script

April 4, 2018

In [ ]:
```
import multiprocessing as mp
import pandas as pd
import numpy as np
import math
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
import copy
import json
```

In [ ]:
```
def cross_data(X,y,X_test,y_test,k_fold=5):
    """
    The function to divide data for cross validation

    args:
    X:train data
    y:train data label
    X_test:test data
    y_test:test data label
    k_fold: number of "K" for cross validation

    return:
    X_train,X_test,y_train,y_test:list of each data
    example: X_train[0] is the train data for first step cross validation
    train_idx_list,test_idx_list: list of index we selected
    """
    kf=KFold(n_splits=k_fold,shuffle=True)
    X_train=[]
    X_test=[]
    y_train=[]
    y_test=[]
    train_idx_list=[]
    test_idx_list=[]
    for train_idx,test_idx in kf.split(X):

        X_train.append(X[train_idx])
        X_test.append(X[test_idx])
        y_train.append(y[train_idx])
```

```
            y_test.append(y[test_idx])
            train_idx_list.append(train_idx)
            test_idx_list.append(test_idx)
        return X_train,X_test,y_train,y_test,train_idx_list,test_idx_list
```

In [ ]:
```
def train_process(q2,X,w,y):
    """
    A fucntion of trainning process.

    args:
    q2:Queue Obeject in python for store returns of python multiprocess
    X:train data
    w:weights in Adaboost algorithm
    y:data label

    returns:
    store each train step of pars,alpha and weights in Queue object
    """
    temp_pars=train(X,w,y)
    temp_error=_error(X,y,w,temp_pars)
    temp_alpha=np.log((1-temp_error)/temp_error)
    tempw=np.multiply(w,np.exp(temp_alpha*(classify(X,temp_pars)!=y)))
    out=temp_pars,temp_alpha,tempw
    q2.put(out)
```

In [ ]:
```
def stump_classifier(x,par):
    """
    An implementation of Stumpy Classifier, which could be regard as simple tree with
    args:
    x:a single sample of data,which is a column of homework decribe.
    par:parameters for this classifier.

    returns:
    Classification of this algorithm.

    """
    j,theta,m=par
    x=np.array(x)
    length=x.shape[1]
    x=x.reshape(length)
    #print(x.shape)
    if x[j]>theta:
        return m
    else:
        return -m
```

In [ ]:
```
def classify(X,pars):
    """
```

```
        One of the function that homework request.
        Classify the whole data input(X)

        args:
        X:data want to classify
        pars:parameters for this classification step

        returns:
        label:A vector of classification that each sample is classified to.
        """
        j,theta,m=pars
        thredhold=X[:,j].reshape(X.shape[0])
        #print(thredhold)
        label=np.array([m if x>theta else -m for x in thredhold])
        return label
```

```
In [ ]: def _error(X,y,w,pars):
        """
        A function to calculate error, which is the implementation of (2) of homework
        ars:
        X:data
        y:label of data
        w:weights
        pars:parameters

        returns:
        error:the weighted error described in (2)"""
        #n=X.shape[0]
        wrong_match=y!=classify(X,pars)
        error=sum(np.multiply(w,wrong_match))/sum(w)
        return error
```

```
In [ ]: def train(X,w,y):
            """
            One of the function that homework request.
            Train the data once with weights give.

            args:
            X:data
            w:weights
            y:label of data

            returns:
            pars:(j,theta,m) The parameter decribed in homework.
            """
            theta_rage = np.arange(-1, 1, 0.01)
            n = len(theta_rage)
            p = X.shape[1]
```

```
    m_list = np.array([1, -1])
    error_list = np.zeros((n, p, 2))
    theta_list = np.zeros((n, p, 2))

    for j in range(p):
            print(j)
    # Xj=X[:,j]
    # print(Xj)
    # Xj=np.array(Xj)
    # length=Xj.shape[0]
    # Xj=Xj.reshape(length)
    # Xj_sorted=np.sort(Xj)#increasing
            for itr_m in range(2):

                    for k in range(n):
                            theta = theta_rage[k]
                            if itr_m == 0:
                                    temp_pars = (j, theta, 1)
                            else:
                                    temp_pars = (j, theta, -1)
                    error_list[k, j, itr_m] = _error(X, y, w, temp_pars)
                    theta_list[k, j, itr_m] = theta

                    idx_i, idx_j, idx_m = np.unravel_index(error_list.argmin(), error_list
                    theta = theta_list[idx_i][idx_j][idx_m]
                    j = idx_j
                    m = m_list[idx_m]
                    pars = (j, theta, m)
                    return pars
```

In [ ]:
```
def agg_class(X,alpha,allPars):
    """
    One of the function that homework request.
    Given all parameters we have during training,predict the final label.

    args:
    X:data
    alpha: a list of alpha we get during training
    allPars: a list of pars we get during training

    returns:
    c_hat:a vector of label that we predict.
    """
    B=len(alpha)
    c_hat=np.zeros((X.shape[0]))
    for b in range(B):
        c_hat+=classify(X,allPars[b])*alpha[b]
    c_hat=np.array([1 if x>0 else -1 for x in c_hat])
```

```
            return c_hat
```

In [ ]:
```
"""
Deal with training data.
#data# is the training data. #label# is training data label.

"""
train3=pd.read_table('train_3.txt',sep=",",header=None)
train8=pd.read_table('train_8.txt',sep=",",header=None)
test_raw=pd.read_table('zip_test.txt',sep=" ",header=None)
n3=train3.shape[0]
n8=train8.shape[0]
nwhole=n3+n8
_=np.array([1,-1])
label=np.repeat(_,[n3,n8],axis=0)
data_pd=pd.concat([train3,train8])
data=np.array(data_pd)
```

