The Experiment Report of Machine Learning

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
Xia Shu, Liu Fen and LiuRui

Supervisor:
Qingyao Wu

Student ID：
201721046227 201721046111 201721045626

Grade:
Postgraduate

December 20, 2017

Face Classification Based on AdaBoost Algorithm

Abstract—In this experiment we further understand the Adaptive Boosting algorithm, a famous unconstrained face detecting algorithm which is short as AdaBoost. We putting the theory into experiment and using AdaBoost to solve the problem of classifying face pictures and non-face pictures, learn from the whole machine learning process.

I. INTRODUCTION

AdaBoost, short for Adaptive Boosting, is a machine learning meta-algorithm. The objective of AdaBoost is to find and locate faces in an image. It is the first step in automatic face recognition applications.

AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. AdaBoost is sensitive to noisy data and outliers. In some problems it can be less susceptible to the overfitting problem than other learning algorithms. The individual learners can be weak, but as long as the performance of each one is slightly better than random guessing, the final model can be proven to converge to a strong learner.

In this experiment, we use the provided pictures dataset, of which 500 pictures are human face RGB images and the other 500 are non-face RGB images. Here are the experiment steps:

1） 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.

2） 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.)

3） The data set is divided into training set and validation set, this experiment does not divide the test set.

4） Write all AdaboostClassifier functions based on the reserved interface in ensemble.py. The following is the guide of fit function in the AdaboostClassifier class:
4.1 Initialize training set weights , each training sample is given the same weight.
4.2Training a base classifier , which can be sklearn.tree library DecisionTreeClassifier (note that the training time you need to pass the weight as a parameter).
4.3 Calculate the classification error rate of the base classifier on the training set. 4.4 Calculate the parameter according to the classification error rate .
4.5 Update training set weights .
4.6 Repeat steps 4.2-4.6 above for iteration, the number of iterations is based on the number of classifiers.

5） 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 .

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

We introduce the methods and theory in part II, and show our experiment results in part III, at last we made some conclusion in part IV.

II. METHODS AND THEORY

A. Normalized Pixel Difference Feature(NPD)

The NPD feature measures the relative difference between two pixel-values. The sign of f (x, y) indicates the ordinal relationship between the two pixels x and y, and the magnitude of f (x, y) measures the relative difference (as a percentage of the joint intensity x + y) between x and y.

The NPD feature between two pixels in an image is defined as

$$f(x,y) = \frac{x-y}{x+y}$$

where x, y $\geqslant$ 0 are intensity values of the two pixels1, and f(0, 0) is defined as 0 when x = y = 0.

B. AdaBoost Classifier

AdaBoost refers to a particular method of training a boosted classifier. Each boosted classifier is a weak learner that learner that taker an object as input and returns a value indicating the class of the object. At each iteration a weak learner is selected and assigned a coefficient alpha such that the sum training error of the resulting boost classifier is minimized.

At each iteration of the training process, a weight is assigned to each sample in the training set equal to the current error. These weights can be used to inform the training of the weak learner.

For very iteration feneratesn a new base learner $h_m(X)$ and its importance score α_m has

$$H(x) = sign(\sum_{m=1}^{M} \alpha_m h_m(x))$$

This nonlinear function makes adaboost able to solve nonlinear problems. There are m base learners, and for each base classifier h_m

$$h_m(\mathbf{x}) : \mathbf{x} \mapsto \{-1, 1\}$$

Suppose that the error rate of the base learners are independ to each other and express as

$$\epsilon_m = p(h_m(\mathbf{x}_i) \neq y_i) = \sum_{i=1}^{n} w_m(i)\mathbb{I}(h_m(\mathbf{x}_i) \neq y_i)$$

When we make the ϵ_m lower to zero we can get

$$\alpha_m = \frac{1}{2}\log\frac{1-\epsilon_m}{\epsilon_m}$$

So, we can conclude Adaboost algorithm by the pseudocode as follow.

```
Input: $D = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$, where $\mathbf{x}_i \in X, y_i \in \{-1, 1\}$
Initialize: Sample distribution $w_m$
Base learner: $\mathcal{L}$
$w_1(i) = \frac{1}{n}$
for m=1,2,...,M do
    $h_m(x) = \mathcal{L}(D, w_m)$
    $\epsilon_m = \sum_{i=1}^{n} w_m(i)\mathbb{I}(h_m(\mathbf{x}_i) \neq y_i)$
    if $\epsilon_m > 0.5$ then
        break
    end
    $\alpha_m = \frac{1}{2}\log\frac{1-\epsilon_m}{\epsilon_m}$
    $w_{m+1}(i) = \frac{w_m(i)}{z_m}e^{-\alpha_m y_i h_m(\mathbf{x}_i)}$, where $i = 1, 2, ..., n$ and
      $z_m = \sum_{i=1}^{n} w_m(i)e^{-\alpha_m y_i h_m(\mathbf{x}_i)}$
end
Output: $H(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m h_m(\mathbf{x})$
```

C. Decision Tree Classifier

AdaBoost with decision trees as the weak learners is often referred to as the best out-of-the-box classifier. When used with decision tree learning, information gathered at each stage of the AdaBoost algorithm about the relative 'hardness' of each training sample is fed into the tree growing algorithm such that later trees tend to focus on harder-to-classify examples.

