Final Project
Zhang Yichi (CEG23079), Shi Fangze (CEG23046), Zhang Zhi (CEG23080)
Github: zyc123456zzz/ML_face_detector (github.com)

Problem Definition

Given a set of raw labelled faces which has been detected and centered by the Viola Jones face detector and collected from the web.[1] The dataset contains insufficient and sufficient faces labelled with their names. Our job is to clean the data, feature the faces and learn from training data so that we can give a name with respect to the input face. In this problem, we treat each name as a class, so our goal is to classify the faces to the right classes. Our classes are just about the existing name set, so we can not label the face does not appear in our training set as “other”.

For example:
Here we give a picture coming from the datasets.
Input:

Figure 1

Output:
Colin_Powell

Actual class: Colin_Powell
Result: True

Connection to Class Material

In this final project, we designed three different machine learning algorithms to process Labelled Faces in the Wild (LFW) Dataset, which is a database of face photographs designed for studying the problem of unconstrained face recognition. The data set contains multiple names and at least two 250x250x3 jpg images for each of them. Each student chooses a machine learning algorithm and implements or improves them to achieve the functionality of face recognition. Specifically, we train the model through a training set, and take the unlabeled pictures as the test set, expecting to get the real labels of the pictures, that is, the names corresponding to the face pictures. The models we selected include the non-parametric model KNN, the linear machine multiclass SVM and the bagging model Adaboost

Data Processing

In the first step, we retrieve names from the ‘./archive/lfw_allnames.csv’ and filter them based on a minimum of 100 faces per person. Subsequently, with the name list, RGB images were loaded using 'plt.imread'.

Then we extract features from the image. The feature extraction process involved flattening the image to capture color features, converting it to greyscale, and calculating Histogram of Oriented Gradients (HOG) features from the greyscale image. Then these color and HOG features were combined into a unified array. The feature matrix was then created by applying the 'create_features' function to each image in the dataset, summarizing the transformation of raw image data into a structured format suitable for analysis.

For data dimension reduction we use PCA to reduce memory usage and eliminate the effects of irrelevant noise. This method will be discussed in detail in the following section.

Other trick is used to replace the name labels with numbers. We encoded class labels (Y) using LabelEncoder from scikit-learn. Finally, using ‘train_test_split’, we split the training set and test set with the ratio of 75 : 25.

Data Dimension Reduction

Images are often represented by high-dimensional pixel values, making them computationally expensive and challenging for machine learning models. PCA addresses this by extracting a set of principal components that capture the essential information in the images.

Aiming in filtering out noise present in images, PCA ensures that the transformed images retain the essential structures while minimizing the impact of irrelevant details. Here is the comparison of different number of PC.

Number of Components	Accuracy		
Eigenfaces	SVM	SAMME	KNN
100	92.28%	76.41%	68.43%
500	80.70%	74.27%	66.77%
1000	92.63%	72.25%	66.77%

Algorithm:

Muti-class SVM Using One Against All

One-against-all SVM is based on binary SVM [2]. The concept is that, for each class, a binary SVM is constructed by treating it as the positive class and all samples of the remaining N-1 classes as the negative class.

The mathematical formulations for the SVM hinge loss function and parameter updates are as follows.

Loss function:

$$L(\mathbf{w}, b) = \frac{1}{N}\sum_{i=1}^{N} \max\left(0, 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + b)\right)$$

In this loss function, if a sample point is correctly classified, the loss is 0, otherwise the loss is $1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + b)$

Parameters update:

The mathematical formula for parameter updating, using gradient descent, is an update of the gradient of the loss function with respect to weight and bias.

For weight $\mathbf{w}$:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha\left(\lambda\mathbf{w} - \frac{1}{N}\sum_{i=1}^{N} 1\,(1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + b) > 0) \cdot y_i \cdot \mathbf{x}_i\right)$$

For bias $\mathbf{b}$:

$$b \leftarrow b - \alpha\frac{1}{N}\sum_{i=1}^{N} 1\,(1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + b) > 0) \cdot y_i$$

Results: The overall accuracy of our method is 92.28% while use SVM in sklearn the accuracy is 92.87%. We think we have achieved a good performance.

Multi-class Adaboost:

Adaboost:

It is a kind of ensemble methods, and the main idea of boosting is to train a strong classifier by combining weak classifiers. But boosting algorithm mainly focus on the two-class classification problem and it was firstly introduced by their Adaboost. Nevertheless, in going from two-class to multi-class classification, most boosting algorithms have been restricted to reducing the multi-class classification problem to multiple two-class problems.(one-others strategy)

SAMME:

According to [5],which develops a new algorithm that directly extends the AdaBoost algorithm to the multi-class case without reducing it to multiple two-class problems. So I implement this algorithm on our face classification problem. The pseudo code is shown below.

