华南理工大学

South China University of Technology

《机器学习》课程实验报告

学　　院　软件学院

专　　业　软件工程

组　　员　陈安妮、蔡天鑫、甄淑怡

学　　号　201530611142、201530611067、201530613771

邮　　箱　1412209084@qq.com

指导教师　吴庆耀

提交日期　2017 年 12 月 20 日

1. 实验题目:

基于 AdaBoost 算法的人脸分类

2. 实验时间：

2017 年 12 月 09 日

3. 报告人:

陈安妮、蔡天鑫、甄淑怡

4. 实验目的:

1.深入理解 Adaboost 的原理
2.熟悉人脸检测的基本方法
3.学会利用 Adaboost 解决人脸分类问题，将理论和实际工程接轨
4.体验机器学习的完整过程

5. 简述 Adaboost 原理：

AdaBoost，是英文"Adaptive Boosting"（自适应增强）的缩写，由 Yoav Freund 和 Robert Schapire 在 1995 年提出。它的自适应在于：前一个基本分类器分错的样本会得到加强，加权后的全体样本再次被用来训练下一个基本分类器。同时，在每一轮中加入一个新的弱分类器，直到达到某个预定的足够小的错误率或达到预先指定的最大迭代次数。

具体说来，整个 Adaboost 迭代算法就 3 步：

1、初始化训练数据的权值分布。如果有 N 个样本，则每一个训练样本最开始时都被赋予相同的权重：1/N。

2、训练弱分类器。具体训练过程中，如果某个样本点已经被准确地分类，那么在构造下一个训练集中，它的权重就被降低；相反，如果某个样本点没有被准确地分类，那么它的权重就得到提高。然后，权重更新过的样本集被用于训练下一个分类器，整个训练过程如此迭代地进行下去。

3、将各个训练得到的弱分类器组合成强分类器。各个弱分类器的训练过程结束后，加大分类误差率小的弱分类器的权重，使其在最终的分类函数中起着较大的决定作用，而降低分类误差率大的弱分类器的权重，使其在最终的分类函数中起着较小的决定作用。换言之，误差率低的弱分类器在最终分类器中占的权重较大，否则较小。

6. 数据集以及数据分析：

本实验提供 1000 张图片，其中 500 张是含有人脸的 RGB 图片，储存在./datasets/original/face 内；另外 500 张是不含有人脸的 RGB 图，储存在./datasets/original/nonface 内。通过 Image 中的 open 读取图片，将全部图片转成大小为 24*24 的灰度图，并将人脸图片标记为正样本（+1），非人脸图片标记为负样本（-1），再使用 feature.py 中 NPDFeature 类的方法提取特征。用 pickle 库中的 dump()函数将预处理后的特征数据保存到缓存中，之后使用 load()函数读取特征数据，再通过 train_test_split 将数据集随机分成 67%的训练集和 33%的

验证集。

7. 实验步骤:

1.读取数据集数据。读取图片，将全部图片转成大小为 24*24 的灰度图，数据集正负类样本的个数和比例不限，数据集标签形式不限。

2.处理数据集数据，提取 NPD 特征。使用 feature.py 中 NPDFeature 类的方法提取特征。(提示：因为预处理数据集的时间比较长，可以用 pickle 库中的 dump()函数将预处理后的特征数据保存到缓存中，之后可以使用 load()函数读取特征数据）

3.将数据集切分为训练集和验证集，本次实验不切分测试集。

4.根据 ensemble.py 中的预留的接口编写 AdaboostClassifier 所有函数。以下为 AdaboostClassifier 类中的 fit()方法的思路：

4.1 初始化训练集的权值 w,每一个训练样本被赋予相同的权值。

4.2 训练一个基分类器，基分类器可以使用 sklearn.tree 库中 DecisionTreeClassifier(注意训练的时候需要将权重 w 作为参数传入)。

4.3 计算基分类器在训练集上的分类误差率 ε 。

4.4 根据分类误差率 ε，计算参数 α。

4.5 更新训练集的权值 w。

4.6 重复以上 4.2-4.6 的步骤进行迭代，迭代次数为基分类器的个数。

5.用 AdaboostClassifier 中的方法在验证集上进行预测并计算精确率,并用 sklearn.metrics 库的 classification_report()函数将预测结果写入 report.txt 中。

