Project 1 : Face Classification and Detection

Han Bing
516030910523
mr.han@sjtu.edu.cn

June 2019

This project is to help us to have an deep insight of current models, including the intuition about the feature and the model, and the discrimination power of different models. The task is as following:

- Implement face classification using different models
- Implement face detection(Combining face classification and the sliding window)
- Visualize the face features
- Visualize models
- Write a report

1 Face classification Using Different Models

In the next subsection, I will introduce how I use the different models on Face classification task. And The results of models will also be given.

1.1 Preprocess(Extract Feature from Dataset)

The dataset we use is **FDDB:Face Detection Data Set and Benchmark**, and we can download it from Here.

This dataset includes 5171 faces in a set of 2845 images taken from the Faces in the Wild dataset. And there are ten annotation folders:FDDB-fold-01,...,FDDB-fold-10.

1.1.1 Data partitioning

The face images in the first eight folders as training samples and use face images in the last two folders as testing samples.

- Training set. First 8 folders.
- Testing set. Last 2 folders.

1.1.2 Generate positive and negative Images

Positive:For each images, there is an annotation to show the position of faces in this image. For each face, we need use a rectangle without any rotations to capture the face which is in the center of rectangle. This bounding box's scale is extended by $\frac{1}{3}$.

Negative:For each faces, we can generate eight negative images based on each face by sliding the bounding box by $\frac{1}{3}$. And we use face images in the first four folders to generate negative images for training, the last one folders for testing.

- Training set. 20680 samples.
- Testing set. 5203 samples.

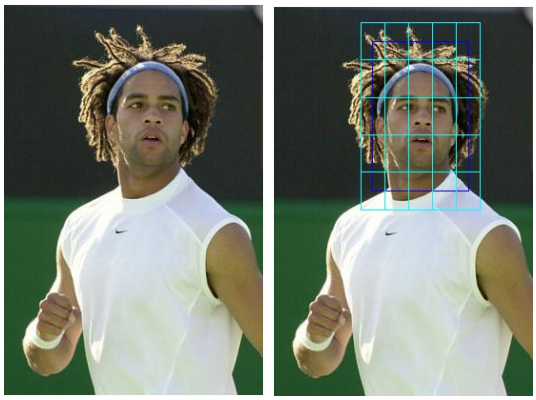
PS:If the face is on the border of the image we need fill the empty space using the nearest pixel.

1.1.3 Get the hog Feature

According to the parameters given in the pdf, I get the hog feature by using scikit-image package. For each face, we represent it by 900-d array(hog feature). We can use it as input in several models for classification.

1.1.4 Visualize bounding boxes of positive and negative samples

I paint the bounding box on the original image. Comparing them in the Fig.1.



(a) Origin (b) Bounding

Figure 1: Visualize bounding boxes of positive and negative

1.1.5 Visualize HOG features

I use **scikit-image** package to extract hog feature from the face. And I visulized the hog feature, comparing it with the original picture, and show them in Fig.2.

(a) Origin (b) HOG

Figure 2: Visualize HOG feature of label

1.2 Logistic Regression model for classification

1.2.1 Description

In logistic regression, we assume all dimensions x_{ij} in x_i are conditionally independent given y_i.

$$Pr(y_i|X_i) = Bernoulli(p_i), i.e. Pr(y_i = 1|X_i) = p_i, Pr(y_i = 0|X_i) = 1 - p_i$$

We assume that $logit(p_i) = log\frac{p_i}{1-p_i} = X_i^T\beta$, Then:

$$p_i = sigmoid\left(X_i^\top\beta\right) = \frac{e^{X_i^\top\beta}}{1+e^{X_i^\top\beta}} = \frac{1}{1+e^{-X_i^\top\beta}}$$

Then we can get the probability:

$$Pr\left(y_i = y_i^*|\mathbf{X}_i\right) = p_i^{y_i^*}\left(1-p_i\right)^{1-y_i^*} = \left(\frac{e^{X_i^\top\beta}}{1+e^{X_i^\top\beta}}\right)^{y_i^*}\left(1-\frac{e^{X_i^\top\beta}}{1+e^{X_i^\top\beta}}\right)^{1-y_i^*} = \frac{e^{y_i^*X_i^\top\beta}}{1+e^{X_i^\top\beta}}$$

At last, we can get the corresponding log-likelihood is:

$$\log Pr(\beta) = \sum_{i=1}^{n}\left[y_i^*X_i^\top\beta - \log\left(1+\exp X_i^\top\beta\right)\right]$$

