South China University of Technology

The Experiment Report of *Machine Learning*

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
Jieqiong Liu

Supervisor:
Mingkui Tan

Student ID:
201830661465

Grade:
Undergraduate

November 28, 2020

Face Detection based on AdaBoost Algorithm

***Abstract*—In this experiment we used models based on Adaboost algorithm to deal with face detection. We learned how to apply a machine learning algorithm to practical fields and experienced the whole process of a machine learning project.**

I. Introduction

FACE detection is one of the most classical computer vision problem which requires to identify specific human faces from a surrounding environment. Other advanced tasks in this field include face alignment, face classification, and faces identification

II. Methods and Theory

A. Introduction to AdaBoost algorithm

AdaBoost (Adaptive Boosting) is one of ensemble methods which aims to improve Boosting algorithm. Due to simple learners' limited ability to deal with relative complex problems, to achieve a better performance, Adaboost combines basic learner in a linear form.

In AdaBoost algorithm, there are 2 weights: weight of each data point (wi) and weights of weak learners (a), and the algorithm updates these parameters in an iterative manner.Following are steps in each iteration.

In the beginning, all the data points share the same weight $w_1(i)$.

$$w_1(i)\frac{1}{n}, i = 1, 2, ..., n$$

In which n represents the number of data points. Then keep iterating the following steps:

1) Fit the base learner $h_t(x)$ to data points The prediction result of $h_t(x)$ is:

$$h_t(x) \in -1, +1$$

2) According to each data point's classification, calculate the error rate e_t of $h_t(x)$:

$$e_t = p(h_t(x) \neq y_i) = \Sigma_{i=1}^{n} w_t(i) I(h_t(x_i) \neq y_i)$$

Where w_i is the weight of each data point. And:

$$I(X = x_i) = \begin{cases} 1, & h_t(x_i) \neq y_i \\ 0, & h_t(x_i) = y_i \end{cases}$$

3) Update the model weight of h_t Then we can calculate the weight at of ht(x) in the current classifier.

$$\alpha_t = \frac{1}{2} ln \frac{1 - e_t}{e_t}$$

When e_t is smaller, at becomes larger. Because if we draw the function of e_t and α_t, we get curve like this:

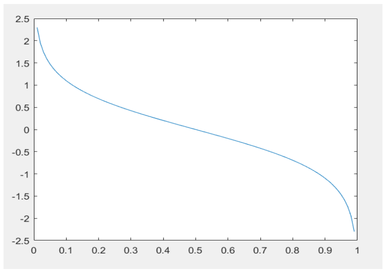


Fig. 1: function curve

We can also explain this characteristic in another way: When the error rate of a base learner becomes low, it means that the learner are better fit than others. As a result, we want this learner has a larger weight in the final decision so the final result will be improved.

4) Update the weight of each data point

$$w_{t+1}(i) = \frac{w_t(i)}{z_t} e^{-\alpha_t y_i h_t(x_i)}$$

And z_t in this formula is calculated as:

$$z_t = \Sigma_{i=1}^{n} w_t(i) e^{-\alpha_t y_i h_t(x_i)}$$

Which is a normalization term in order to keep wi in range(0,1) and makes $w_t(i)$ a probability distribution. For classifiers, it is easy to find that $y_i h_t(x_i)$ is 1 or -1, thus we simplify the representation as

$$w_{t+1}(i) = \begin{cases} \frac{w_t(i)}{z_t} e^{-\alpha_t}, & \text{right prediction} \\ \frac{w_t(i)}{z_t} e^{\alpha_t}, & \text{wrong prediction} \end{cases}$$

After each iteration, the model use a weighted vote for each weak classifier to output a final result. If the result is smaller than the threshold value, then the training process will be ended and we get the final AdaBoost Model, which is the following formula.

$$H(x) = sign(\Sigma_{t=1}^{T} \alpha_t h_t(x))$$

In this formula, the sign() means the sign function, α_t and $h_t(x)$ represent the weight of the t_{th} learner and the t_{th} learner model.

To conclude, Adaboost moderates the error rate according to the response from weak classifier's learning and can achieve a relative good prediction.