6.整理实验结果并完成实验报告。

8. 代码内容:

feature.py

```
import numpy

class NPDFeature():

    __NPD_table__ = None

    def __init__(self, image):
        '''Initialize NPDFeature class with an image.'''
        if NPDFeature.__NPD_table__ is None:
            NPDFeature.__NPD_table__ = NPDFeature.__calculate_NPD_table()
        assert isinstance(image, numpy.ndarray)
        self.image = image.ravel()
        self.n_pixels = image.size
        self.features = numpy.empty(shape=self.n_pixels * (self.n_pixels - 1) // 2,
dtype=float)

    def extract(self):
```

```
        '''Extract features from given image.

        Returns:
            A one-dimension ndarray to store the extracted NPD features.
        '''
        count = 0
        for i in range(self.n_pixels - 1):
            for j in range(i + 1, self.n_pixels, 1):
                self.features[count] =
NPDFeature.__NPD_table__[self.image[i]][self.image[j]]
                count += 1
        return self.features

    @staticmethod
    def __calculate_NPD_table():
        '''Calculate all situations table to accelerate feature extracting.'''
        print("Calculating the NPD table...")
        table = numpy.empty(shape=(1 << 8, 1 << 8), dtype=float)
        for i in range(1 << 8):
            for j in range(1 << 8):
                if i == 0 and j == 0:
                    table[i][j] = 0
                else:
                    table[i][j] = (i - j) / (i + j)
        return table
```

ensemble.py

```
import pickle
import numpy as np

class AdaBoostClassifier:

    def __init__(self, weak_classifier, n_weakers_limit):
        self.weak_clf = weak_classifier #弱分类器的类
        self.limit = n_weakers_limit #弱分类器的最大数量
        self.G = [] #弱分类器的集合
        self.num = -1 #弱分类器的数量
        self.alpha = []
        self.validation_score_list = []

    def fit(self,X,y):
        self.W = np.ones(X.shape[0] )/X.shape[0] #初始化样本权值分布
        self.Score = np.zeros(X.shape[0])
```

```
            for i in range(self.limit):
                clf = self.weak_clf.fit(X,y,sample_weight = self.W)
                self.G.append(clf)
                P = clf.predict(X)    #采用弱分类器得到的分类结果
                error = np.sum((P != y.reshape(-1,)) * self.W)#分类误差率
                self.validation_score_list.append(1.0-error)
                #error = 1.0 - self.G[i].score(self.X,self.y)
                if error > 0.5:
                    continue
                elif error == 0:
                    break
                e = 0.5*np.log((1.0-error)/error)
                self.alpha.append(e)
                Z = np.multiply(self.W,np.exp(-self.alpha[i]* np.multiply(y.reshape(-1,) , P)))
#规范化因子
                self.W = Z/np.sum(Z)
            return self,self.validation_score_list

        def predict_scores(self, X):
            Score = np.zeros(X.shape[0])
            if self.num == -1:
                self.num = self.limit
            for i in range(self.num):
                Score += self.alpha[i] * self.G[i].predict(X).flatten(1)
            return Score

        def predict(self, X, threshold=0):
            predict_y = np.zeros(X.shape[0])
            Score = self.predict_scores(X)
            predict_y[np.where(Score > threshold)] = 1
            predict_y[np.where(Score <= threshold)] = -1
            return predict_y


        @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

```
#coding: utf-8
import os
import pickle
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.tree import DecisionTreeClassifier
from feature import NPDFeature
from ensemble import AdaBoostClassifier

def get_grayscale(path,x=None,y=None):
    for file in os.listdir(path):
        file_path = os.path.join(path,file)
        labels = None
        if path == 'datasets/original/face/':
            labels = [1]
        elif path == 'datasets/original/nonface/':
            labels = [-1]
        image = Image.open(file_path).convert('L').resize((24,24)) #将 rgb 图片转化成灰
度图
        if (x is None) & (y is None):
            x=np.array([NPDFeature(np.asarray(image)).extract()])
            y=np.array([labels])
        else:
            #把脸的和非脸的图片拼接起来
            x=np.vstack((x,NPDFeature(np.asarray(image)).extract()))
            y=np.vstack((y,labels))
    return x,y

def pre_image():
    x,y = get_grayscale('datasets/original/face/')
    x,y = get_grayscale('datasets/original/nonface/',x,y)
    print(x.shape,y.shape)
    with open('features', "wb") as file:
        pickle.dump(x, file)
    with open('labels', "wb") as file:
        pickle.dump(y, file)

def acc_plot(validation_score_list):
    plt.title('Adaboost')
```

```
        plt.xlabel('Iteration')
        plt.ylabel('Accuracy')
        plt.plot(range(len(validation_score_list)),validation_score_list)
        #plt.grid()
        plt.show()

    if __name__ == "__main__":
        # write your code here
        pre_image()
        with open('features', "rb") as f:
            x = pickle.load(f)
        with open('labels',"rb") as f:
            y = pickle.load(f)
        X_train,X_test,y_train,y_test = train_test_split(x,y,test_size=0.33,random_state=0)
        maxIteration = 10
        s,validation_score_list =
AdaBoostClassifier(DecisionTreeClassifier(max_depth=3),maxIteration).fit(X_train, y_train)
        predict_y = s.predict(X_test)

        acc_plot(validation_score_list)