1.2.2 Optimizing methods

The maximum likelihood is to find the most plausible explanation to the observed data. We can use gradient decent to find the solution.

$$\begin{aligned}\frac{\partial}{\partial\beta_j}J(\beta) &= -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\frac{1}{h_\beta\left(\mathrm{x}^{(\mathrm{i})}\right)}\frac{\partial}{\partial\beta_j}h_\beta\left(\mathrm{x}^{(\mathrm{i})}\right) - \left(1-y^{(i)}\right)\frac{1}{1-h_\beta\left(\mathrm{x}^{(\mathrm{i})}\right)}\frac{\partial}{\partial\beta_j}h_\beta\left(\mathrm{x}^{(\mathrm{i})}\right)\right)\\ &= -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\frac{1}{g\left(\beta^Tx^{(\mathrm{i})}\right)} - \left(1-y^{(i)}\right)\frac{1}{1-g\left(\beta^Tx^{(\mathrm{i})}\right)}\right)\frac{\partial}{\partial\beta_j}g\left(\beta^Tx^{(\mathrm{i})}\right)\\ &= \frac{1}{m}\sum_{i=1}^{m}\left(h_\beta\left(\mathrm{x}^{(\mathrm{i})}\right) - y^{(i)}\right)x_j^{(\mathrm{i})}\end{aligned}$$

we can update the β by

$$\beta_j = \beta_j + \alpha\sum_{i=1}^{m}\left(y^{(i)} - h_\beta\left(\mathrm{x}^{(\mathrm{i})}\right)\right)x_j^{(\mathrm{i})}$$

1.2.3 Result

Accuracy by sgd , gd, Langevin dynamics

Kernel type	SGD(lr=0.01)	GD(lr=0.01)	langevin(lr=0.01, eps = 0.01)
Accuracy on train set	0.96637	0.97505	0.96349
Accuracy on test set	0.97045	0.96771	0.96002

1.2.4 Conclusion

According to my observation:

- Gradient Descent converges faster (with fewer steps) than SGD and langevin. Because its get the global gradient. This gradient is more correct than others.
- SGD and langevin has more random than GD. For each step, they cost less time than GD, because they only get gradient of one sample not global. But sometimes, they will go to the wrong direction because one sample might be wrong.

1.3 Fisher model for classification

1.3.1 Description

First, we need to compute the β which can split the data into two parts.

$$S_W = n_{\text{pos}}\Sigma^+ + n_{\text{neg}}\Sigma^-$$

where $\Sigma^+ = E_{i\in\Omega^+}\left[\left(X_i - E_{i'\in\Omega^+}\left[X_{i'}\right]\right)\left(X_i^\top - E_{T'\in\Omega^+}\left[X_{i'}^\top\right]\right)\right]$
And $\Sigma^- = E_{i\in\Omega^-}\left[\left(X_i - E_{i'\in\Omega^-}\left[X_{i'}\right]\right)\left(X_i^\top - E_{T'\in\Omega^-}\left[X_{i'}^\top\right]\right)\right]$
Then, we can get β

$$\beta \propto S_W^{-1}\left(\mu^+ - \mu^-\right)$$

At last, we can compute the inter variance and intra variance.

$$\sigma^2_{\text{within}} = n_{\text{pos}}\sigma^2_{\text{pos}} + n_{\text{neg}}\sigma^2_{\text{neg}}$$

$$\sigma^2_{\text{between}} = \left[\left(\mu^+ - \mu^-\right)^\top \beta\right]^2$$

1.3.2 Result

Here I will show the accuracy on trainset and test set by fisher model. And I also show the intra variance and inter variance.

	Accuracy		Variance
Train set	0.97408	Intra var	8.424e-07
Test set	0.96771	Inter var	8.426e-07

1.3.3 Approach to Improve Performance

I search for the threshold by grid search between postive mean and negative mean. In this way, I found the most appropriate threshold to achieve the highest classification accuracy.

1.4 SVMs for classification

SVM(Support Vector Machine) is a supervised mathods used for classification.

1.4.1 Description

Assume that w is the support vector. Then:

$$\begin{cases} w^T x_i + b \geq +1, y_i = +1 \\ w^T x_i + b \leq -1, y_i = -1 \end{cases}$$

Then, we want to find a support vector which is maximize the distance between support vector and points. Then :

$$\begin{aligned} &\min_{w,b} \tfrac{1}{2}\|w\|^2 \\ &\text{s.t.} y_i \left(w^T x_i + b\right) \geq 1, \quad i = 1, 2, \ldots, m \end{aligned}$$

At last, we use the KKT conditions and Lagrange multiplier to solve this problem.

