Exercises of ANN classifiers

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

7 de diciembre de 2021

We practice the use of Artificial Neural Network (ANN) classifiers on the datasets `wine.data` (with 2 classes) and `hepatitis.data` (with 3 classes). For the Multi-Layer Perceptron (MLP) classifier, we use the functionality provided by three libraries:

1. `nnet` package in `Octave`[1], whose compressed file are in TEAMS, or `Matlab Neural Network Toolbox`.

2. Class `MLPClassifier` of `scikit-learn` package in `Python`[2].

3. `nnet` package in programming language `R`[3]

1. Programs in Octave

The tasks can be done using some of these programming languages (`R`, `Octave`, `Matlab` or `Python`. Using the `nnet` package of `Octave`, the important functions are:

1. `prestd()`: preprocesses the data so that the mean is 0 and the standard deviation is 1. For example: `[mTrainInputN,cMeanInput,cStdInput] = prestd(mTrainInput)`, where `mTrainInput` is the source input data used for training, `mTrainInputN` is the normalized input data, `cMeanInput` is the mean value of training input and `cStdInput` is the standard desviation of the training input.

2. `trastd`: standardizes data additional to training data for neural network simulation (e.g. the test set).

3. `newff`: creates a feed-forward backpropagation network.

[1] https://octave.sourceforge.io/nnet/
[2] https://scikit-learn.org/stable/modules/neural_networks_supervised.html
[3] https://cran.r-project.org/web/packages/nnet/index.html

4. `train`: trains a neural feed-forward network.

5. `sim`: simulates a trained neural network: `netoutput = sim (net, mInput)`, where `net` is the trained network, `nInput` is the test set and `netoutput` is the predicted output for the test set.

I provide for the lab exercices an example of use of the `nnet` package in `octave` trained and tested with the same dataset using default values (one hidden layer with 4 neurons and the output layer with the same neurons as the number of classes of the problem).

```
clear all;
warning off all
addpath("nnet-0.1.13/nnet/inst")
%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);
nHidden=4; %number of hidden neurons
nOutput=C; %number of output neurons
y=mlp(x, c, x, [nHidden, nOutput]);
[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)
```

This program uses the function `mlp()`, which is a wrapper of the functions of the `nnet` package. Its code is:

```
%mlp: implements the mlp 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
%        neurons a vector with the number of neurons of each layer( the last
%        layer is the output layer)
function y = mlp(x, c, xtest, neurons)
  mTrainInput=x';
  %normalization: 0 mean and 1 variance
  [mTrainInputN,cMeanInput,cStdInput] = prestd(mTrainInput);
  mTestInputN = trastd(xtest',cMeanInput,cStdInput);
  nl=numel(neurons);tf=cell(1,nl); %tf=transfer function
```

```
  for i=1:nl-1
    tf{i}='tansig';
  end
  tf{nl}='purelin';
  MLPnet = newff(min_max(mTrainInputN), neurons, tf, "trainlm", "learngdm", "mse");
  MLPnet.trainParam.show=inf;
  N=size(x,1);
  c2=zeros(neurons(end),N);
  for i=1:N
    j=c(i);c2(j,i)=1;
  end
  net = train(MLPnet,mTrainInputN,c2);
  y2 = sim(net,mTestInputN);
  [~,y]=max(y2);
end
```

Test the program on datasets `wine.data` and `hepatitis.data`.

Repeat the experiments using cross-validation using 4 folds. In the case of MLP classifier, use the validation set to tune the hyper-parameters: the number of hidden layers H (test the values 1, 2 and 3) and the number of neurons by layer (test the values 10, 20 and 30). The configuration of number of layers and number of neurons with the highest kappa will be use to train the MLP classifier and test with the test set The example code is:

```
clear all;more off
%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);
H=3; %number of hidden layers
IK=30; %number of neurons by layer
best_kappa=-100;best_neurons=[];
for h=1:H
  for j=10:10:IK
    neurons=[C];
    for m=1:h
      neurons=[j neurons];
    end
    for i=1:K
```

```
      y=mlp(tx{i}, tc{i}, vx{i}, neurons);
      [kappa(i), acc(i)]=evaluate(vc{i}, y, C);
    end
    kappa_mean=mean(kappa);acc_mean=mean(acc);
    printf('neurons=');printf('%d ',neurons);
    fprintf(': kappa=%.1f%% accuracy=%.1f%%\n',kappa_mean,
       acc_mean)
    if kappa_mean>best_kappa
      best_kappa=kappa_mean;best_neurons=neurons;
    end
  end
end
printf('best_config=');fprintf('%d ',best_neurons);printf('\n')
cmt=zeros(C); % confusion matrix
kappa=zeros(1,K);acc=zeros(1,K);
for i=1:K
  y=mlp([tx{i}; vx{i}], [tc{i}; vc{i}], sx{i}, best_neurons);
  [kappa(i), acc(i), cm]=evaluate(sc{i}, y, C);
  fprintf('fold %d: 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)
```

