華南理工大學

South China University of Technology

The Experiment Report of Machine Learning

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
Siyuan Xiao, Li Zhang, Shengyan Wen

Supervisor:
Qingyao Wu

Student ID：
201721045886, 201721045909, 201721045893

Grade:
Postgraduate

December 18, 2017

Face Classification Based on AdaBoost Algorithm

Abstract—AdaBoost is a popular machine learning framework that combines multiple weak learners, and finally gain more accuracy than the original single weak method.

I. Introduction

This report will talk about the whole experiment I have made on a face classifier based on AdaBoost. Its content is organized as follow:

1) Section II contains the experiment steps.
2) Section III contains the code for the two experiments.
3) Section IV makes conclusion for the experiment result.

II. Methods and Theory

Here we use 1,000 images which is placed in the requirement. Half of them contain faces and the rest do not.

Then the experiment will be performed by the following steps:

1) If it is not the first time executing this program, load the preprocessed data into memory and go to step 6; else, go to step 2.
2) Covert the images into grayscale mode.
3) Resize the images to 24 x 24.
4) Extract NPD feature vectors from them.
5) Split the feature data into training set and validation set, then save the preprocessed data.
6) Set parameters for the AdaBoost classifier, then start to train it.
7) In each iteration, train a base classifier by the given weights, and calculate weights for the next classifier.
8) Do prediction by combining all the trained classifiers.
9) Print the prediction result, compare with the single weak classifier.

III. Experiment

Here I placed the critical code for the experiment:

1) ensemble.py:

```
import pickle
import math
import numpy as np
import copy

class AdaBoostClassifier:
    '''A simple AdaBoost Classifier.'''
    __base_classifier__ = None
    __classifiers__ = None
    __max_base__ = 0
    __n_base__ = 0
    __alpha__ = None

    def __init__(self, weak_classifier, n_weakers_limit):
        self.__base_classifier__ = weak_classifier
        self.__max_base__ = n_weakers_limit

    def fit(self,X,y):
        self.__alpha__ = []
        self.__classifiers__ = []
        W = np.zeros([self.__max_base__, X.shape[0]])
        W[0, :] = 1 / X.shape[0]
        for m in range(0, self.__max_base__):
            #train m-th classifier
            print ("train the " + str(m + 1) + " base classifier")
            base = copy.deepcopy(self.__base_classifier__)
            base = base.fit(X, y, W[m])
            self.__classifiers__.append(base)

            #predict through the m-th classifier
            y_predict = base.predict(X)
            #calculate error
            h = np.zeros(y_predict.shape)
            for i in range(0, y.shape[0]):
                if y_predict[i] != y[i]:
                    h[i] = 1
                else:
                    h[i] = 0
                h[i] = h[i] * W[m, i]
            epsilon = np.sum(h)

            #calculate alpha value
            self.__alpha__.append(0.5 * math.log(1 / epsilon - 1))

            #reach max number of classifiers or good enough
            if m >= self.__max_base__ - 1 or epsilon < 0.1:
                self.__n_base__ = m + 1
                break

            #calculate weights for the next classifier
            w = np.zeros([X.shape[0]])
            for i in range(0, X.shape[0]):
                w[i] = W[m, i] * math.exp(-self.__alpha__[m] * y[i] * y_predict[i])
            z = np.sum(w)
            for i in range(0, X.shape[0]):
                W[m+1, i] = w[i] / z

    def predict(self, X, threshold=0):
        #sum prediction of all classifiers by their alpha
        alpha = np.array(self.__alpha__)
        h = np.zeros([self.__n_base__, X.shape[0]])
        for m in range(0, self.__n_base__):
            h[m] = alpha[m] * self.__classifiers__[m].predict(X)
        return np.sum(h, axis=0)
```

2) train.py:

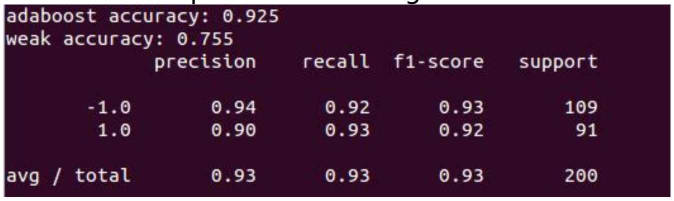
```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
import numpy as np
import os
import time
from scipy import misc
from feature import NPDFeature
import pickle
from ensemble import AdaBoostClassifier
from sklearn.metrics import classification_report

def rgb2gray(rgb):
    return np.dot(rgb[..., :3], [0.299, 0.587, 0.114])

if __name__ == "__main__":
    #load face data
    datafile = 'data'
    #data already preprocessed
    if os.path.exists(datafile):
        input = open(datafile, 'rb')
        X_train = pickle.load(input)
        X_vali = pickle.load(input)
        y_train = pickle.load(input)
        y_vali = pickle.load(input)
        input.close()
    #preprocess data
    else:
        facepath = 'datasets/original/face'
        nonfacepath = 'datasets/original/nonface'

        face = []
        nonface = []

        #for each image, convert it into grayscale presentation
        #scale to 24x24
        #and extract its NPD feature
        facedir = os.listdir(facepath)
        for i in range(0, len(facedir)):
            if facedir[i].endswith('jpg'):
                path = os.path.join(facepath, facedir[i])
                img = mpimg.imread(path)
                img = rgb2gray(img)
                img = misc.imresize(img, [24, 24])
                face.append(NPDFeature(img).extract())

        nonfacedir = os.listdir(nonfacepath)
        for i in range(0, len(nonfacedir)):
            if nonfacedir[i].endswith('jpg'):
                path = os.path.join(nonfacepath, nonfacedir[i])
                img = mpimg.imread(path)
                img = rgb2gray(img)
                img = misc.imresize(img, [24, 24])
                nonface.append(NPDFeature(img).extract())

        X = np.array(face + nonface)
        y = np.ones([1000])
        y[500:999] = -1

        X_train, X_vali, y_train, y_vali = train_test_split(X, y, test_size=0.2, random_state=24)
        output = open(datafile, 'wb')
        pickle.dump(X_train, output)
        pickle.dump(X_vali, output)
        pickle.dump(y_train, output)
        pickle.dump(y_vali, output)
        output.close()

    #create adaboost/weak classifier
    dtc = DecisionTreeClassifier(random_state=0,
        max_depth=3, max_features="sqrt")
    classifier = AdaBoostClassifier(dtc, 15)
    #train classifiers
    classifier.fit(X_train, y_train)
    dtc.fit(X_train, y_train)
    #do prediction
    result = classifier.predict(X_vali)
    weakresult = dtc.predict(X_vali)

    #calculate predicting accuracy for both
    adacount = 0
    weakcount = 0
    for i in range(0, result.shape[0]):
        if (np.abs(result[i]-1) < np.abs(result[i] + 1)):
            result[i] = 1
        else:
            result[i] = -1
        if result[i] == y_vali[i]:
            adacount = adacount + 1
        if weakresult[i] == y_vali[i]:
            weakcount = weakcount + 1
    print ("adaboost accuracy: " + str(adacount / result.shape[0]))
    print ("weak accuracy: " + str(weakcount / result.shape[0]))

    print(classification_report(y_vali, result))
```

IV. CONCLUSION

We combine 15 weak classifiers in this experiment.

The weak classifier is a decision tree with max depth assigned to 3, uses $\log_2$(#features) features to split.

Here is the experiment result gained:



```
adaboost accuracy: 0.925
weak accuracy: 0.755
             precision    recall  f1-score   support

       -1.0       0.94      0.92      0.93       109
        1.0       0.90      0.93      0.92        91

avg / total       0.93      0.93      0.93       200
```

Then we can draw a conclusion according to the experiment:

1) AdaBoost classifier gain much improvement against the original weak classifier.