1.4.2 Result

According to the result in the table, we can see that kernel of linear, rfb, poly can all reach to a high accuracy while the sigmoid doesn't have a satisfied performance.

Kernel type	Linear	RBF	Poly	Sigmoid
Accuracy on train set	0.97171	0.98960	0.99782	0.89400
Accuracy on test set	0.96848	0.97636	0.98001	0.88449

1.4.3 Support Vectors

The vector is so long to show in the report. I print them in the program and store in a txt file. You can look it by running the svm code or in the support_vector.txt.

1.4.4 Approach to Improve Performance

Tune the parameters. I use grid search of sk-learn to combine different combinations of parameters such as gamma, coef0 and so on.

And I also check for how to choose the different kernels.

- Linear. When linearly separable, when the number of features is large, the number of samples is large.
- RBF, When the feature dimension is small and the number of samples is normal, it is used when there is no prior knowledge.
- Poly. For image processing, it has more parameters than RBF.
- Sigmoid. Generating neural networks, which are similar to RBF in some parameters, may be invalid in some parameters.

1.5 Convolutional Neural Networks for classification

1.5.1 Description

The code is based on tensorflow. The network I build has 6 layers. And the structure is as following : Conv1((3,3,32)) $->$ Conv2((3,3,32)) $->$ maxpool1(2,2) $->$ Drop(0.25) $->$ conv3((3,3,64)) $->$ conv4((3,3,64)) $->$ maxpool2(2,2) $->$ Drop(0.25) $->$ Flatten() $->$ Fc1(256,relu) $->$ Fc2(2,sigmoid).

The optimizer is SGD(lr=0.01, decay = 1e-6, momentum = 0.9)

1.5.2 Data Preparation

In the data preprocess step, I cut and save the negative and postive image into local disk. When training in the CNN, I read it and do normalization to them. What's more, transform the label to one hot vector.

1.5.3 Result

Here is the result of CNN on FDDB dataset. We can see that it has better performance than other classical classify methods such as SVM, Logistic Regression, and others.

	Accuracy
Train set	0.9955
Test set	0.9873

1.5.4 Approach to Perform Better

As we all know that some hyper-parameters have important impact on the performance of CNN model. So I tuned the learning rate, learning rate scheduler, and structures of model. At last, I find that the model is enough to imply on this task. Change other hyper-parameters has no influence on the result. The highest is 0.9955(train) and 0.9873(test). Otherwise, the model will overfitting.

1.6 Feature Visualize

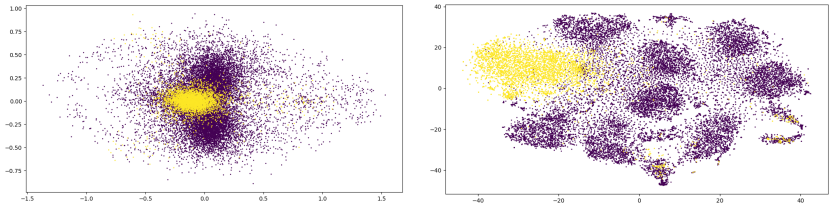
I visualize hog feature and CNN's feature by pca and tsne. Here is the result.

1.6.1 Hog feature

Here is the result of hog feature visualization in Fig.3.

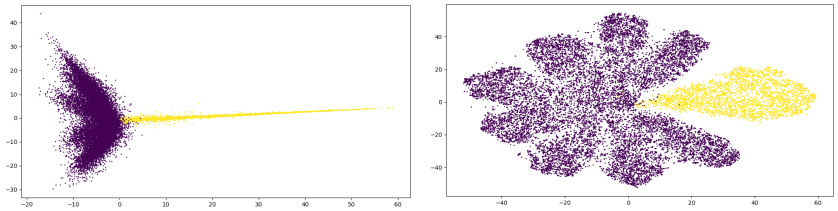
1.6.2 CNN feature

I extract the output of CNN's first fc layer. dimension is 512. And here is the result in Fig.4.



(a) PCA (b) TSNE

Figure 3: Visualize hog feature by TSNE and PCA



(a) PCA (b) TSNE

Figure 4: Visualize CNN feature by TSNE and PCA

1.6.3 Conclusion

By comparing with these picture. We can see these conclusion:

- TSNE perform better than PCA. But PCA is faster than TSNE, can save much time.
- CNN feature is better than hog feature by observing the result.

2 Face Detection

I will realize the face detection by using the model trained before.