B. ***NPD feature and Naar feature***

1) *NPD feature:*

According to the paper [1], the Gentle AdaBoost algorithm is trained to select the most discriminative features of NPD based deep quadratic trees. NPD Feature (Normalized Pixel

Difference feature) between two pixels in an image is used to measure the relative difference, is defined as follows:

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

x, y are intensity values of the 2 pixels and $f(0,0)$ is 0, when $x = y = 0$.

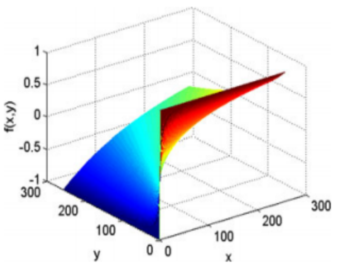


Fig. 2: NPD function plot

The sign of NPD features indicates the ordinal relationship between x and y, the magnitude of NPD measures the difference. NPD has many advantages over other features:

- NPD feature is antisymmetric, thus leads to a limited feature space. A $s * s$ image which is vectorized as $p * 1.(p = s * s)$ has only $p(p-1)/2$ NPD features.
- The sign NPD features which indicates the ordinal relationships between pixels is of great use for image processing, since the ordinal relationships remain the same under different situations.
- NPD feature is scale invariant. This characteristic contribute to a resistance to illumination change.
- NPD feature is bounded in [-1,1].which makes it easy to be implemented in other algorithms.

As following figures illustrates, four NPD features and automatically selected in the process of learning, and then combined in a deep quadratic tree for face prediction.

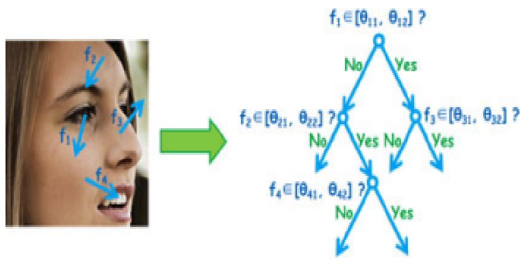

f_2
f_3
f_1
f_4
$f_1 \in [\theta_{11}, \theta_{12}]$?
No
Yes
$f_2 \in [\theta_{21}, \theta_{22}]$?
$f_3 \in [\theta_{31}, \theta_{32}]$?
No
Yes
No
Yes
$f_4 \in [\theta_{41}, \theta_{42}]$?
No
Yes

Fig. 3: Use of NPD features

In conclusion, the NPD feature is efficient to compute and makes it convenient to reconstruct the original picture.

2) Naar feature:
In the face detection task we use OpenCV library with Naar feature encapsulated. Naar-like feature is also called Haar feature, which is one common operator in Computer vision. There are 14 features in OpenCV, which are shown as follows. These features can be divided into 3 main categories: edge features, line features and center-surround features. These

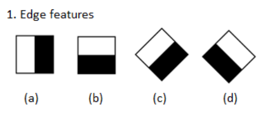


Fig. 4: edge feature

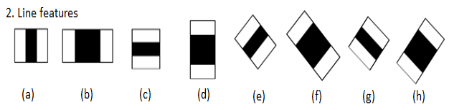


Fig. 5: line feature

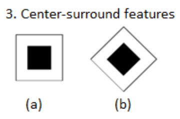


Fig. 6: center feature

features can represent the change of a grayscale image. A feature template is composed of black and white rectangles and the feature value is computed as the difference between the sum of black pixels and the white pixels. The feature is sensitive to pixel distribution which changes in a specific direction, for example, the shadow around eyes or mouth on a face.

$$featureValue(x) = w_{white}\Sigma_{p \in white}p - w_{black}\Sigma_{p \in black}p$$

Among which w is the weight of pixels.

III. Experiments

A. *Dataset*

We used 1000 pictures provided for training and testing, of which 500 are RGB pictures containing human faces and another 500 are RGB pictures without human faces. Divided the dataset into training set and validation set by a ratio of 4:1.

Listing 1: **divide.py**

```
# devide the dataset into traning set and
    validation set
from sklearn.model_selection import train_test_split

X_train, X_val, y_train, y_val = train_test_split (X, y,
    test_size=0.25)
print (y_train.shape)
```

B. *Implementation*

1) Face Classification:
My work includes the following:

- Data Loading
  These figures are changed into 24 *24 and are all grayscale figures.
- Data Preprocessing
  We extract the NPD features by functions of class NPDFeature in feature.py. Additionally, due to the long

processing time of data preprocessing, we used dump() function from pickle library and cached it into the memory. In this way we will be able to read data by function load() quickly.

