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:
Zijie Shu
Wenjie Pan
Qiaoyuan Wen

Supervisor:
Qingyao Wu

Student ID:

201530612712

201530612590

201530612989

Grade:

Undergraduate

December 21, 2017

Face Classification Based on AdaBoost Algorithm

Abstract—Boosting is a machine learning ensemble meta-algorithm for primarily reducing bias, and also variance in supervised learning, and a family of machine learning algorithms which convert weak learners to strong ones. AdaBoost is a machine learning meta-algorithm. It can be used in conjunction with many other types of learning algorithms to improve performance. AdaBoost (with decision trees as the weak learners) is often referred to as the best out-of-the-box classifier. In this experiment, we use Adaboost to solve the face classification problem that we need to classify the datasets with 1000 pictures into two classes: face and non-face. Finally, the average classification accuracy.

I. Introduction

AdaBoost, short for Adaptive Boosting, is a machine learning meta-algorithm formulated by Yoav Freund and Robert Schapire, who won the 2003 Gödel Prize for their work. It can be used in conjunction with many other types of learning algorithms to improve performance. The output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier. 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.

Every learning algorithm tends to suit some problem types better than others, and typically has many different parameters and configurations to adjust before it achieves optimal performance on a dataset, 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.

II. Methods and Theory

Problems in machine learning often suffer from the curse of dimensionality — each sample may consist of a huge number of potential features (for instance, there can be 162,336 Haar features, as used by the Viola–Jones object detection framework, in a 24×24 pixel image window), and evaluating every feature can reduce not only the speed of classifier training and execution, but in fact reduce predictive power, per the Hughes Effect. Unlike neural networks and SVMs, the AdaBoost training process selects only those features known to improve the predictive power of the model, reducing dimensionality and potentially improving execution time as irrelevant features need not be computed.

Ensemble learning is combining numerous weak learners to a strong learner. Its main methods are boosting and bagging. Adaboost makes the wrong predictive samples more important, and handles it in next round.

$$w_{m+1}(\mathrm{i}) = \frac{w_m(\mathrm{i})}{z_m} e^{-\alpha_m y_i h_m(x_i)}$$

$z_m = \sum_{i=1}^{n} w_m(i)\, e^{-\alpha_m y_i h_m(x_i)}$ is normalization term, makes $w_m(i)$ become probability distributions. So in next round, $\frac{w_{wrong}(i)}{w_{right}(i)} = e^{2\alpha_m} = \frac{1-\epsilon_m}{\epsilon_m}$ and $\epsilon_m < 0.5$, wrong samples will be more important.

Every iteration generates a new base learner $h_m(x)$ and its importance score α_m.

Base learner:

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

Error rate:

$$\epsilon_m = p(h_m(x_i) \neq y_i) = \sum_{i=1}^{n} w_m(\mathrm{i})\, I(h_m(x_i) \neq y_i)$$

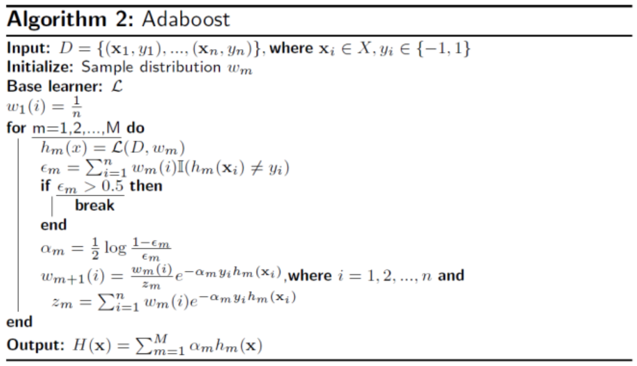
Make the base learner with lower ϵ_m more important:

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

Final learner:

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

Note: $h_m(x) = \mathrm{sign}(w^T x)$ is a nonlinear function, so the Adaboost can deal with nonlinear problem. The algorithm of Adaboost is shown in the figure below.



```
Algorithm 2: Adaboost
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: $\mathcal{L}$
1  $w_1(i) = \frac{1}{n}$
2  for m=1,2,...,M do
3      $h_m(x) = \mathcal{L}(D, w_m)$
4      $\epsilon_m = \sum_{i=1}^{n} w_m(i)\mathbb{I}(h_m(\mathbf{x}_i) \neq y_i)$
5      if $\epsilon_m > 0.5$ then
6          break
7      end
8      $\alpha_m = \frac{1}{2}\log\frac{1-\epsilon_m}{\epsilon_m}$
9      $w_{m+1}(i) = \frac{w_m(i)}{z_m} e^{-\alpha_m y_i h_m(\mathbf{x}_i)}$, where $i = 1, 2, ..., n$ and
         $z_m = \sum_{i=1}^{n} w_m(i) e^{-\alpha_m y_i h_m(\mathbf{x}_i)}$
10 end
Output: $H(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m h_m(\mathbf{x})$
```

III. Experiment

Dataset:

This experiment provides 1000 pictures, of which 500 are human face RGB images, stored in datasets/original/face; the other 500 is a non-face RGB images, stored in datasets/original/nonface. We downloaded it and divided it into 750 samples of training set and 250 samples of validation set after mixing up the dataset.

Figure 1 face and non-face datasets

Experiment Step:

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 divisded into training set and calidation 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.

4.2 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).

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 .

Main code:

```
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).
    '''
    num = X.shape[0]
    w = np.ones(num)/num
    if not os.path.exists('base_classifier'):
        os.mkdir('base_classifier')
    for i in range(self.n_weakers_limit):
        print('training base classifier %d/%d' %
(i+1,self.n_weakers_limit))
        # Configure Decision tree
        b_learner =
self.weak_classifier(random_state=0, max_depth=2)
        #Build Decision tree according training data
        b_learner.fit(X, y, sample_weight=w)
        # Save Model to a file in order to use it next
time without build model step

self.save(b_learner,'base_classifier/base_classifi
er_%d.pkl' % i)

        pre_y = b_learner.predict(X)
        errorVector = np.zeros(num)
        errorVector[pre_y != y] = 1
        errorRate = np.dot(errorVector,w)
        if errorRate > 0.5:
            break
        alpha = 0.5 * np.log(( 1 - errorRate ) /
errorRate)
        self.alphas[i] = alpha
        w = w * np.exp( - alpha * y * pre_y )
        w = w / w.sum()

    print('training finish')
```

Result:

```
predicted results: [ 1  1  1  1  1 -1  1 -1 -1  1 -1  1 -1 -1  1  1  1  1
1  1 -1 -1 -1  1  1 -1  1  1 -1  1 -1 -1  1 -1 -1 -1 -1 -1 -1  1  1  1 -1
1  1  1 -1 -1  1 -1 -1  1  1  1 -1  1  1 -1  1 -1 -1  1  1  1 -1  1 -1  1
-1  1  1 -1  1  1 -1 1  1  1  1 -1  1  1 -1 -1 -1 -1 -1  1 -1  1  1 -1  1
1  1 -1  1  1  1 -1 -1  1  1 -1  1  1 -1 -1  1 -1 -1 -1 -1  1 -1  1 -1  1
-1 -1 -1 -1 -1  1  1 1 -1 -1 -1 -1 -1 -1 -1  1  1  1 -1 -1 -1  1  1  1  1
-1 -1 -1  1  1  1 -1 1 -1 -1 -1 -1 -1 -1  1  1  1 -1 -1  1  1  1  1 -1 -1
-1 -1  1  1 -1  1  1  -1  1 -1  1  1 -1  1 -1 -1 -1 -1  1  1 -1 -1 -1 -1 -1
1 -1  1  1  1  1 -1 -1  1 -1 -1  1  1  1  1  1 -1 -1  1 -1  1  1  1  1 -1
-1  1  1  1 -1 -1 -1 1 -1 -1  1 -1 -1  1 -1 -1  1 -1  1 -1  1 -1 -1  1  1
1 -1  1  1 -1  1 -1]
accuracy: 0.952000
```

Report:

```
             precision    recall  f1-score   support
        face      0.97      0.94      0.95       127
     nonface      0.94      0.97      0.95       123
```

avg / total	0.95	0.95	0.95	250

Figure 2 result of 250 samples of validation datasets

IV. CONCLUSION

Through this experiment,we understand Adaboost further,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 and experience the complete process of machine learning