2.1 Sliding Windows

By observing face annotation of the dataset FDDB, I found that the smallest face's scale is about 40. So I set 40 as the smallest window to slide. The smallest window's is (30,40).

At first, I set the slide stride is 1. When I cut the windows from the image, I find there exist more than 100,000 images need to classify. It's too slow to detect. Finally, I set the stride as 10 for slide windows.

For all the scales. I found that the for all types scale. It's also too many windows need to detect. And I set only 10 window's scale for slide windows. From smallest scale to the largest scale, I gave 10 degrees for it. Then, the result need to detect is less.

According to the count, there are exists **5138** windows in this picture. The next step we will classify which windows are face.

2.2 Hog Feature

For all the windows cut from the original images. I resize them into 96*96, and extract the hog features for classifing.

2.3 Classifying

The models I choose are logistic model, SVM kernel with linear, SVM kernel with RBF and CNN which trained before. These model all has very high accuracy for classifying.

2.3.1 Only one Model

If I only use one model, to predict all the windows. The windows number is too large. The performance is just as following in the Fig.5. We can see that the performance is terrible. So it's not a good solution to detect. According to the count, there are 206 windows in this picture.

2.3.2 All Models

One models maybe make some windows wrong. So I use filters to classify whether the windows is the face of man. The idea is that , First, use logistic to predict, if no, skip this window, otherwise, use svm to predict, if yes, continue to use CNN to predict. Only all the model predict this window is face. I will set this windows as face. I want to decrease the windows number in this way. However, according to the count, there are 206 windows in this picture which is only 2 less than one model detection. This methods is also terrible. 5

(a) one (b) many

Figure 5: use different model number

2.3.3 Increase the threshold

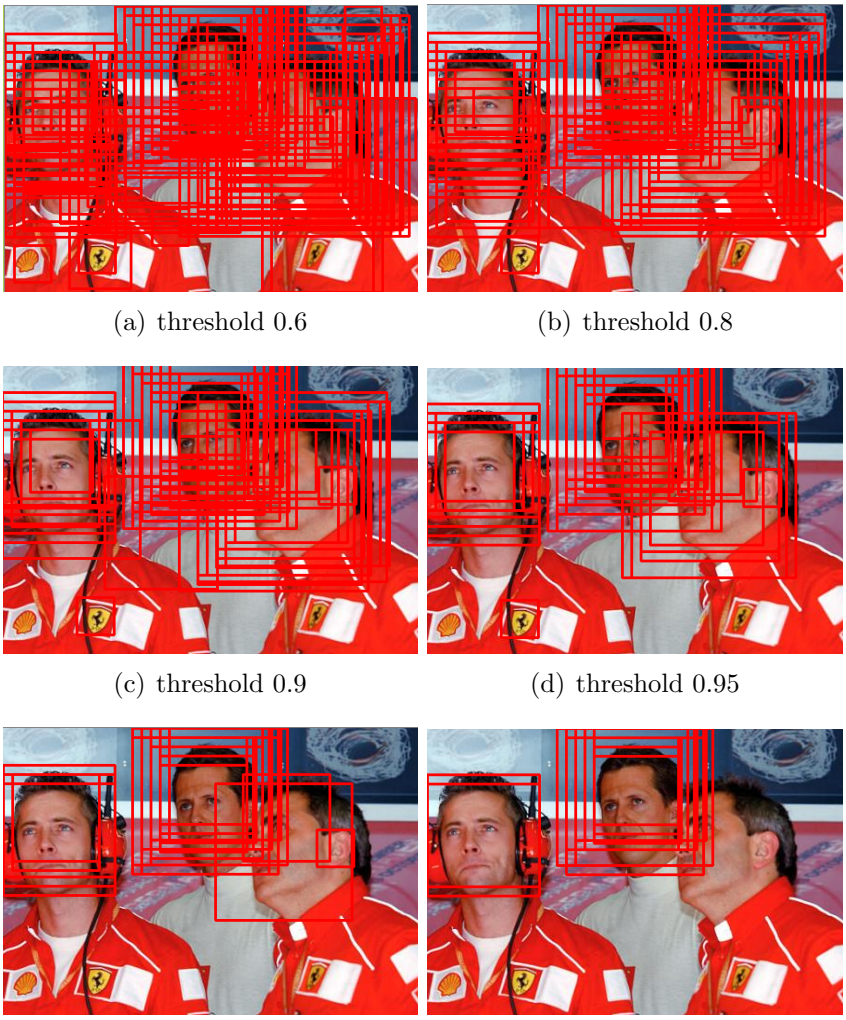
For each model, we can increase the threshold to get the most probobality windows which are faces.

