr=nc/N;  % probabilities
for  i=1:C
            u=mc(i,:);t=u/S;
            w(i,1)=log(pr(i))-t*u'/2;  % offset
            w(i,2:end)=t;  %  linear term
end
% standarized xtest
% preprocessing: mean 0, desviation 1
xtest=bsxfun(@rdivide,bsxfun(@minus,xtest,mx),stdx);

L=[ones(size(xtest,1),1)  xtest] * w';  % linear scores
% implement softmax function
P=exp(L)./repmat(sum(exp(L),2),[1 C]);  % class probabilities
[~,y]=max(P,[],2);  % predicted class by LDA
```

Firstly, we are going to use the lda.m function to train and test the classifier using the whole dataset. The Matlab program could be:

```
clear all;
% 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);
y=lda(x, c, x);
[kappa, accu, cm]=evaluate(c, y, C);
disp('Confusion matrix=');disp(cm);
fprintf('dataset %s: accuracy=%.2f%%\n',dataset,accu)
fprintf('dataset %s: kappa=%.2f%%\n',dataset, kappa)
```

which use the following function `evaluate()` to calculate the confusion matrix, accuracy and Cohen kappa:

```
% Return: kappa, accuraccy and confusion matrix
% Inputs:  tc (true class), pc (predicted class) and C (number of
    classes)
function [kappa,acc,cm]=evaluate(tc,pc,C)
    cm=zeros(C);np=length(tc);
        for  i=1:np
                j=tc(i);k=pc(i);cm(j,k)=cm(j,k)+1;
        end
        s=sum(sum(cm));pa=trace(cm);acc=100*pa/s;pe=0;
```

```
        for k=1:C
            pe=pe+sum(cm(k,:))*sum(cm(:,k))/s;
        end
        kappa=100*(pa-pe)/(s-pe);
end
```

Secondly, we will apply the LDA classifier using cross-validation with two dataset (training and testing sets). The validation set is not necessary because there is no hyper-parameter to tune. I also share the function code to do this operation:

```
% createFolds: create the folds for cross-validation
% Inputs: x (matrix of patterns), x (desired output) and K (
    number of folds)
% Outputs: tx matrix with training patterns(rows)
%          tc vector with the desired output for training
    patterns
%          vx, vc: idem to validation set
%          sx, sc: idem to test set
function [tx,tc,vx,vc,sx,sc]=createFolds(x, c, K)
rand('seed',0);
[N,n]=size(x); % Number of patterns and features
val=unique(c); % output values
Q=numel(val); % number of classes

for j=1:Q
        fprintf('  class %i: %i patterns\n',j,sum(c==j))
end

ntf=K-2; % Number of training folds
nvf=1;  % Number of validation folds: the number of test folds is
    K-ntf-nvf
% creation of folds
npc=zeros(1,Q); % No. Patterns per class
% ntp/nvp/nsp=no. train/valid/test patterns of each class;
% npf=no. patterns of each class per fold
ntp=zeros(1,Q);nvp=zeros(1,Q);
nsp=zeros(1,Q);npf=zeros(1,Q);
tx=cell(1,K);tc=cell(1,K);
vx=cell(1,K);vc=cell(1,K);
sx=cell(1,K);sc=cell(1,K);
for i=1:Q
  t=find(c==i);j=numel(t);npc(i)=j;k=randperm(j);
  ind=t(k); % ind=indices of patterns of each class
  npf(i)=floor(j/K);ntp(i)=ntf*npf(i);
```

```
  nvp(i)=nvf*npf(i);nsp(i)=j-ntp(i)-nvp(i);
  start=1;
  for k=1:K
    p=start;u=[];
    for l=1:ntp(i)  % indices of train patterns
      u=[u ind(p)];p=p+1;
      if p>npc(i); p=1; end
    end
    tx{k}=[tx{k};x(u,:)];tc{k}=[tc{k};c(u)];u=[];
    for l=1:nvp(i)  % indices of validation patterns
      u=[u ind(p)];p=p+1;
      if p>npc(i); p=1; end
    end
    vx{k}=[vx{k};x(u,:)];vc{k}=[vc{k};c(u)];u=[];
    for l=1:nsp(i)  % indices of test patterns
      u=[u ind(p)];p=p+1;
      if p>npc(i); p=1; end
    end
    sx{k}=[sx{k};x(u,:)];sc{k}=[sc{k};c(u)];
    start=start+npf(i);
  end
end
```

The Matlab code to use the function `createFolds()` could be:

```
clear all;
% 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);
cmt=zeros(C); % confusion matrix
kappa=zeros(1,K);acc=zeros(1,K);
for i=1:K
  ti=[tx{i}; vx{i}]; % join training and validation sets for training
  ci=[tc{i}; vc{i}]; % idem for desired output
  y=lda(ti, ci, sx{i});
  [kappa(i), acc(i), cm]=evaluate(sc{i}, y, C);
  fprintf('Confusion matrix fold %d=\n', i);disp(cm);
```

