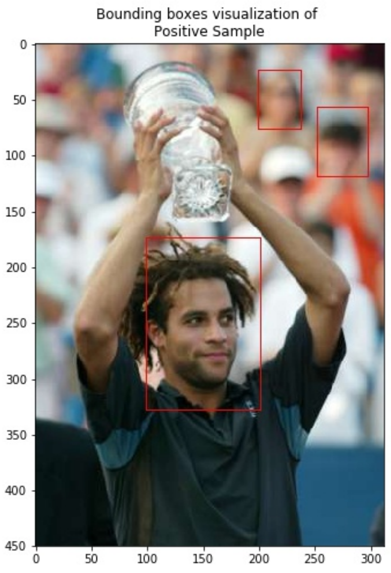
Machine Learning Course Project 1 Report

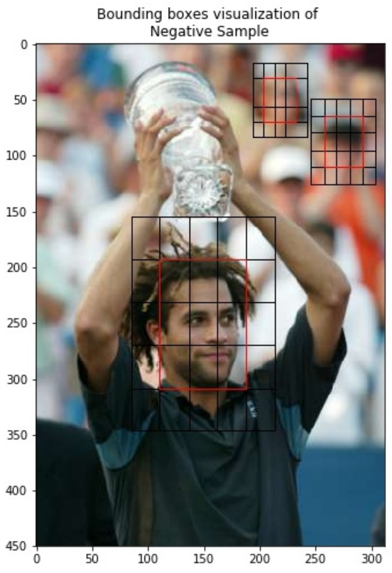
Student ID: 516030910544 Name: Hongbin Chen

Part I: Data Preprocessing

1. About Dataset

I do the project on dataset FDDB(Face Detection Data Set and Benchmark), which is a dataset for face detection task. I use this dataset to do classification task, so I crop the face within the bounding box as a positive sample according to ground truth annotation. And for each face, I generate eight negative images based on each face by sliding the bounding box by 1/3. Here shows an example, the red bounding boxes in figure are for positive samples while black bounding boxes are for negative samples.





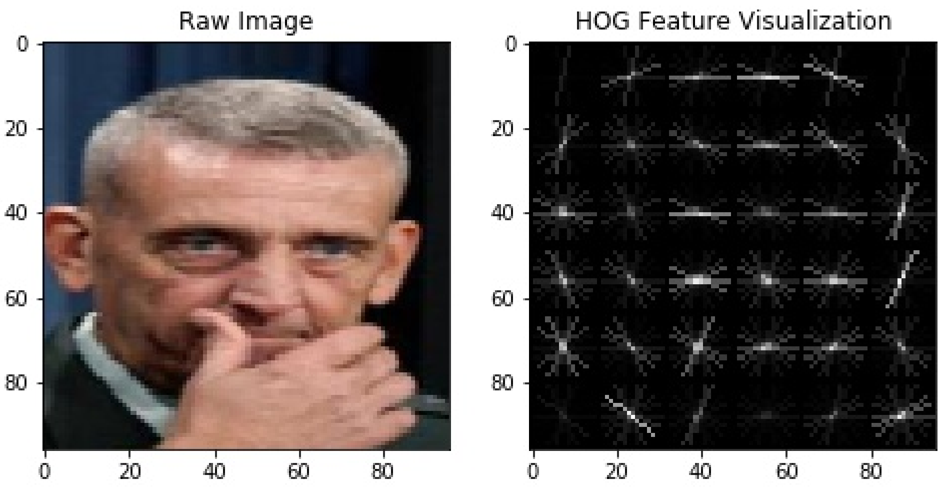
Finally, I obtain 4136 positive samples and 16544 negative samples for training set, 1035 positive samples and 4168 negative samples for test set.

