Exercises with ensembles

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

11 de enero de 2021

We practice with the use of two ensembles:

1. **Random Forest (RF)**: the most popular bagging algorithm, in which the base classifiers (random trees) are trained on different bootstrap samples of the training set.

2. **AdaBoost**: a popular boosting algorithm, in which the base classifiers are trained on the same training set but with different pattern weightings.

Both classifiers are available in libraries of the following programming languages:

1. **Python**: the `scikit-learn` package[1]: `RandomForestClassifier` and `AdaBoostClassifier` objects for RF and AdaBoost classifiers respetively.

2. **R**: function `randomForest` in the `RandomForest` package[2] for the RF classifier and function `boosting` in the `AdaBag` package[3] for AdaBoost classifier.

The tasks can be done using some of these programming languages (`R` or `Python`). The examples provided are coded in Python. The `sklearn.metrics`[4] module provides different functions to measure the classifier quality. The main functions are:

- `roc_curve(yTrue, yScore)`: compute Receiver operating characteristic (ROC), which is only applicable to two-class problems. If you have a multiclass problem, you can do a ROC for any pair of classes.

- `roc_auc_score(yTrue, yScore)`: compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores.

[1] https://scikit-learn.org/stable/supervised_learning.html#supervised-learning
[2] https://www.rdocumentation.org/packages/randomForest/versions/4.6-14/topics/randomForest
[3] https://www.rdocumentation.org/packages/adabag/versions/4.2
[4] https://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics

- `cohen_kappa_score(y1, y2)`: Cohen's kappa, which is a statistic that measures inter-annotator agreement.
- `confusion_matrix(yTrue, yPred)`: compute confusion matrix to evaluate the accuracy of a classifier.
- `accuracy_score(yTrue, yPred)`: accuracy classification score.
- `precision_score(yTrue, yPred)`: compute the precision.
- `recall_score(yTrue, yPred)`: compute the recall.
- `f1_score(yTrue, yPred)`: compute the F1 score, also known as balanced F-score or F-measure.

The code to apply RF classifier to the dataset `hepatitis` (two-class problem) using the above classification measures are in the following:

```
# random forest using a training and a testing set
from sklearn.ensemble import *
from sklearn.metrics import *
from sklearn.model_selection import *
from numpy import *
# 3 classes
#dataset='wine';x=loadtxt('wine.data');
# 2 classes
dataset='hepatitis';x=loadtxt('hepatitis.data');
c=x[:,0]-1 # the true vulue
x=delete(x,0,1) # delete column 0
[N,I]=x.shape
cl=unique(c);C=len(cl);
# preprocessing: mean 0, desviation 1
x=(x-mean(x,0))/std(x,0)
# Split the dataset in two equal parts
Xtrain, Xtest, ytrain, ytest = train_test_split(x, c, test_size=0.5, random_state=0)
# Create the RF classifier with 10 random trees
model = RandomForestClassifier(n_estimators=10)
# Train the classifier
model = model.fit(Xtrain, ytrain)
# Compute the prediction of the classifier for the test set
y=model.predict(Xtest)
kappa=cohen_kappa_score(ytest, y)
print('Dataset=%s, kappa=%.2f%%'%(dataset,kappa*100))
cf=confusion_matrix(ytest,y)
print('Confusion matrix'); print(cf)
```

```
a=accuracy_score(ytest, y)
print('Accuracy= %.2f'%a)
pre=precision_score(ytest,y)
re=recall_score(ytest,y)
f1=f1_score(ytest,y)
print('Precision=%.2f, recall=%.2f and F1=%.2f' %(pre, re, f1))
# ROC curve ----------------------------------
aux=model.predict_proba(Xtest)
p=aux[:,1] # probability of class 1
fpr, tpr, thresholds = roc_curve(ytest,p)
from matplotlib.pyplot import *
clf(); plot(fpr,tpr,'bs--');
ylabel('True positive rate')
xlabel('False positive rate')
title('AUC= %.4f'% roc_auc_score(ytest,p))
grid(True); show(False)
```

where the function `train_test_split()` is used to divide the input data into two random sets `Xtrain` and `XTest`, which are used to train and test the RF classifier respectively. The training of the RF classifier is done using recommended values (number of random trees equal to 10).