Listing 2: **feature.py**

```
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
```

The dump function is used as follows: The pickle module implements binary protocols for serializing and deserializing a object structure.
Pickling is the process when an object hierarchy is converted into a byte stream, and unpickling is the inverse operation.To serialize an object hierarchy, we call the function dumps() and to de-serialize a data stream, we use loads() function.

Listing 3: **dump.py**

```
if dataset_dump_file not in glob.glob(os.getcwd() +
    r'\*'):
    D_face = load_img(os.getcwd() + r'\datasets\
        original\face', 1)
    D_nonface = load_img(os.getcwd() + r'\
        datasets\original\nonface', −1)
    D = np.vstack((D_face, D_nonface))
    with open(dataset_dump_file, 'wb') as f:
        # pickle library dump() function
            into the memory
        #use load() function to process
            data
pickle.dump(D, f, pickle.HIGHEST_PROTOCOL)
```

- Encode the AdabootClassifier.py
  The times of iteration is set to the the number of base classifiers. The whole process can be listed as follows:
  - Initialize the weights of training set, namely, to give an average weight for each data point.
  - Train a base classifier, for simplicity we use DecisionTreeClissifier form sklearn.tree dirertly.
  - Calculate the error rate on training set and calculate the parameter a based on error rate. If the error rate is larger than 0.5 or the error rate is equal to 0.0, break current iteration.
  - Validate the model using validation set.

Listing 4: **AdaboostClassifier.py**

```
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).
        '''
        #the same weight at the
            beginning
        w = np.ones(y.shape)
        #normalization
        w = w/w.sum()
        self.a = []
        self.base_clfs = []

        for i in range(self.n_weakers_limit):
            base_clf=copy.copy(self.weak_clf)
            base_clf.fit(X, y.flatten(),w.flatten()
                )
            # train a basic classifier
            y_pred = base_clf.predict(X).reshape
                ((−1,1))
            #calculate the error rate
            err_rate = w.T.dot(y_pred!=y)[0][0]/w.
                sum()

            if err_rate > 1/2 or err_rate == 0.0:
                break

            weight_param_a = math.log((1−
                err_rate)/err_rate)/2

            self.base_clfs.append(base_clf)
            self.a.append(weight_param_a)
            #update the weight
            w = w * np.exp(−weight_param_a*y*
                y_pred)
            w = w/w.sum()
```

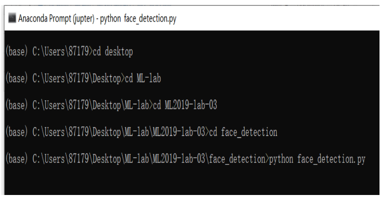
Call function in AdaboostClassifier.py to predict the facial detection and calculate the accuracy. Additionally, use classification_report() from sklearn.metrics to write the prediction results into a txt file named as classifier_report.txt.

Classification_report function accepts a series of prediction and groung truth values and returns a dictionary or string that contains information about precision, recall rate, f1-score and support value.

Listing 5: **report.py**

```
report_fp.write(' classification  report of  train  data:\n'
      )
report_fp.write( classification_report (y_true =
      y_train_true , y_pred = y_train_pred,
target_names = class_name)+'\n\n')
report_fp.write(' classification  report of  val  data:\n')
report_fp.write( classification_report (y_true =
      y_val_true, y_pred = y_val_pred,
target_names = class_name)+'\n\n')
```

2) *Face Detection:*
Experience the face detection function provided by OpenCV. Run face_detection.py and experience the facial detection method based on Haar features and AdaBoost algorithm encapsulated in OpenCV. The results is shown as follows, it can be seen that the face of Lenna has been clearly marked with a rectangular.



```
(base) C:\Users\87179>cd desktop

(base) C:\Users\87179\Desktop>cd ML-lab

(base) C:\Users\87179\Desktop\ML-lab>cd ML2019-lab-03

(base) C:\Users\87179\Desktop\ML-lab\ML2019-lab-03>cd face_detection

(base) C:\Users\87179\Desktop\ML-lab\ML2019-lab-03\face_detection>python face_detection.py
```