2. HOG Features Extraction

For traditional machine learning models, feature extraction is usually necessary. In this project, I extract the HOG features for each sample. For convenience, I use the *hog* API of skimage library. Some detail settings are as follow:

```
Block size = 2 x 2
Cell size = 16 x 16
Number of bins = 9
Block overlap =1 x 1
```

In this way, I extract a 900-d feature vector for each sample, and I visualize some HOG features.



Part II: Face Classification

For face classification task, I have tried logistic model, fisher model, SVM and CNN. The best performances of each model are shown on following table.

Model	**Best Performance**
Logistic Model	96.81%
Fisher Model (LDA)	97.02%
SVM with Linear Kernel	96.91%
SVM with RBF Kernel	97.23%
SVM with Sigmoid kernel	96.94%
CNN Model	98.21%

Generally, CNN model has the best performance while logistic model has relatively poor performance.

1. Logistic Model

The loss function can be represented as: $L(\beta) = \sum_{i=1}^{n} log(1 + exp(-y_i x_i^T \beta))$

The gradient of loss function is: $L^{'}(\beta) = \sum_{i=1}^{n} -\frac{y_i x_i}{1+exp(y_i x_i^T \beta)}$

In this project, I use both SGD and Langevin dynamics to optimize the model. Their performances are shown below:

	SGD Optimizer	**Langevin Dynamics Optimizer**
Best Performance	96.81%	93.14%

a. SGD Optimizer

SGD optimizer updates the parameters for each sample, so it can do almost 20,000 iterations for each epoch. I run 1000 epoches with *learning rate* 0.001 and obtain accuracy 96.81% on test set. This is a pretty good result.

b. Langevin Dynamics Optimizer

The Langevin dynamics is:

$X_{t+\Delta t} = X_t - \frac{1}{2}U^{'}(X_t)\Delta t + \sqrt{\Delta t}\epsilon_t$

In above formula, U can be considered as the loss function, and when I set Δt as 1 for each iteration simply. Thus, the parameters updating formula can be rewritten as:

$\beta_{t+1} = \beta_t - lr * L^{'}(\beta_t) + \epsilon_t$

The ϵ_t can be seen as a Gaussian noise for each iteration.

However, Langevin dynamics optimizer only update parameters once for each epoch, so it is very time consuming. Finally, I only run 1,1000 epoches with learning rate* 0.06 and obtain accuracy 93.14%, which is worse than optimizing with SGD.

2. Fisher Model

Fisher model project the data to a 1-dimension space, I can simply calculate the projection vector using formula:

$\beta = S_W^{-1}(\mu^+ - \mu^-)$ where $S_W = n_{pos}\Sigma^+ + n_{neg}\Sigma^-$

The results are shown as follows:

	Accuracy	Intra-class Variance	Inter-class Variance
Fisher Model	97.02%	9.978779e-4	9.944171e-7

Further Discussion

Fisher model learn a projection vector to project each sample into 1-dimension space, but how I should use the projection to do classification.

Usually, fisher model suppose the sample can be fitted by Gaussian distribution, so I can consider the samples after projection as two Gaussian distribution (Positive and Negative), and predict a sample by calculating the probability it belongs to the Gaussian distribution. In this way, I get an accuracy 96.43%.

However, I choose a classification threshold between two class center which leads to best performance on training set to do predict, and get an accuracy 97.02%.

From above experiments, I find that the samples don't fit Gaussian distribution very well.

3. SVM

a. Results of Different Kernels

SVM is a kind of powerful traditional machine learning model, it has many modified versions, such as with different kernels. I try three different kernels, which is linear kernel, RBF kernel and sigmoid kernel. The results are as follows:

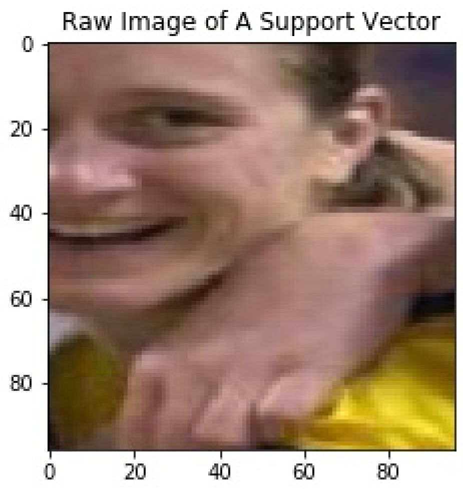
Kernel	Linear Kernel	RBF Kernel	Sigmoid Kernel
Best Performance	96.91%	97.23%	96.94%

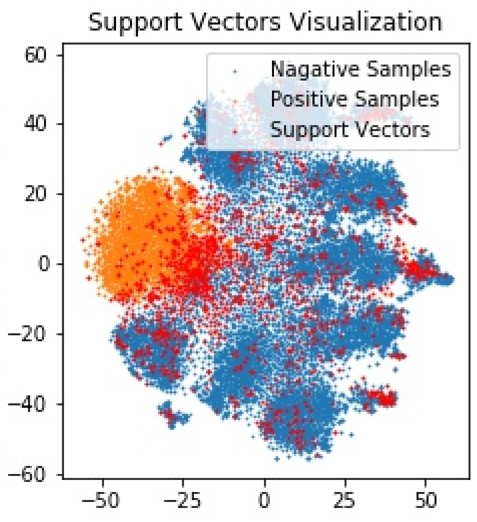
I find that SVM with RBF kernel has the best performance, maybe because it projects samples to high dimension space and has stronger capacity.

b. About Support Vectors

I research on the support vectors of SVM with RBF kernel.

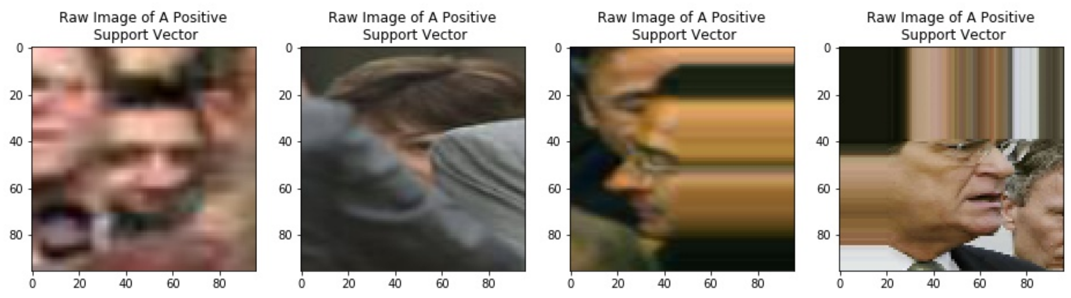
The model has 1643 support vectors. I visualize the support vectors using t-sne trying to find their distribution compared to other samples.





As shown in above right figure, the red dots represent support vectors, the data of visualization are from HOG features of each sample. I find the support vectors don't separate positive and negative samples very well, they may scatter everywhere.

Further, I see the ground truth label of each support vector, I am surprised about positive labels accounting for 45.77%, almost half. We show some support vectors with positive label here:


Raw Image of A Positive Support Vector
Raw Image of A Positive Support Vector
Raw Image of A Positive Support Vector
Raw Image of A Positive Support Vector

We can see that these samples are dim, partial or with padding. They are not all a normal and standard positive samples. And for some negative samples, because they are generated by sliding the bounding boxes of positive samples, they are naturally to be support vectors.