Algorithm *SAMME*

1. Initialize the observation weights $w_i = \frac{1}{n}$, $i = 1, 2, \ldots, n$
2. For $m = 1\ to\ M$:
   a) Fit a classifier $T^{(m)}(x)$ to the training data using weights w_i.
   b) Compute
   $$err^{(m)} = \frac{\sum_{i=1}^{n} w_i I(c_i \neq T^{(m)}(\boldsymbol{x}_i))}{\sum_{i=1}^{n} w_i}$$
   c) Compute
   i. $a^{(m)} = \log\frac{1-err^{(m)}}{err^{(m)}} + \log(K-1)$
   d) Set
   $$w_i \leftarrow w_i \cdot exp\big(a^{(m)} \cdot I(c_i \neq T^{(m)}(\boldsymbol{x}_i))\big),$$
   $$for\ i = 1, \ldots, n.$$
   e) Re-normalize w_i

3. Output

$$C(\boldsymbol{x}) = \arg\max_{k} \sum_{i=1}^{n} a^{(m)} \cdot I(k = T^{(m)}(\boldsymbol{x}))$$

Implementation:

Firstly, we use HOG to extract the features of the image and use PCA to reduce the dimension of the input image. Then the learning process starts, I turn from the simple adaboost[6] to the advanced multi-class adaboost(SAMME) to train the model.

Hyperparameter setting:

To ensure that we have sufficient training data, we try to have a higher upper bound of the number of the training data, which is depicted as min_faces_per_person.

min_faces_per_person = 80 (I do not tune this hyperparameter)

Numerical results:

To figure out the influence of the depth of decision tree used as weak classifier, I set the **max_depth** of the decision tree to be 1, 2, 3. The results show that, when **max_depth = 2**, the SAMME have the highest accuracy.(**74.27%**)

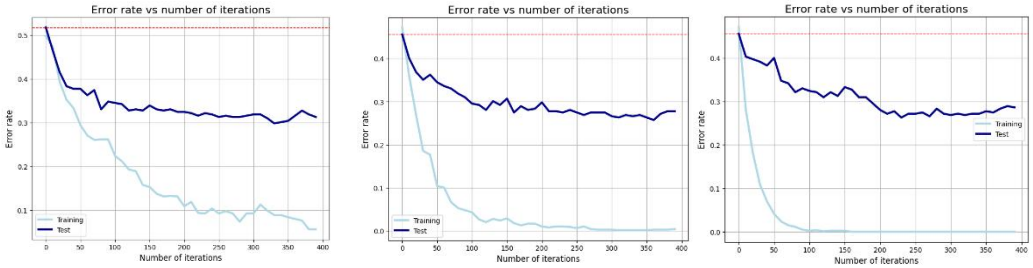


Figure 2. From left to right, max_depth = 1, 2, 3

To find the best feature choice, for different PCA **n_components** which includes “100”, “500”, “1000”, I individually train SAMME with different **M** ranging from 10~400, the stride is 10. (where max_depth = 2) We can see the dimension where we can get more profits is hard to find, but we can get a better result with **n_components<1000.** (The highest accuracy is **76.41%**)

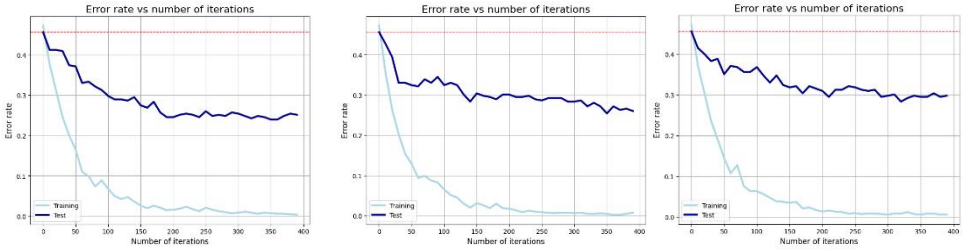


Figure 3. From left to right, n_components = 100, 500, 1000.

Compared with the results computed by sklearn.ensemble.AdaBoostClassifier. The results are not good as the algorithm above. The max accuracy is **67.25%**.

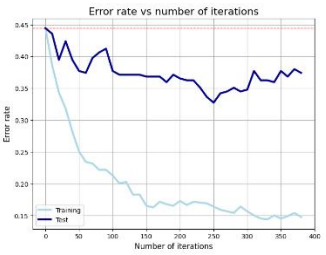


Figure 4. Error rate over all M

Insights:

The face pictures may have some irrelevant information about the human face which may result in bad performance in the training process.

I firstly apply the simple two-class Adaboost on this multi-class problem, but it ends with extremely low accuracy(about 20%), which is just like I train a one-other classifier, then I transform the two-class version to multi-class version following [5]. There is a problem about the result, the accuracy got from sklearn is lower than my algorithm. Actually, I do not know the exact reason behind this. I think it is because the standard algorithm stops when the test error reaches the plateau but I let the training process go on.

Weighted K-Nearest Neighbors Algorithm using k-d tree

The K-Nearest Neighbors Algorithm, also known as KNN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point [3]. It depends on the assumption that similar points can be found near one another.