```
  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)
```

2. Programs in Python

The LDA classifier can be executed using the object `sklearn.linear_discriminant.LinearDiscriminantAnalysis` object. The training and test on the whole dataset can be executed using the following program:

```
from numpy import *
from sklearn.discriminant_analysis import *
from sklearn.metrics import *
from sklearn.model_selection import *

#dataset='wine';
dataset='hepatitis';
nf='%s.data'%dataset;x=loadtxt(nf)
y=x[:,0];x=delete(x,0,1)
# preprocessing: mean 0, desviation 1
x=(x-mean(x,0))/std(x,0)
print('LDA dataset %s:'%dataset)
#-------------------------------------------------------
# training and test on the whole dataset
#-------------------------------------------------------
lda=LinearDiscriminantAnalysis().fit(x,y)
z=lda.predict(x)
kappa=cohen_kappa_score(y,z);acc=accuracy_score(y,z)
print('Train+Test: kappa=%.1f%% accuracy=%.1f%%'\
        %(100*kappa,100*acc))
cf=confusion_matrix(y,z)
print('confusion matrix:'); print(cf)
#-------------------------------------------------------
# 4-fold cross-validation using cross_val_predict sklearn function
#-------------------------------------------------------
lda=LinearDiscriminantAnalysis()
K=4;z=cross_val_predict(lda,x,y,cv=K)
```

```
kappa=cohen_kappa_score(y,z);acc=accuracy_score(y,z)
print('%i-fold CV: kappa=%.1f%% accuracy=%.1f%%'\
        %(K,100*kappa,100*acc))
cf=confusion_matrix(y,z)
print('confusion matrix:'); print(cf)

C=len(unique(y))
if C==2:
        pre=precision_score(y,z)
        re=recall_score(y,z)
        f1=f1_score(y,z)
        print('precision=%.1f%% recall=%.1f%% f1=%.1f%%'\
           %(100*pre,100*re,100*f1))
```

In order to perform 4-fold cross-validation, the following program uses the corresponding function `createFolds()` for splitting data into train, validation and test sets:

```
from numpy import *
from sklearn.discriminant_analysis import *
from sklearn.metrics import *
from sys import exit

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

# preprocessing: mean 0, deviation 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
kappa=zeros(K);acc=zeros(K);cm=zeros([C,C])
print('%10s %10s %10s'%('k','kappa(%)','acc(%)'),end='')
if C==2:
        pre=zeros(K);re=zeros(K);f1=zeros(K)
        print('%15s %10s %10s'%('Precision(%)','Recall(%)','F1(%)'),end='')
print('')
for k in range(K):
        x=vstack((tx[k],vx[k]));y=concatenate((ty[k],vy[k]))
        modelo=LinearDiscriminantAnalysis().fit(x,y)
        z=modelo.predict(sx[k]);y=sy[k]
        kappa[k]=100*cohen_kappa_score(y,z)
        acc[k]=100*accuracy_score(y,z)
        cm+=confusion_matrix(y,z)
        print('%10i %10.2f %10.2f'%(k+1,kappa[k],acc[k]),end='')
        if C==2:
                pre[k]=100*precision_score(y,z)
                re[k]=100*recall_score(y,z)
                f1[k]=100*f1_score(y,z)
                print('%15.2f %10.2f %10.2f'%(pre[k],re[k],f1[k]),end='')
        print('')
kappa_mean=mean(kappa);acc_mean=mean(acc);cm/=K
print('kappa_mean=%.2f%% acc_mean=%.2f%%'%(kappa_mean,acc_mean),end='')
```

```
if C==2:
        pre_mean=mean(pre);re_mean=mean(re);f1_mean=mean(f1)
        print('precision_mean=%.2f%% recall_mean=%.2f%%\
       F1_mean=%.2f%%'%(pre_mean,re_mean,f1_mean))
else:
        print('')
```

3. Exercises to do by the students

The lab work for the students is:

1. Calculate the accuracy, Cohen kappa and confusion matrix for both datasets using the LDA classifier using the whole dataset as training and test set.
2. Repeat the process using cross-validation with 4 folds.
3. Implement the cross-validation using the leave-one-pattern-out approach and provide the results. In this case, the process training-test is repeated N times, each one excluding a pattern.
4. Use the LDA classifier for the classification of the textures problems of the previous week (`lbpTrain.txt`, `lbpTest.txt`; `mlbpTrain.txt`, `mlbpTest.txt`; and `haralick Train.txt` `haralickTest.txt`).
5. Compare the KNN and LDA classifiers using Wilcoxon-Signed Rank Test. You can use the `ranksum(perfClass1, perfClass2)` function in `matlab/octave` and the `wilcoxon(perfClass1, perfClass2)` function from the `scipy.stats` module of `python` (`perfClass1` and `perfClass2` are the performance measure for the classifier 1 or 2 on different datsets).
6. **Optional task**: test the LDA classifier with another classification problem from the UCI machine learning repository or an owner dataset.