4. CNN

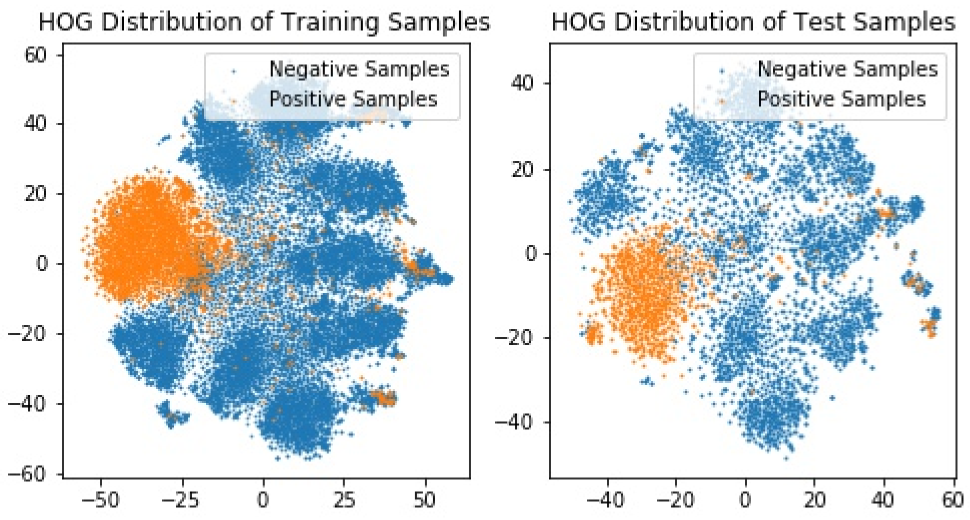
Convolution neural network is a very powerful implement. I define a CNN with three conv-layers to classify faces. The structure is shown as follows:

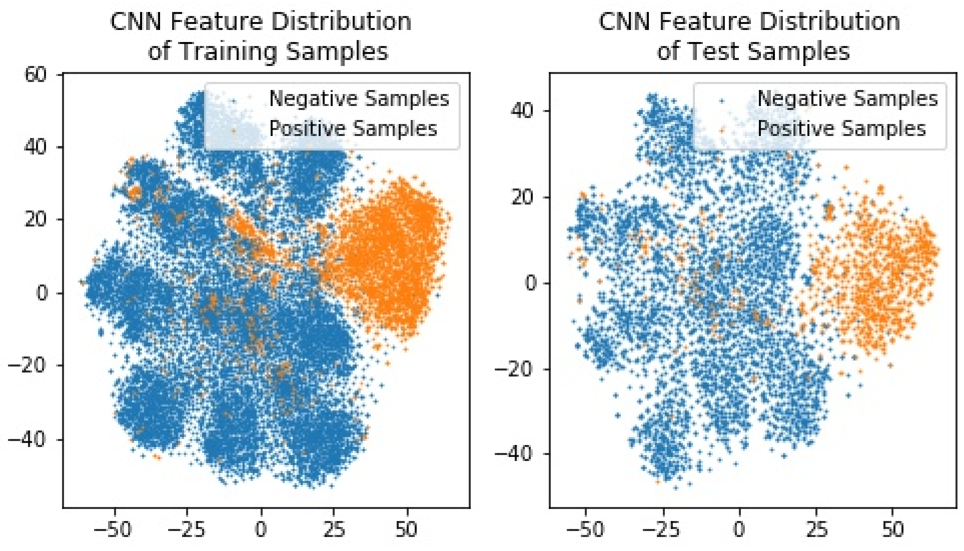
```
Net(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(7, 7), stride=(1, 1))
  (conv3): Conv2d(16, 32, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=2048, out_features=600, bias=True)
  (fc2): Linear(in_features=600, out_features=150, bias=True)
  (fc3): Linear(in_features=150, out_features=18, bias=True)
  (fc4): Linear(in_features=18, out_features=2, bias=True)
)
```

I use the optimizer SGD with momentum = 0.9 and *learning rate* 0.001, and obtain an amazing performance 98.21%.

Part III: Feature Distribution Visualization

I visualize two kinds of feature distribution. One is HOG features of each samples and the other is the last conv-layer's output of CNN, they both are reduced to 2-d using t-sne dimension reduction. Results are shown in following figures.





The left sub figures show the feature distribution of training samples while the right figures show the feature distribution of test samples. As you can see, CNN feature distribution seems to be more similar between training samples and test samples. So maybe CNN features have stronger generalization.