In [ ]:
```
"""
Dealing with test data
#data_test# is test data
#label_test# is test data label
"""
test_raw=np.matrix(test_raw)
test_label_raw=test_raw[:,0]
goodvalues=[3,8]
draw=np.where(test_label_raw==goodvalues)

test_data=test_raw[:,1:]
test_label=test_label_raw[draw[0]]
#print(test_label)
test_label=np.array(test_label.T)[0]
test_label=test_label.astype('int').astype('str')
label_test=np.array([1 if x=="3" else -1 for x in test_label])
data_test=np.array(test_data[draw[0],:])
```

In [ ]:
```
"""
Get data
"""
X_train,X_test,y_train,y_test,train_idx_list,test_idx_list=cross_data(data,label,data_t
X=data
y=label
```

In [ ]:
```
"""
Num of B
"""
B=20
```

In [ ]:
```
"""
some list to store result
```

```
        """
        n=X.shape[0]
        w=np.repeat(1/n,n)
        allPars=[]
        alpha=[]
        val_err=[]
        test_err=[]

        #cross pars
        n1=X_train[0].shape[0]
        n2=X_train[1].shape[0]
        n3=X_train[2].shape[0]
        n4=X_train[3].shape[0]
        n5=X_train[4].shape[0]
        w1=np.repeat(1/n1,n1)
        w2=np.repeat(1/n2,n2)
        w3=np.repeat(1/n3,n3)
        w4=np.repeat(1/n4,n4)
        w5=np.repeat(1/n5,n5)
        allPars1=[]
        alpha1=[]
        allPars2=[]
        alpha2=[]
        allPars3=[]
        alpha3=[]
        allPars4=[]
        alpha4=[]
        allPars5=[]
        alpha5=[]
```

In [ ]:
```
        if __name__ == '__main__':
        """
        The main function. This is implementation of Adaboost.
        """
            for b in range(B):

                        q1=mp.Queue()
                        q2=mp.Queue()
                        q3=mp.Queue()
                        q4=mp.Queue()
                        q5=mp.Queue()
                        q6=mp.Queue()
                        p1=mp.Process(target=train_process,args=(q1,X_train[0],w1,y_train[0],)
                        p2=mp.Process(target=train_process,args=(q2,X_train[1],w2,y_train[1],)
                        p3=mp.Process(target=train_process,args=(q3,X_train[2],w3,y_train[2],)
                        p4=mp.Process(target=train_process,args=(q4,X_train[3],w4,y_train[3],)
                        p5=mp.Process(target=train_process,args=(q5,X_train[4],w5,y_train[4],)
                        p6=mp.Process(target=train_process,args=(q6,X,w,y,))
```

```
p1.start()
p2.start()
p3.start()
p4.start()
p5.start()
p6.start()
p1.join()
p2.join()
p3.join()
p4.join()
p5.join()
p6.join()
newpara1=q1.get()
newpara2=q2.get()
newpara3=q3.get()
newpara4=q4.get()
newpara5=q5.get()
newpara=q6.get()
#multiprocessing

#train and update paras
#temp_pars,temp_alpha,tempw=train_process(X,w,y)
temp_pars,temp_alpha,tempw=newpara
allPars.append(temp_pars)
alpha.append(temp_alpha)
w=tempw
c_hat_test=agg_class(data_test,alpha,allPars)
test_err.append(np.mean(c_hat_test!=label_test))

#cross-val and update
val_err_temp=[]

temp_pars1,temp_alpha1,tempw1=newpara1
allPars1.append(temp_pars1)
alpha1.append(temp_alpha1)
w1=tempw1
c_hat_test1=agg_class(X_test[0],alpha1,allPars1)
val_err_temp.append(np.mean(c_hat_test1!=y_test[0]))

temp_pars2,temp_alpha2,tempw2=newpara2
allPars2.append(temp_pars2)
alpha2.append(temp_alpha2)
w2=tempw2
c_hat_test2=agg_class(X_test[1],alpha2,allPars2)
val_err_temp.append(np.mean(c_hat_test2!=y_test[1]))

temp_pars3,temp_alpha3,tempw3=newpara3
allPars3.append(temp_pars3)
```

```
                alpha3.append(temp_alpha3)
                w3=tempw3
                c_hat_test3=agg_class(X_test[2],alpha3,allPars3)
                val_err_temp.append(np.mean(c_hat_test3!=y_test[2]))

                temp_pars4,temp_alpha4,tempw4=newpara4
                allPars4.append(temp_pars4)
                alpha4.append(temp_alpha4)
                w4=tempw4
                c_hat_test4=agg_class(X_test[3],alpha4,allPars4)
                val_err_temp.append(np.mean(c_hat_test4!=y_test[3]))

                temp_pars5,temp_alpha5,tempw5=newpara5
                allPars5.append(temp_pars5)
                alpha5.append(temp_alpha5)
                w5=tempw5
                c_hat_test5=agg_class(X_test[4],alpha5,allPars5)
                val_err_temp.append(np.mean(c_hat_test5!=y_test[4]))
                val_err.append(np.mean(val_err_temp))

whole=[allPars,alpha,val_err,test_err]
with open("final_result.txt","w",encoding="utf-8") as f:
                f.write(str(whole))
    #with open("result_json.txt","w+") as fj:
                #json.dump(whole,fj)
```