Tree models where the target variable can take a discrete set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels.

III. EXPERIMENT

Experimental Code

```
def dealimage(img_path):
    print("dealimage")
    #img_path = './datasets/original/nonface'
    image_list = [os.path.join(img_path, f) for f in os.listdir(img_path)]
    img = []
    for i in range(len(image_list)):
        img_temp = Image.open(image_list[i]).convert('L')
        img_temp.thumbnail((24, 24))
        img.append(np.array(img_temp))
return img
```

Convert the image into grayscale in size 24*24:

```
def extractfeature(img_face,filename):
    feature_list = []
    for i in range(len(img_face)):
        f = NPDFeature(img_face[i])
        f.extract()
        feature_list.append(f.features)
      output = open(filename, 'wb')
    pickle.dump(feature_list, output)
 print("dump over")
```

Processing data set data to extract NPD features:

```
def shuffle_array(X, y):
    randomlist = np.arange(X.shape[0])
    np.random.shuffle(randomlist)
    X_random = X[randomlist]
    y_random = y[randomlist]
return X_random, y_random
```

Mix the data of face and nonface:

```
if os.path.exists('face_feature'):
        print("get face_feature")
    else:
        print("don't found face_feature")
        img_face = dealimage('./datasets/original/face')
        extractfeature(img_face,'face_feature')

    if os.path.exists('nonface_feature'):
        print("get nonface_feature")
    else:
        print("don't found nonface_feature")
        img_nonface = dealimage('./datasets/original/nonface')
        extractfeature(img_nonface,'nonface_feature')
```

Check the feature file:

```
input = open('nonface_feature', 'rb')
nonface_data = pickle.load(input)
X = nonface_data
y = np.ones(len(X))
y = y * (-1)
input = open('face_feature', 'rb')
face_data = pickle.load(input)
face_y = np.ones(len(face_data))
X.extend(face_data)
y = np.append(y, face_y)
X = np.array(X)

X, y = shuffle_array(X, y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

abc = AdaBoostClassifier(DecisionTreeClassifier(), 20)
abc.fit(X_train, y_train)

y_test_predict = abc.predict(X_test, 0)
target_names = ['face', 'nonface']
print(classification_report(y_test, y_test_predict, target_names=target_names))
with open('report.txt', 'w') as f:
f.write(classification_report(y_test, y_test_predict, target_names=target_names))
```

Test the AdoboostClassifier:

```
def fit(self, X, y):
    w = np.ones(X.shape[0])
    w = w * (1 / X.shape[0])

    for i in range(self.n_weakers_limit):
        print(i)
        basicclassfier = DecisionTreeClassifier(max_depth=1)
        basicclassfier.fit(X, y, sample_weight=w)
        self.classifier_list.append(basicclassfier)
        y_predict = basicclassfier.predict(X)
        epsilon = np.sum(w[y != y_predict])
        if epsilon > 0.5:
          break
        self.alpha[i] = 0.5*np.log((1-epsilon)/epsilon)
        temp = w * np.exp((-1) * y * y_predict )
        z = np.sum(temp)
        w = temp / z
```

The fit function of AdaBoostClassifier:

```
def predict_scores(self, X):
      h = []
      score = np.zeros(X.shape[0])
      for i in range(self.n_weakers_limit):
        h.append(self.classifier_list[i].predict(X))
        print(h[i])
        score += self.alpha[i] * h[i]
 return score
```

The predict_scores function of AdaBoostClassifier:

```
def predict(self, X, threshold=0):
    score = self.predict_scores(X)
    score[score > threshold] = 1
    score[score < threshold] = -1
  return score
```

The predict function of AdaBoostClassifier:

Result

	precision	recall	fl-score	support
face	0.98	0.96	0.97	101
nonface	0.96	0.98	0.97	99
avg/total	0.97	0.97	0.97	200

IV. CONCLUSION

1) In this experiment we have further understand the Adaptive Boosting algorithm. In addition, we learned the basic dealing of image and the process of extracting NPD feature.
2) Adaboosting can construct a strong integration base on some learners which are very weak in generalization.
3) Increasing the number of weak classifiers in a certain scope can improve the classification accuracy.