The code to use the **AdaBoost** classifier on the datasets `wine` and `hepatitis` using: 1) the whole dataset as training and testing set; and 2) the cross-validation functionality provided by `cross_val_predict()` function. In both cases, the default parameters of `AdaBoost` classifier are used.

```
from sklearn.ensemble import *
from sklearn.metrics import *
from sklearn.model_selection import *
from numpy import *
# 3 classes
dataset='wine';x=loadtxt('wine.data');
# 2 classes
# dataset='hepatitis';x=loadtxt('hepatitis.data');
c=x[:,0]-1
x=delete(x,0,1) # delete column 0
[N,I]=x.shape
cl=unique(c);C=len(cl);
# preprocessing: mean 0, desviation 1
x=(x-mean(x,0))/std(x,0)
# using the whole dataset as training and testing set
model = AdaBoostClassifier()
model = model.fit(x, c)
```

```
y=model.predict(x)
kappa=cohen_kappa_score(c, y)
print('Dataset=%s, kappa=%.2f%%'%(dataset,kappa*100))
cf=confusion_matrix(c,y)
print('Confusion matrix'); print(cf)
# 4-fold cross validation-------------------------------------
y2=cross_val_predict(model, x, c, cv=4)
kappa=cohen_kappa_score(c, y2)
print('4fold cv: kappa=%.2f%%'%(kappa*100))
cf=confusion_matrix(c,y2)
print('Confusion matrix'); print(cf)
```

The tuned parameters for RF classifier are the number of decision trees B (parameter `n_estimators`) and the number of features q (parameter `max_features`) to use in each node (normally used as the squared root of the number of input features). The `scikit-learn` package provides utilities to tune the hyper-parameters of a classifier. Two generic approaches to parameter search are provided: for given values, `GridSearchCV()` function exhaustively considers all parameter combinations, while `RandomizedSearchCV` can sample a given number of candidates from a parameter space with a specified distribution. As we are using small datasets, we only use the first approach. A search consists of a classifier (RF or AdaBoost), a parameter space (set of values for each tuned parameter), a method for searching or sampling candidates, a cross-validation scheme and a score function. We can tune the hyper-parameter using two approaches:

1. **Approach 1 to tune the hyper-parameters**: divide the whole dataset into two datasets (train set and test set), using the train set to tune the hyper-parameters using the `GridSearchCV()` function and, once the best values for the hyper-parmeters are calculated, test the RF classifier over the test set.

2. **Appoach 2 to tune the hyper-parameters**: use the whole dataset to tune the hyper-parameters using the function `GridSearchCV()` and, once the best values for the hyper-parmeters are calculated, test the RF classifier over the whole dataset using cross-validation (using the `cross_val_predict()` function).

The following code is an example of the tuning parameters (`n_estimators` and `max_features`) of the RF classifier using the approach 1:

```
from sklearn.ensemble import *
from sklearn.metrics import *
from sklearn.model_selection import *
from numpy import *
# 3 classes
#dataset='wine';x=loadtxt('wine.data');
# 2 classes
```

```
dataset='hepatitis';x=loadtxt('hepatitis.data');
c=x[:,0]-1
x=delete(x,0,1) # delete column 0
[N,I]=x.shape
cl=unique(c);C=len(cl);
# Split the dataset in two equal parts
Xtrain, Xtest, ytrain, ytest = train_test_split(x, c, test_size=0.5, random_state=0)
# Tune the hyper-parameters
# Set the values of hyper-parameters to tune
tunedParameters = {'n_estimators': [5,10,15,20,25,30], 'max_features': [3,5,7,9]}
# Set the score function
score = 'f1' #'recall' 'precision'
# Set the method for searching and classifier
model = GridSearchCV(RandomForestClassifier(random_state=0), \
    tunedParameters, scoring='%s_macro' % score)
# preprocessing: mean 0, desviation 1
mx=mean(Xtrain,0); stdx=std(Xtrain,0)
Xtrain=(Xtrain-mx)/stdx
model.fit(Xtrain, ytrain)
print("Best parameters set found on development set:")
print(model.best_params_)
print("Grid scores on development set:")
means = model.cv_results_['mean_test_score']
stds = model.cv_results_['std_test_score']
for mean, std, params in zip(means, stds, model.cv_results_['params']):
    print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))
print()
Xtest=(Xtest-mx)/stdx # normalization
# Compute the classifier prediction on the test set
ytrue, ypred = ytest, model.predict(Xtest)
kappa=cohen_kappa_score(ytrue, ypred)
print('Dataset=%s, kappa=%.2f%%'%(dataset,kappa*100))
cf=confusion_matrix(ytrue,ypred)
print('Confusion matrix'); print(cf)
```