        with open('report.txt', "wb") as f:
            report = classification_report(y_test,predict_y,target_names=["face","nonface"])
            f.write(report.encode())
```

9. 实验结果和曲线图:

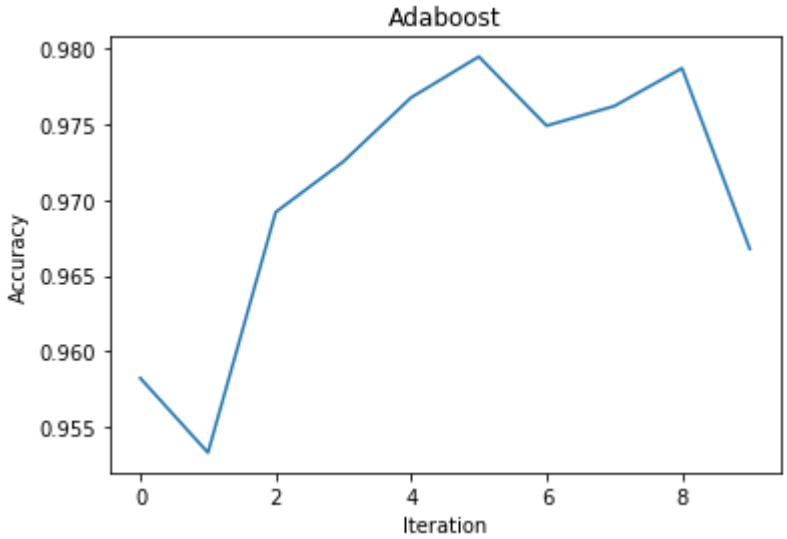
超参数选择（弱分类器个数、决策树深度等）：

n_weakers_limit=10，
max_depth=3，
test_size=0.33,
random_state=0

预测结果（最佳结果）：

	precision	recall	f1-score	support
face	0.86	0.78	0.82	167
nonface	0.79	0.87	0.83	163
avg / total	0.83	0.82	0.82	330

精度曲线图：



10.实验结果分析：

在本次实验中，采用 Image 中的 open 读取 500 张人脸图片和 500 张不含人脸图片，再将其转成大小为 24*24 的灰度图，并将人脸图片标记为正样本（+1），非人脸图片标记为负样本（-1）。使用 feature.py 中 NPDFeature 类的方法提取特征，用 pickle 库中的 dump()函数将预处理后的特征数据保存到缓存中，之后使用 load()函数读取特征数据，再通过 train_test_split 将数据集随机分成 67%的训练集和 33%的验证集。使用 sklearn.tree 库中 DecisionTreeClassifier 作为基分类器，将前一个基本分类器分错的样本权值加强，加权后的全体样本再次被用来训练下一个基本分类器。同时，在每一轮中加入一个新的弱分类器，直到达到预先指定的最大迭代次数，并将各个训练得到的弱分类器组合成强分类器。由实验结果分析可知：

1. 划分的训练集越大，训练样本越多，得到的强分类器的分类效果越好，分类的准确率越高；
2. 在本次实验的数据集中，最佳的弱分类器个数是 10、决策树深度是 3.弱分类器的数目太少的话，分类器的准确率较低，分类器的数目太多的话，容易造成过拟合，分类器的准确率也会下降。同理，决策树的深度为 3 是最合适的；
3. 由于划分训练集和验证集是随机的，所以分类器的测试结果结果也是不确定的，但是测试结果相对集中，波动不大。

11.实验总结：

本次实验使我们更加深入的理解了 Adaboost 的原理，即选定一个弱分类器作为基分类器，并对样本进行训练。将前一个基分类器分错的样本权值加强，加权后的全体样本再次被用来训练下一个基分类器。同时，在每一轮中加入一个新的

弱分类器，直到达到某个预定的足够小的错误率或达到预先指定的最大迭代次数，并将各个训练得到的弱分类器组合成强分类器，从而获得准确率较高的分类器。

此外，我们还学习和熟悉了人脸检测的基本方法，就是先将待检测的图片转化为 24*24 的灰度图，再提取 NPD 特征（利用归一化的像素差异特征 Normalized PixelDifference)，将图片信息转变成数据矩阵。由于人脸的 NPD 特征与非人脸的 NPD 特征存在较大差异性，所以可以利用 NPD 特征判断出图片是否为人脸图片，从而达到人脸检测的目的。

在本次实验中，我们将理论和实际工程接轨，利用 Adaboost 算法对人脸和非人脸的 NPD 特征进行训练，得到准确率较高的人脸和非人脸的分类器，从而解决了人脸分类问题，体验机器学习的完整过程，收获颇大。

在这次实验中，我们也遇到了一些难题，例如参数的选择，内存错误，以及一些乱七八糟的 bug，所幸的是我们基本上都解决了，但是程序总体的运行时间还是比较久的，这点有待改进。