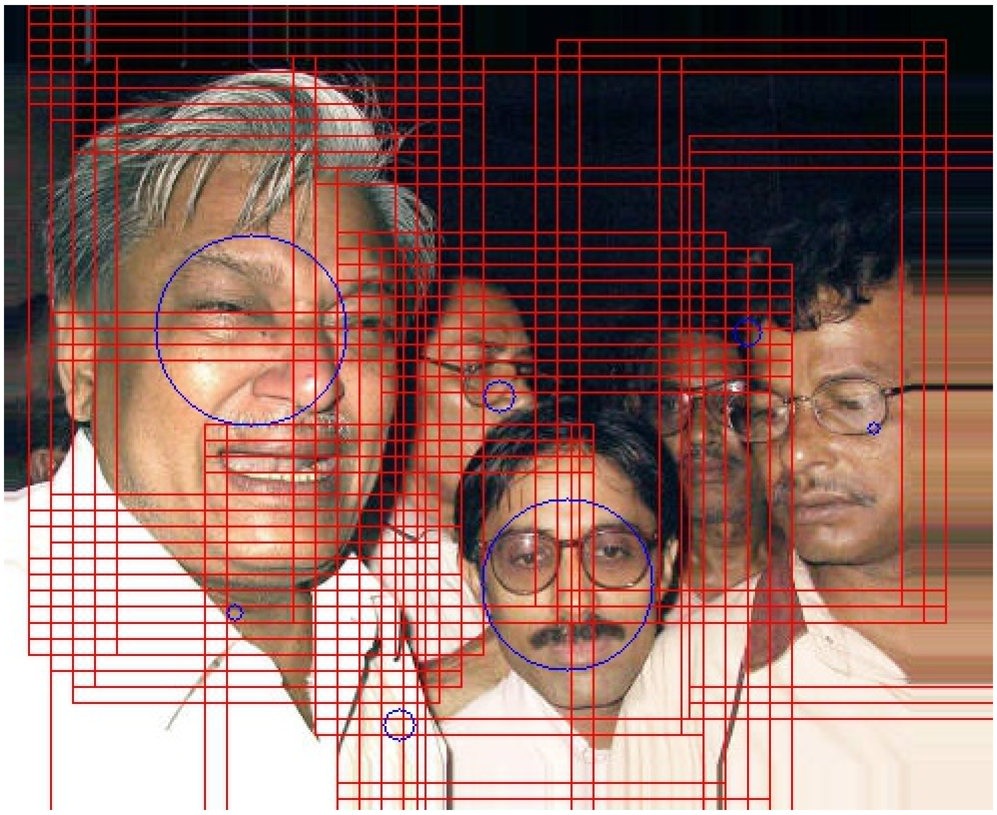
Part IV: Face Detection

1. Combining Face Classification and Sliding Window

Sliding window and checking whether the window bounding box is a face is a simple and natural idea for face detection. However, the potential bounding boxes are too many and it could be time-consuming. So I use some tips to reduce the number of potential bounding boxes sharply. I bound the search space as follows:

```
window_height = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8]*img_height # bound of scale
window_height : window_width = 3 : 2                              # bound of window shape
height_stride = int(img_height / 40)                              # bound of window number
width_stride = int(img_width / 40)                                # bound of window number
```

But the face detection results are still not satisfactory. As following figure shows, the potential boxes of a face are too many.