Fig. 7: face detection

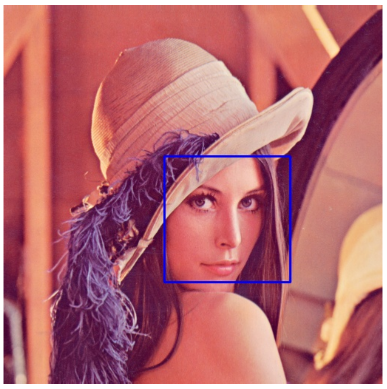
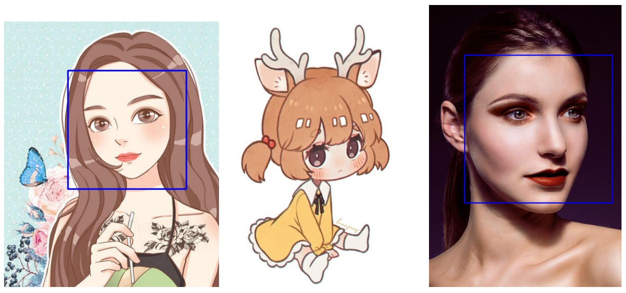


Fig. 8: Detection result



(a) successful (b) unsuccessful (c) successfull

Fig. 9: Detection results

Replace other images and tested the same function, the results are as follows. However, when I had a try with some other cartoon picture which contains a human face, it fails. Maybe other face detection networks which are especially built for cartoon characters will have a better performance.

C. *Results*

Following is the loss estimate of a single weak classifier. The exponential loss during is 0.7627 during training and validation process. The 01 loss during training and validation is 0.168 and 0.196 respectively.

TABLE I: Weak Classifier's report of training data

Class	Precision	Recall	F1-score	Support
Face	0.91	0.74	0.82	381
Nonface	0.78	0.92	0.84	369
Macro avg	0.84	0.83	0.83	750
Weighted avg	0.84	0.83	0.83	750

TABLE II: Weak Classifier's report of training data

Class	Precision	Recall	F1-score	Support
Face	0.91	0.66	0.76	119
Nonface	0.75	0.94	0.83	131
Macro avg	0.83	0.80	0.80	250
Weighted avg	0.82	0.80	0.80	250

Following is the loss estimate of an Adaboost classifier. The exponential loss during is 0.42115 during training and validation process. The 01 loss during training and validation is 0.023 and 0.052 respectively.

TABLE III: AdamBoost Classifier's report of training data

Class	Precision	Recall	F1-score	Support
Face	0.98	0.98	0.98	381
Nonface	0.98	0.98	0.98	369
Macro avg	0.98	0.98	0.98	750
Weighted avg	0.98	0.98	0.98	750

TABLE IV: AdaBoost Classifier's report of validation data

Class	Precision	Recall	F1-score	Support
Face	0.93	0.97	0.95	119
Nonface	0.97	0.93	0.95	131
Macro avg	0.95	0.95	0.95	250
Weighted avg	0.95	0.95	0.95	250

IV. Conclusion

From this experiment, I know the way AdaBoost algorithm is embedded into an architecture. Additionally, I become similar with face detection task and apply a network to solve practical problems. Moreover, I get to know more tasks like face detection, face classification, face identification, and image segmentation.

Firstly is the encoding with AdaBoost. I became more skilled at the encapsulation about Python and learned more about convenient libraries like dump() and classification_report(), which are of great help to our analysis of the results.

Secondly, by reading relative papers about NPD features and Haar features, I knew more about classic features involved in computer vision. NPD feature is extremely fast compared to other pixel features. Moreover, I experienced the face detection function provided by OpenCV, which is based on Haar features. The Haar features can be understood by examples like shadows and sharp edges on a face.

Finally, according to the paper, in the learning process the author applied a method called soft cascade. Soft cascade can not only contributes to more efficient training and early rejection of negative samples, but also can be seen as a single AdaBoost classifier with one exit per weak classifier.

In conclusion, the experiment equipped me with more knowledge about face detection and AdaBoost algorithm and aroused my interest in relative field.

V. References

[1] S. Liao, A. K. Jain and S. Z. Li, "A Fast and Accurate Unconstrained Face Detector," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 38, no. 2, pp. 211-223, 1 Feb. 2016, doi: 10.1109/TPAMI.2015.2448075.