华南理工大学

South China University of Technology

The Experiment Report of Machine Learning

SCHOOL: SCUT

SUBJECT: SOFTWARE ENGINEERING

Author:
JialeDeng ZhiweiChen Zhengzhang

Supervisor:
Mingkui Tan

Student ID：
201530611371 , 201530611340
and 201530613702

Grade:
Undergraduate

December 22, 2017

Human face classification based on AdaBoost algorithm

Abstract—

I. INTRODUCTION

We want to understand AdaBoost further and get familiar with the basic method of face detection.

Learn to use Adaboost to solve the face classification problem, and combine the theory with the actual project.

Experience the complete process of machine learning.

II. METHODS AND THEORY

1. the weight of all training samples initialized by 1 / N, of which N is a sample number

2.

A). Training the weak classifier YM () to minimize the weight error function (weighted error function):

$$\epsilon_m = \sum_{n=1}^{N} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)$$

B) then the discourse weight alpha of the weak classifier is calculated.

$$\alpha_m = \ln\left\{\frac{1-\epsilon_m}{\epsilon_m}\right\}.$$

C) update weight:

$$w_{m+1,i} = \frac{w_{mi}}{Z_m}\exp(-\alpha_m \mathbf{t}_i\, \mathbf{y}_m(x_i)), \quad i = 1, 2, \cdots, N$$

Zm: is a normalization factor that makes all W and 1. (the formula is a little bit messy here)

$$Z_m = \sum_{i=1}^{N} w_{mi}\exp(-\alpha_m \mathbf{t}_i\, \mathbf{y}_m(x_i))$$

3. get the final classifier:

$$Y_M(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m y_m(\mathbf{x})\right).$$

III. EXPERIMENT

Experimental steps:

a) Read data set data. The images are supposed to converted into a size of 24 * 24 grayscale, the number and the proportion of the positive and negative samples is not limited, the data set label is not limited.

b) Processing data set data to extract NPD features. Extract features using the NPDFeature class in feature.py. (Tip: Because the time of the pretreatment is relatively long, it can be pretreated with pickle function library dump () save the data in the cache, then may be used load () function reads the characteristic data from cache.)

c) The data set is divisded into training set and calidation set, this experiment does not divide the test set.

d) Write all AdaboostClassifier functions based on the reserved interface in ensemble.py. The following is the guide of fit function in the AdaboostClassifier class:

d1 Initialize training set weights , each training sample is given the same weight.

d2 Training a base classifier , which can be sklearn.tree library DecisionTreeClassifier (note that the training time you need to pass the weight as a parameter).

d3 Calculate the classification error rate of the base classifier on the training set.

d4 Calculate the parameter according to the classification error rate.

d5 Update training set weights .

d6 Repeat steps 4.2-4.6 above for iteration, the number of iterations is based on the number of classifiers.

e) Predict and verify the accuracy on the validation set using the method in AdaboostClassifier and use classification_report () of the sklearn.metrics library function writes predicted result to report.txt .

f) Organize the experiment results and complete the lab report (the lab report template will be included in the example repository).

Code:

AdaBoostClassifier

```
class AdaBoostClassifier:
    '''A simple AdaBoost Classifier.'''

    def __init__(self, weak_classifier, n_weakers_limit):
        '''Initialize AdaBoostClassifier

        Args:
```

```
        weak_classifier: The class of weak classifier, which is recommend to be sklearn.tree.DecisionTreeClassifier.
        n_weakers_limit: The maximum number of weak classifier the model can use.
        '''
        self.weaker = weak_classifier
        self.M = n_weakers_limit

    def is_good_enough(self,X,y):
        '''Optional'''
        y_pred = self.predict(X)
        y_pred.resize((len(y_pred),1))
        idx = np.where((y_pred-y)!=0)
        return len(idx[1]) #return the number of wrong prediction

    def fit(self,X,y):
        '''Build a boosted classifier from the training set (X, y).

        Args:
            X: An ndarray indicating the samples to be trained, which shape should be (n_samples,n_features).
            y: An ndarray indicating the ground-truth labels correspond to X, which shape should be (n_samples,1).
        '''
        n = X.shape[0]
        self.G = {}
        self.alpha = {}
        for i in range(self.M):
            self.G.setdefault(i)
            self.alpha.setdefault(i)
        self.sum=np.zeros(y.shape)
        self.W=np.ones((n,1))/n
        self.cnt=0 # initialize
        for i in range(self.M):
            w = self.W.flatten(1)
            self.G[i] = self.weaker.fit(X,y,sample_weight=w)
            e = 1-self.G[i].score(X,y,sample_weight=w)
            if e > 0.5:
                break # bad classifier
            self.alpha[i] = 1/2*np.log((1-e)/max(e,1e-16))
            h = self.G[i].predict(X)
            h.resize((n,1))
            #print('h',h[:10])
            Z = np.multiply(self.W,np.exp(-self.alpha[i]*np.multiply(y,h)))
            #print('Z',Z[:10])
            self.W = (Z/Z.sum())
            #print('W',self.W[:10])
            #print(self.W.sum()) # the sum should be 1

            self.cnt = i+1 # number of weakers

            #show the result in each iteration
            #target_names = ['NEGATIVE', 'POSITIVE']
            #y_pred = AdaBoost.predict(X)
            #result = classification_report(y,y_pred,target_names=target_names)
            #print(result)
            #print(self.is_good_enough(X,y))

            if self.is_good_enough(X,y) == 0:
                print(self.cnt,"weak classifiers is already good enough.")
                break

    def predict_scores(self, X):
        '''Calculate the weighted sum score of the whole base classifiers for given samples.

        Args:
            X: An ndarray indicating the samples to be predicted, which shape should be (n_samples,n_features).

        Returns:
            An one-dimension ndarray indicating the scores of differnt samples, which shape should be (n_samples,1).
        '''
        sum = np.zeros((X.shape[0],1))
        for i in range(self.cnt):
            t = self.G[i].predict(X).flatten(1)*self.alpha[i]
            t.resize((X.shape[0],1))
            sum = sum+t
        return sum


    def predict(self, X, threshold=0):
        '''Predict the catagories for geven samples.

        Args:
            X: An ndarray indicating the samples to be predicted, which shape should be (n_samples,n_features).
            threshold: The demarcation number of deviding the samples into two parts.

        Returns:
            An ndarray consists of predicted labels, which shape should be (n_samples,1).
        '''
        y_pred = self.predict_scores(X)
        y_pred[y_pred>=threshold] = +1
        y_pred[y_pred<threshold] = -1
        return y_pred

    @staticmethod
    def save(model, filename):
        with open(filename, "wb") as f:
            pickle.dump(model, f)

    @staticmethod
    def load(filename):
        with open(filename, "rb") as f:
            return pickle.load(f)
```

train.py

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

IMG_SIZE = 12 * 12

def get_path(path):
    return [os.path.join(path,f) for f in os.listdir(path)]

def grayscale(src_path,dst_path):
    #imgs = get_path(src_path)
    for src in os.listdir(src_path):
        dst = os.path.join(dst_path,src)

Image.open(os.path.join(src_path,src)).resize((24,24)).convert('L').save(dst)

def extract(path):
    features = []
    cnt = 0
    for img in os.listdir(path):
        f = NPDFeature(np.array(Image.open(os.path.join(path,img)))).extract()
        #print(f)
        features.append(f)
        cnt = cnt + 1
    return cnt,features

def init_features():
    grayscale('datasets\\original\\face','datasets\\gray\\face')

grayscale('datasets\\original\\nonface','datasets\\gray\\nonface')
    (cnt0,features0) = extract('datasets\\gray\\nonface')
    (cnt1,featrues1) = extract('datasets\\gray\\face')
    y = np.ones((cnt0+cnt1,1))
    y[:cnt0] = -1
    x = np.array([features0,featrues1]).reshape((1000,-1))
    AdaBoostClassifier.save(x,'x.ds')
    AdaBoostClassifier.save(y,'y.ds')

if __name__ == "__main__":
    # write your code here
    if not(os.path.isfile('x.ds') and os.path.isfile('y.ds')):
    init_features()
    x = AdaBoostClassifier.load('x.ds')
    y = AdaBoostClassifier.load('y.ds')
    print('the size of X:',x.shape)
    print('the size of y:',y.shape)
    x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.33)

    #AdaBoosting
    AdaBoost = AdaBoostClassifier(DecisionTreeClassifier(max_depth=2),10)
    AdaBoost.fit(x_train,y_train)
    print('the wrong number of train sample:',AdaBoost.is_good_enough(x_train,y_train))

    #show the result
    target_names = ['NEGATIVE', 'POSITIVE']
    y_pred = AdaBoost.predict(x_test)
    result = classification_report(y_test,y_pred,target_names=target_names)

    print(result)
    with open("report.txt","w") as f:
        f.write(result)
```

Experimental results:

```
In [127]: y_pred = AdaBoost.predict(x_test)
          #print( y_pred)
          #print(y_test)
          result = classification_report(y_pred, y_test, target_names=target_names)
          print(result)
```

	precision	recall	f1-score	support
NEGATIVE	0.68	0.86	0.76	141
POSITIVE	0.87	0.70	0.78	189
avg / total	0.79	0.77	0.77	330

(3 weak classifiers)

IV. CONCLUSION

In this experiment, I have learned the power of Ada--Boosting. Although the base learner is weak (in this case I use a low-max-depth DecisionTreeClassifier), the algorithm stopped soon because the error for training samples is already 0 (only using 7 weakers), and the fit result is also well.