For the Extreme Learning Machine (ELM), we use a modify version of the code provided by the authors[4], due to the code of the authors assume that the inputs are normalized between -1 to +1. So, the input argumentes of this modify function `elm()` (uploaded to TEAMS) are the normalized data instead of the files with the training and testing data. The function `ELMscale()` normalizes the data:

```
% ELMscale: scale the input data between -1 and +1
% inputs: matriz of number of patters (rows) times number of
   inputs (cols)
%          (first column is the desired output, which is not
   scaled)
% output: the data scaled
function x=ELMscale(x)
  I=size(x,2); i=2:I; x2=x(:,i);
  x(:,i)=(2*x2-max(x2)-min(x2))./range(x2);
end
```

[4] https://personal.ntu.edu.sg/egbhuang/elm_random_hidden_nodes.html

and the example code to use the whole dataset as training and testing set is:

```
clear;clc
rand('seed', 0);
%dataset='hepatitis'; fn='hepatitis.data'; x=load(fn);
dataset='wine'; fn='wine.data';x=load(fn);
elm_type=1; %1=classification, 0=regression
h=50;  %no. hidden neurons
act='sig'; %sig, sin, hardlin, tribas, radbas
%input data scaled between -1 and +1
x=ELMscale(x);
[train_time,test_time,train_acc,test_acc]=elm(x,x,elm_type,h,act);
fprintf('dataset %s: train_acc=%.2f%%test_acc=%.2f%%\n',dataset,100*train_acc,100*test_acc)
```

which only calculate the accuracy as quality measure.

2. Programs in Python

The MLP classifier can be executed using the object `sklearn.neural_network.MLPClassifier` object. The 4-fold cross validation with hyper-parameter tuning of number of hidden neurons, using the same `createFolds()` function as in the LDA, can be executed using the following program:

```
from numpy import *
from sklearn.neural_network import *
from sklearn.metrics import *
#from sklearn.model_selection import *

import warnings
warnings.filterwarnings("ignore")

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

# preprocesamento: media 0, desviacion 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
vkappa=zeros(K);kappa_mellor=-Inf;
H=3     # number of hidden layers
IK=30   # number of neurons by layer
for i in range(1,H+1):
  print('%10s '%('H%i'%i),end='')
print('%10s'%'Kappa(%)')
for h in range(1,H+1):
  for j in range(10,IK+1,10):
    neurons=[C]
    for m in range(1,h+1):
      neurons.insert(0,j)
    for k in range(K):
      modelo=MLPClassifier(hidden_layer_sizes=neurons).fit(tx[k],ty[k])
      z=modelo.predict(vx[k])
      vkappa[k]=cohen_kappa_score(vy[k],z)
    kappa_med=mean(vkappa)
    for i in range(h):
      print('%10i '%(neurons[i]),end='')
    for i in range(h+1,H+1):
      print('%10s '%'',end='')
    print('%10.1f'%(100*kappa_med))
    if kappa_med>kappa_mellor:
      kappa_mellor=kappa_med;neurons_mellor=neurons
```

```
print('mellor arquitectura');print(neurons_mellor[:-1])
print('kappa=%.1f%%\n'%(100*kappa_mellor))
mc=zeros([C,C])
if C==2:
  pre=zeros(K);re=zeros(K);f1=zeros(K)
for k in range(K):
  tx[k]=vstack((tx[k],vx[k]));ty[k]=concatenate((ty[k],vy[k]))
  mx=mean(tx[k],0);stdx=std(tx[k],0);tx2=(tx[k]-mx)/stdx
  modelo=MLPClassifier(hidden_layer_sizes=neurons).fit(tx2,ty[k])
  sx2=(sx[k]-mx)/stdx
  z=modelo.predict(sx2);y=sy[k]
  vkappa[k]=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=mean(vkappa);mc/=K
print('MLP dataset=%s kappa=%.2f%%'%(dataset,100*kappa))
print('matriz de confusion:'); print(mc)
```

The MLP for regression is implemented in the following program:

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

import warnings
warnings.filterwarnings("ignore")

dataset='airfoil';
nf='%s.data'%dataset;x=loadtxt(nf)
y=x[:,0];x=delete(x,0,1)
print('MLP dataset %s'%dataset)

def crea_folds_reg(x,y,K):
  from numpy.random import shuffle,seed
  seed(100)
  [N,n]=x.shape # Number of patterns and features
  j=argsort(y);ind=arange(N);shuffle(ind)
  ntf=K-2 # Number of training folds
  nvf=1  # Number of validation folds: no. test folds is K-ntf-nvf
  # ntp/nvp/nsp=no. train/valid/test patterns of each class;
  # npf=no. patterns of each class per fold
  tx=[];ty=[]
  vx=[];vy=[]
  sx=[];sy=[]
  npf=int(N/K)  # no. patterns/fold
  ntp=int(ntf*npf)  # no. training patterns
  nvp=int(nvf*npf)  # no. validation patterns
```