The following code is an example of the tuning parameters (`n_estimators` and `max_features`) of the RF classifier using the approach 2:

```
from sklearn.ensemble import *
from sklearn.metrics import *
from sklearn.model_selection import *
from numpy import *
# 3 classes
```

```
#dataset='wine';x=loadtxt('wine.data');
# 2 classes
dataset='hepatitis';x=loadtxt('hepatitis.data');
c=x[:,0]-1
x=delete(x,0,1) # delete column 0
[N,I]=x.shape
cl=unique(c);C=len(cl);
# preprocessing: mean 0, desviation 1
x=(x-mean(x,0))/std(x,0)
# Set the parameters by cross-validation
tunedParameters = {'n_estimators': [5,10,15,20,25,30], 'max_features': [3,5,7,9]}
score = 'f1' #'recall' 'precision'
model = GridSearchCV(RandomForestClassifier(random_state=0), \
    tunedParameters, scoring='%s_macro' % score)
model.fit(x, c)
print("Best parameters set found on development set:")
print(model.best_params_)
print("Grid scores on development set:")
means = model.cv_results_['mean_test_score']
stds = model.cv_results_['std_test_score']
for mean, std, params in zip(means, stds, model.cv_results_['params']):
    print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))
print()
# 4-fold cross validation-------------------------------------
y2=cross_val_predict(model, x, c, cv=4)
kappa=cohen_kappa_score(c, y2)
print('4fold cv: kappa=%.2f%%'%(kappa*100))
cf=confusion_matrix(c,y2)
print('Confusion matrix'); print(cf)
```

Both approaches have some limitations. The approach 1 tests the classifier only over one partition of the whole dataset, so, its performance depends on the partitions selected. The approach 2 tunes the hyper-parameters using the whole dataset and then tests the classifier also over the whole dataset, so, it is expected that the performance achieved will be quite optimistic. For both approaches the selection of the patterns does not guarantee that the relative class population is kept for all partitions. Thus, it is not known the way in which the selection of the best tuned hyper-parameters is done.