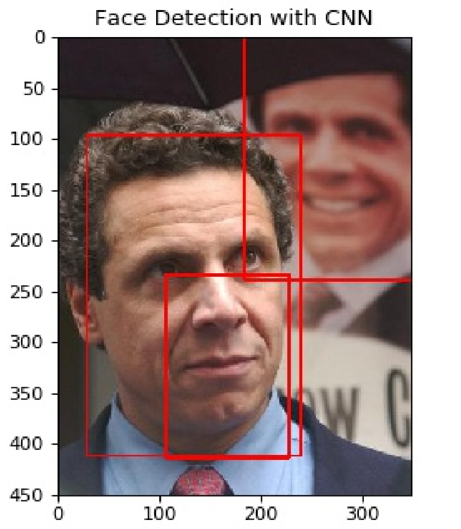
2. Improvement

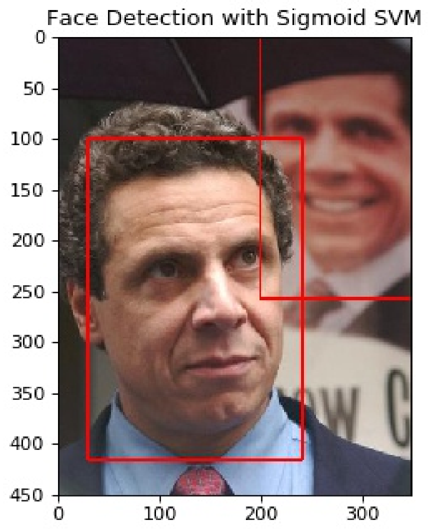
The problem is that a face may have many potential bounding boxes around it, I need to reduce these bounding boxes to a proper one.

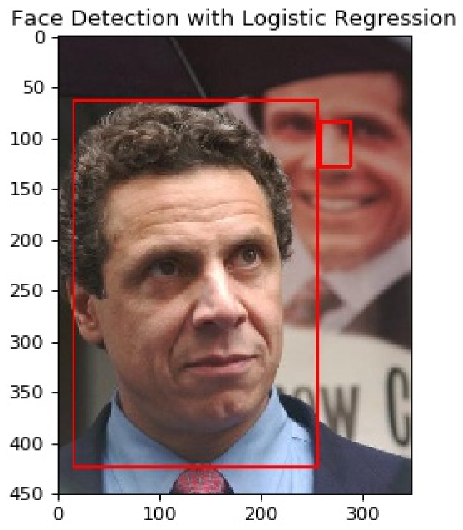
Notice that around a higher confidence face, there are more potential bounding boxes around it, and almost each normal and standard face will bring to a group of potential bounding boxes. Based on this fact, I use the mean shift clustering algorithm to find groups and locate the face. The description of algorithm is as follows:

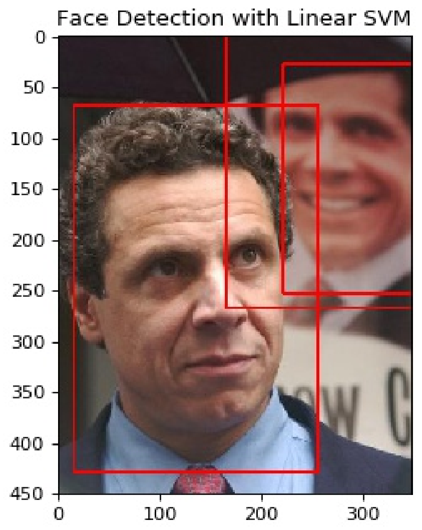
```
1. At each window scale, find all potential bounding boxes.
2. Cluster all potential bounding boxes using mean shift algorithm.
3. Find the groups that are big enough to account for 30% samples, take the groups' center
as bounding boxes' center to generate potential results.
4. Travel throught all potential window scales.
5. Finally, combine some potential results, that is if two potential results' center is
close and the larger one covers the smaller one, then remove the larger one.
```

The visualization results of face detection are as follows:


Face Detection with CNN
0
50
100
150
200
250
300
350
400
450
0
100
200
300


Face Detection with Sigmoid SVM
0
50
100
150
200
250
300
350
400
450
0
100
200
300


Face Detection with Logistic Regression
0
50
100
150
200
250
300
350
400
450
0
100
200
300


Face Detection with Linear SVM
0
50
100
150
200
250
300
350
400
450
0
100
200
300