```
  nsp=int(N-ntp-nvp) # no. test patterns
  start=0
  for k in range(K):
    tx.append(zeros(n));vx.append(zeros(n));sx.append(zeros(n))
    ty.append(array([],'int'))
    vy.append(array([],'int'))
    sy.append(array([],'int'))
    p=start;u=[]
    for j in range(ntp):
      u.append(p);p=(p+1)%N
    v=ind[u];tx[k]=vstack((tx[k],x[v]))
    ty[k]=append(ty[k],y[v]);u=[]
    for j in range(nvp):
      u.append(p);p=(p+1)%N
    v=ind[u];vx[k]=vstack((vx[k],x[v]))
    vy[k]=append(vy[k],y[v]);u=[]
    for j in range(nsp):
      u.append(p);p=(p+1)%N
    v=ind[u];sx[k]=vstack((sx[k],x[v]))
    sy[k]=append(sy[k],y[v]);u=[];start+=npf
    tx[k]=delete(tx[k],0,0);vx[k]=delete(vx[k],0,0);
    sx[k]=delete(sx[k],0,0)
  return [tx,ty,vx,vy,sx,sy]

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

# preprocesamento: media 0, desviacion 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
  med=mean(ty[k]);dev=std(ty[k])
  ty[k]=(ty[k]-med)/dev;vy[k]=(vy[k]-med)/dev;sy[k]=(sy[k]-med)/dev

rmse=zeros(K);rmse_mellor=Inf
H=3     # number of hidden layers
IK=30   # number of neurons by layer
for i in range(1,H+1):
  print('%10s '%('H%i'%i),end='')
print('%10s'%'RMSE')
for h in range(1,H+1):
  for j in range(10,IK+1,10):
    neurons=[1]
    for m in range(1,h+1):
      neurons.insert(0,j)
    for k in range(K):
      modelo=MLPRegressor(hidden_layer_sizes=neurons).fit(tx[k
```

```
],ty[k])
      z=modelo.predict(vx[k])
      rmse[k]=sqrt(mean((vy[k]-z)**2))
    rmse_med=mean(rmse)
    if rmse_med<rmse_mellor:
      rmse_mellor=rmse_med;best_neurons=neurons
    for i in range(h):
      print('%10i '%(neurons[i]),end='')
    for i in range(h+1,H+1):
      print('%10s '%'',end='')
    print('%10.4f'%(rmse_med))
print('best network');print(best_neurons[:-1])
print('best_rmse=%.4f'%rmse_mellor)
r=zeros(K);mae=zeros(K);y2=[];z2=[]
for k in range(K):
  tx[k]=vstack((tx[k],vx[k]));ty[k]=concatenate((ty[k],vy[k]))
  modelo=MLPRegressor(hidden_layer_sizes=best_neurons).fit(tx[k],ty[k])
  z=modelo.predict(sx[k]);z2=concatenate((z2,z))
  y=sy[k];y2=concatenate((y2,y))
  dif=y-z;rmse[k]=sqrt(mean(dif**2));
  t=corrcoef(y,z);r[k]=t[0,1];mae[k]=mean(abs(dif))

rmse_med=mean(rmse);r_med=mean(r);mae_med=mean(mae)
print('MLP regression dataset=%s RMSE=%.4f R=%.4f MAE=%.4f' \
  %(dataset,rmse_med,r_med,mae_med))
# scatterplot --------------------------------------------------------
from pylab import *
figure(1);clf();plot(y2,z2,'b.')
u=concatenate((y2,z2));vmin=min(u);vmax=max(u)
plot([vmin,vmax],[vmin,vmax],'r-')
xlabel('True value');ylabel('Predicted value')
title('Regression MLP dataset %s RMSE=%.4f R=%.4f MAE=%.4f' \
  %(dataset,rmse_med,r_med,mae_med))
grid(True);savefig('scatterplot_mlpr_python_%s.eps'%dataset);show()
# diagrama comparing true and predicted outputs
figure(2);clf();i=argsort(y2);plot(z2[i],'r.',label='Predicted');
plot(y2[i],'b-',label='True');grid(True)
legend(loc='lower right')
xlabel('Pattern number');ylabel('Value to predict')
title('Regression MLP %s RMSE=%.4f R=%.4f MAE=%.4f'% \
  (dataset,rmse_med,r_med,mae_med))
axis([1,len(y2),vmin,vmax])
savefig('outputs_mlpr_python_%s.eps'%dataset);show()
```

3. Exercises to do by the students

The lab work for the students is:

1. Download the datasets `wine.data` and `hepatitis.data` from the TEAMS.
2. Calculate the accuracy, Cohen kappa and confusion matrix for both datasets using the MLP and ELM classifiers using the whole dataset as training and test set and using the default configuration for the hyper-parameters. What happen if the number of hidden neurons change? See the effects.
3. Repeat the process using cross-validation with 4 folds. In this case, we must tune the hyper-parameters number of hidden layers and number of neurons for the MLP classifier, and the number of neurons for the ELM classifier. So, you must use the validation set to tune the hyper-parameters.
4. Use the MLP and ELM classifiers to the classification of the textures dataset.

Submit before 22 January by TEAMS the results and dificulties founded. It can be done individually or by groups.