On other hand, in some real classification problem, it is necesary to design a specific evaluation methodology. So,the following code is the RF classifier tuning the hyper-parameters using cross-validation with the use of training, validation and testing sets, which provides a more realistic performance.

```
from sklearn.ensemble import *
from sklearn.metrics import *
```

```
from sklearn.model_selection import *
from numpy import *
# 3 classes
#dataset='wine';x=loadtxt('wine.data');
# 2 classes
dataset='hepatitis';x=loadtxt('hepatitis.data');
c=x[:,0]-1
x=delete(x,0,1) # delete column 0
[N,I]=x.shape
cl=unique(c);C=len(cl)
# 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
def createFolds(x, c, K):
    from numpy.random import shuffle,seed
    seed(100)
    [N,n]=x.shape # Number of patterns and features
    val=unique(c) # output values
    Q=len(val) # number of classes
    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(Q,'int') # 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(Q,'int');nvp=zeros(Q,'int')
    nsp=zeros(Q,'int');npf=zeros(Q,'int')
    tx=[];tc=[]
    vx=[];vc=[]
    sx=[];sc=[]
    for i in range(K):
        tx.append(zeros(n));vx.append(zeros(n));sx.append(zeros(n));
        tc.append(array([],'int'))
        vc.append(array([],'int'))
        sc.append(array([],'int'));
    for i in range(Q):
        t=where(c==i)[0];j=len(t);npc[i]=j
        print('  class %i: %i patterns'% (i,j))
        shuffle(t) # 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=0
        for k in range(K):
            p=start
            for l in range(ntp[i]):  # indices of train patterns
                m=t[p]; tx[k]=vstack((tx[k],x[m])); tc[k]=append(tc[k],c[m])
                p=(p+1)%npc[i]
            for l in range(nvp[i]): # indices of validation patterns
                m=t[p]; vx[k]=vstack((vx[k],x[m])); vc[k]=append(vc[k],c[m])
                p=(p+1)%npc[i]
            for l in range(nsp[i]):  # indices of test patterns
                m=t[p]; sx[k]=vstack((sx[k],x[m])); sc[k]=append(sc[k],c[m])
                p=(p+1)%npc[i]
            start=start+npf[i];
    for k in range(K):
        tx[k]=delete(tx[k],0,0);vx[k]=delete(vx[k],0,0);sx[k]=delete(sx[k],0,0)
    return [tx,tc,vx,vc,sx,sc]

K=4
tx, tc, vx, vc,sx,sc=createFolds(x,c,K)

## Set the parameters by cross-validation
tunedParameters = {'n_estimators': [5,10,15,20,25,30], 'max_features': [3,5,7,9]}
score = 'f1' #'recall' 'precision'
avg=[];dev=[]
for k in range(K):
    avg.append(mean(tx[k],0));dev.append(std(tx[k],0))
vne=[5,10,15,20,25,30] #  number of estimators
vmf=[3,5,7,9]  # max features
vkappa=zeros(K);kappa_best=-Inf
print('%10s %10s %10s'%('#estimators','max features','kappa(%)'))
for ne in vne:
    for mf in vmf:
        for k in range(K):
            model=RandomForestClassifier(n_estimators=ne,\
                max_features=mf,random_state=0)
            tx2=(tx[k]-avg[k])/dev[k];
            model.fit(tx2,tc[k])
            vx2=(vx[k]-avg[k])/dev[k]
            y=model.predict(vx2)
            vkappa[k]=cohen_kappa_score(vc[k],y)
        kappa_mean=mean(vkappa)
        print('%10i %10i %10.2f%%'%(ne,mf,100*kappa_mean))
```

```
        if kappa_mean>kappa_best:
            kappa_best=kappa_mean; ne_best=ne; mf_best=mf
print('Best parameters: #estimators=%i max features=%i kappa=%.2f%%'\
    %(ne_best,mf_best,100*kappa_best))
cf=zeros([C,C])
for k in range(K):
    tx[k]=vstack((tx[k],vx[k]));tc[k]=concatenate((tc[k],vc[k]))
    mx=mean(tx[k],0);stdx=std(tx[k],0)
    tx2=(tx[k]-mx)/stdx
    model=RandomForestClassifier(n_estimators=ne_best,\
        max_features=mf_best,random_state=0)
    model.fit(tx2,tc[k])
    sx2=(sx[k]-mx)/stdx
    y=model.predict(sx2)
    vkappa[k]=cohen_kappa_score(sc[k],y)
    cf+=confusion_matrix(sc[k],y)
kappa=mean(vkappa);cf/=K
print('Dataset=%s, kappa=%.2f%%'%(dataset,kappa*100))
print('Confusion matrix'); print(cf)
```

The user defined `createFolds()` function divides the whole dataset into a K number of folds, similary to the same function for `octave/matlab` language.

1. 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. For the two-class problem `hepatitis`, divide randomly the dataset into two partitions: `trainset` and `testset`, and calculate the following quality measures for the classifier RF and AdaBoost: accuracy, Cohen kappa, confusion matrix, precision, recall, F1, Receiver Operating Characteristic (ROC) curve and Area Under ROC Curve (AUC).
3. Calculate the Cohen kappa and confusion matrix for both classifiers (RF and AdaBoost) and both datasets (`wine` and `hepatitis`) using cross-validation provided by `scikit-learn` module (`cross_val_predict()` function) with the number of folds 4, 5 and 10. For the AdaBoost classifier, tune the number of base classifier (parameter `n_estimators` in the `AdaBoostClassifier()` function of `scikit-learn` module of Python) from 40 up to 200 with a step=20. Analyse the results.
4. Calculate the accuracy, Cohen kappa and confusion matrix for both datasets using for both classifiers (RF and AdaBoost) using cross-validation with 4 folds and tune the hyper-parameters using the above approaches.

5. Use the RF classifier to the classification of the textures dataset and compare with other classifiers (SVM, LDA, MLP).

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