In fact, we can think of the KNN algorithm as a kind of boost algorithm, in which each training sample can be regarded as a weak classifier. In general, in the boost algorithm, we need to use the weak classifiers voting method to segment the feature space obtained by PCA, so as to achieve the purpose of classification.

It is easy to argue that Random Forest is another effective approach. However, the performance of random forest without parameter tuning is not ideal, whose accuracy is only about 50%.

Why did this happen? Probably because of overfitting, but this also may be related to the distribution of the data set. After a rough analysis of the data, the following problems can be found: If the number of samples of a certain category is too large and the variance is high, the classifier will inevitably be more inclined to classify the samples waiting for classified into this category when using RF algorithm for classification task.

The solution of Adaboost is to assign a smaller weight to such a weak classifier and combine them to produce a strong classifier. However, considering a new sample to be classified, based on the above hypothesis, the weight of a classifier that is far away from it in feature space should approach 0. Setting the weight of some weak classifiers to 0 in advance can solve some problems of the data set well, such as not converging to a local optimal solution that is far away in the feature space (because they are excluded in advance).

However, general KNN also has some problems. For example, for the selected K nearest neighbor points, KNN assumes that they all contribute equally to the classifier. But this obviously results in the classifier being less robust, because the noise points and the real classes have the same weight in the classifier, even if these noise points are far apart in the feature space.

Therefore, assigning weights to K-nearest neighbor points is a good choice. Our goal is to give higher weights to samples that are closer together, so we can set the weights as follows:

$$w_n = \frac{d_n - d_{min}}{d_{max} - d_{min}} \quad , n \in [1, K]$$

For a point to be classified, the final result of the classifier C is:

$$C = \underset{C_i}{\operatorname{argmax}} \sum_{n=1}^{K} w_n \cdot \operatorname{sign}(s_n \epsilon C_i) \quad , \operatorname{sign}(b) = \begin{cases} 1, b = \text{true} \\ 0, b = \text{false} \end{cases}$$

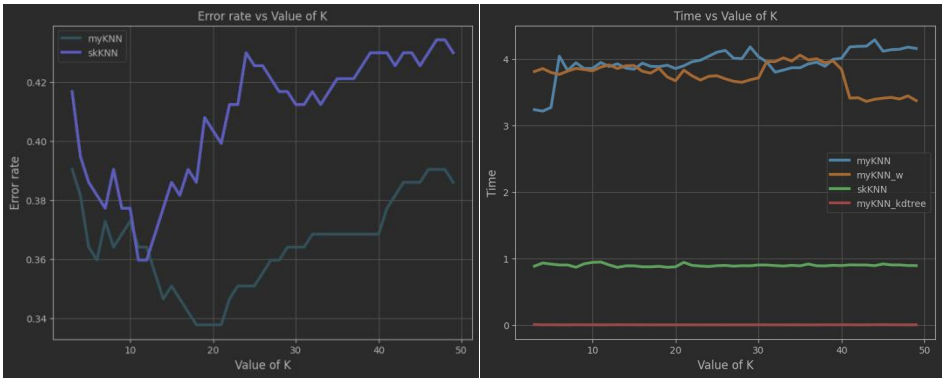
In order to improve the performance of KNN classifier, we can choose a data structure called k-d tree to store distance information on feature space and help us find K-nearest neighbors faster.

The k-d tree is a binary tree in which every node is a k-dimensional point. Every non-leaf node can be thought of as implicitly generating a splitting hyperplane that divides the space into two parts, known as half-spaces. Points to the left of this hyperplane are represented by the left subtree of that node and points to the right of the hyperplane are represented by the right subtree. The hyperplane direction is chosen in the following way: every node in the tree is associated with one of the k dimensions, with the hyperplane perpendicular to that dimension's axis. [4] With the k-d tree, only the O(logn) time overhead is required to query the K-nearest neighbor.

The use of k-d tree reflects the commonality of KNN and FLDA. The process of querying k-nearest neighbors can be thought of as a process of splitting feature Spaces using hyperplanes, although the "hyperplanes" used by KNN are rougher and non-parametric.

Results: The overall error of our method is 31.57% while use KNN in sklearn without parameter tuning the accuracy is 35.96%. We think we have achieved a good performance.

In terms of time overhead, we compare the direct K-nearest neighbor search method with the KNN method using k-d tree, and find that the method using k-d tree is much better than the direct search method in time, and is close to the method given by sklearn. Our results are as follows:



However, as a non-parametric method, KNN's results are obviously rough, and its results depend heavily on training samples, and is easier to overfit

References:

[1] http://vis-www.cs.umass.edu/lfw/

[2] C. W. Hsu and C. J. Lin. 2002. A comparison of methods for multi-class support vector machines. IEEE Trans. Neural Networks. 13, 2, 415-425.

[3] https://www.ibm.com/topics/knn

[4] https://en.wikipedia.org/wiki/K-d_tree
[5] Hastie T, Rosset S, Zhu J, et al. Multi-class adaboost[J]. Statistics and its Interface, 2009, 2(3): 349-360.
[6] https://github.com/jaimeps/adaboost-implementation