- For logistic, the predict result can be regard as the probability of the face. If more than 0.5, we regard it as the face. So we can improve the threshold to number which is more than 0.5.
- For SVM, by set the parameter(probability = True) when training the SVM by sklearn. We can output the probability when predict. When improve the threshold, to find the most likely face.
- For CNN, I extract the last one layer output. by Sigmoid, to find the most likely face windows.

By doing this method, we can decrease the windows number by this way. And here is the result in Fig.6.

According to the result shown in Fig, we can see that as the threshold increasing, the windows number is decreasing, and the performance is better than before. And when threshold is more than 0.99, some face will be drop. So I choose 0.98 as the threshold to classify the face.

threshold	0.5	0.6	0.7	0.8	0.9	0.95	0.97	0.98	0.99	0.995
windows number	206	175	146	102	68	42	32	27	17	15



(a) threshold 0.6 (b) threshold 0.8

(c) threshold 0.9 (d) threshold 0.95

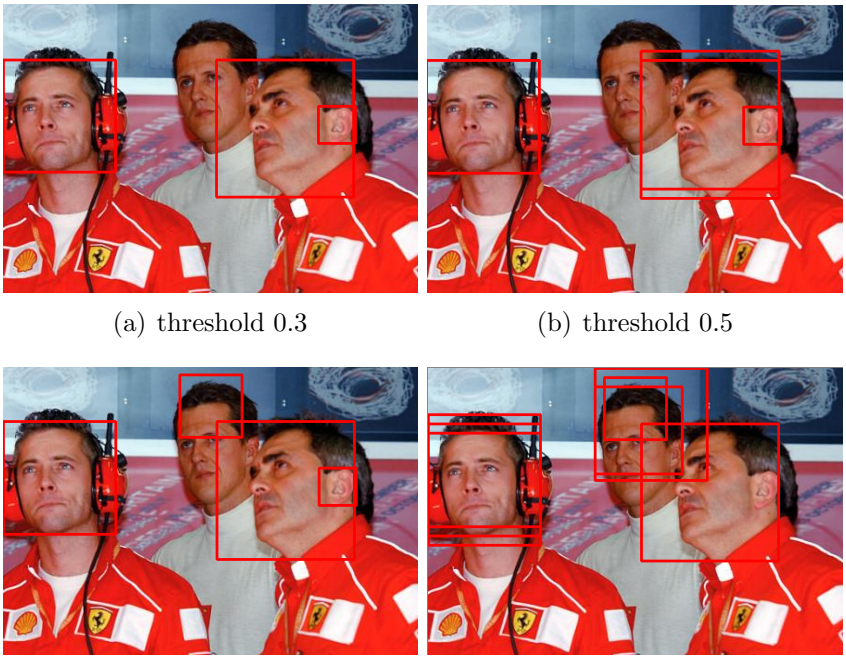
(e) threshold 0.98 (f) threshold 0.995

Figure 6: use different threshold number

2.4 Combine the Windows

According to the result given before, we can see many windows are overlapping. One face was cropped by several windows. So we must choose some algorithms to combine the windows to one if they point to the same face.

The algorithm I choose is NMS(non-maximum-suppression), which can delete some unimportant windows.



(a) threshold 0.3 (b) threshold 0.5

(c) threshold 0.6 (d) threshold 0.65

Figure 7: use different nms threshold

The algorithm is as following:

- Sort the windows by their probability.
- each time choose one windows with largest probability. Then compute the IOU between it and others.
- if IOU is more than threshold, the window will be delete
- then choose the next largest probability windows do the same thing.

By different nms threshold, we can see different performance in Fig.7, according to the result shown following, we can see that the nms threshold equal to 0.6 is the best threshold.

2.5 Conclusion

We can see that, the face detection isn't performance well in this way. I survey the face detection algorithm and I found that the algorithm now used widely is not just sliding windows to classify. There are some algorithm perform better and save time.

And, in this task, we only regard the picture with only half face as the negative sample. In my opinion, maybe we select some picture from background as the negative samples and performance will be better because the classifier will knew which is the background and make less mistakes when detecting.

3 Appendix

Thanks for prof.Zhang and T.A.'s helping to learn in this term. Before, I can only call the machine learning package to solve the problem and seldom read the papers about the newest machine learning. Now, I try to understand the mathmatics and essence of the algorithms, but just know how to use them. What's more, in this lecture, I knew more about how to realize a model from the papers which shows my coding ability is better.

I believe this has laid a solid foundation for my future scientific research. Thanks!