Exercises of ANN classifiers

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

7 de diciembre de 2023

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 some library:

1. `nnet` package in `octave`[1] 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

Using the `nnet` package of `octave`, there are the following important functions:

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` preprocess additional data for neural network simulation (for example the test set).
3. `newff` create a feed-forward backpropagation network:
4. `train` a neural feed-forward network will be trained.

[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

5. `sim` is usuable to simulate a before defined and 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('%i ',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('%i ',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 %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)
```

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()` 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
```

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

```
clear;clc
rand('seed', 0);
```

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

```
%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 Multi-Layer Perceptron (MLP) classifier can be executed using the object `sklearn.neural_network.MLPClassifier` object. The training and test on the whole dataset can be executed using the following program:

```
from numpy import *
from sklearn.neural_network import *
from sklearn.metrics 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)
# preprocessing: mean 0, desviation 1
x=(x-mean(x,0))/std(x,0)
nHidden=30; # number of hidden neurons
nOutput=C; # number of output neurons
neurons=[nHidden, nHidden, nOutput]
modelo=MLPClassifier(hidden_layer_sizes=neurons).fit(x,y)
z=modelo.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)
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.neural_network import *
from sklearn.metrics 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)
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('best architecture: ');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_mellor).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('Confusion matrix:'); print(mc)
```

For the Extreme Learning Machine (ELM), we can use the following own implementation:

```
from numpy import *
from sklearn.metrics import *
from sys import exit
dataset='hepatitis'
#dataset='wine'
nf='%s.data'%dataset;x=loadtxt(nf)
y=x[:,0]-1;x=delete(x,0,1);
N=shape(x)[0];C=len(unique(y))
print('ELM dataset %s'%dataset)
def vec2ind(x):
        n=len(x);m=len(unique(x));y=zeros([n,m])
```

```
        for i in range(n):
                j=int(x[i]);y[i,j]=1
        return y
def elm(x,y,h,z):
        from numpy.random import random
        from numpy.linalg import pinv
        N,n=shape(x);C=len(unique(y))
        a=2*random([h,n])-1  # input weights
        H=1/(1+exp(-dot(a,x.T))); # matrix with activity of neurons in hidden layer
        t=vec2ind(y)  # true outputs for neurons in the output layer
        # (t[i,j]=1 only if pattern i is of class j)
        b=dot(pinv(H.T),t)  # output weights
        # output calculation--------------------------
        H=1/(1+exp(-dot(a,z.T)))  # activity in hidden layer
        q=dot(H.T,b)  # outputs of neurons in output layer
        r=argmax(q,1)  # predicted class label
        return r
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_best=-Inf;h_best=1
print('%10s %10s'%('H','Kappa(%)'))
for h in range(3,N):
        for k in range(K):
                z=elm(tx[k],ty[k],h,vx[k])
                vkappa[k]=100*cohen_kappa_score(vy[k],z)
        kappa_mean=mean(vkappa)
        print('%10i %10.1f'%(h,kappa_mean))
        if kappa_mean>kappa_best:
                kappa_best=kappa_mean;h_best=h
print('h_best=%i kappa_best=%.1f%%'%(h_best,kappa_best))
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
        sx2=(sx[k]-mx)/stdx;y=sy[k]
```

```
        z=elm(tx2,ty[k],h_best,sx2)
        vkappa[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=mean(vkappa);mc/=K
print('ELM dataset=%s kappa=%.2f%%'%(dataset,kappa))
print('confusion matrix:'); print(mc)
```

3. Classifiers comparison

As mention in the lecture of *Model selection and evaluation*, the Friedman test can be used to compare globally the performance of a set of classifiers over a set of datasets. The `Matlab` code was provided in the lecture, and the `Python` code is:

```
from numpy import *
from matplotlib.pyplot import *
a=loadtxt('datos.txt');
classifiers=['SVM', 'RF','NNET','GBM'];
#######################################################
# calculation of Friedman rank results
# perf: matrix with performance measures with models by rows
# and datasets by columns
def friedman_rank(perf):
    [nmodel, ndata]=perf.shape
    pos=zeros([nmodel, ndata])
    for i in range(ndata):
        # descending sort
        ind=argsort(perf[:,i])[::-1]
        for j in range(nmodel):
            pos[ind[j], i]=j+1
    print(pos)
    fr=mean(pos, 1)
    return fr
#######################################################
fr=friedman_rank(a);
print('%10s %10s %10s'%('Classifier', 'position', 'rank'))
n=a.shape[0]
ind=argsort(fr)
for i in range(n):
```

```
    print('%10s %10i %10.1f'% ( classifiers[ind[i]], i+1,fr[ind[i]]))
##############################################################
# box plots
clf(); boxplot(a.T)
xlabel('Classifiers'); ylabel('Kappa(%)')
xticks(range(1,n+1),classifiers); grid(True)
show(False)
```

4. 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 MLP and ELM classifiers using the whole dataset as training and test set and using the default configuration for the hyper-parameters. Default hyper-parameters for MLP classifier are: 1) neurons of output layer equal to the number of classes; and 2) a hidden layer with 30 neurons. For ELM classifier, the hyper-parameter is the number of neurons, which is 25 % of the training patterns. What happen if the number of hidden neurons change? See the effects.

2. 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. Provide the performance and the best configuration for the classifiers.

3. Use the MLP and ELM classifiers with other datasets. For example, the datasets `heart-disease-cleveland.data` (with 2 classes) and `annealing.data` (with 5 classes), which were downloaded from the UCI machine learning repository.

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

5. **Optional task**: calculate the performance of **Quick Extreme Learning Machine**[5] on the `letter`[6] dataset in the UCI Machine Learning Repository. The `letter` dataset has 20000 patterns with 16 atributes and 26 classes. Althought, it is quite large, it can be also processed by ELM 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 `here`. The code `qelm.m` and `runelm.m` run the Quick ELM and ELM in `octave` programming language, respectively.

[5] `https://link.springer.com/article/10.1007/s00521-021-06727-8`
[6] `https://archive.ics.uci.edu/ml/datasets/Letter+Recognition`