Project Report

(12/29/17)

517030910318	Li Jingyu
517030910315	He Wendi
517030910316	Jiang Huangfei

CONTENT

1. INTRODUCTION 3
2. OUR TASKS 3
    - 2.1 Problem Description 3
    - 2.2 Basic Ideas 3
    - 2.3 Theories and Methods 4
        - 2.3.1 LBP 4
        - 2.3.2 SVM 5
3. SOLUTION DESIGN 7
    - 3.1 Collecting training data 7
        - 3.1.1 Detect Faces 7
        - 3.1.2 Get LBP Features 7
    - 3.2. Creating our model 8
        - 3.2.1 Partition dataset 8
        - 3.2.2 Model Training 8
    - 3.3 Application for Real-time Detect 8
4. ANALYSIS AND OUR IMPROVEMENTS 8
    - 4.1 Performance Metrics 8
    - 4.2 Parameter Adjusting 9
    - 4.3 LBP Operators 9
    - 4.4 Data Normalization 10
    - 4.5 Other Improvements 11
        - 4.5.1 In Feature Extraction 11
        - 4.5.2 In Real-time Detect 12
5. OTHER POSSIBLE IMPROVEMENTS 12
    - 5.1 Head pose 12
    - 5.2 other facial features 12

1. INTRODUCTION

Nowadays, machine learning is being widely used in various areas, such as object detect, auto-driving, medical diagnosis and even astronomy. The latest news is that Google's AI help NASA successfully identify two exoplanets. At the same time, connecting modern technology with people's life become more and more popular. We are familiar with the practice that let computers detect the face of a human from a picture and this tech has already been integrated into many smart phone cameras. Moreover, we are quite interested in whether we can tell something more from the facial expression. In particular, the detection of smile is a very useful tool in social life.

2. OUR TASKS

2.1 Problem Description

We are to write a program that is able to tell if the person on the picture is smiling or not, when a picture of a person is given.

As given, we use GENKI-4K dataset to train and test our algorithms. GENKI-4K contains 4,000 face images spanning a wide range of subjects, facial appearance, illumination, geographical locations, imaging conditions, and camera models. All images are labeled for both Smile content (1=smile, 0=non-smile) and Head Pose (yaw, pitch, and roll parameters, in radians).

2.2 Basic Ideas

We use the traditional machine learning techniques.

(1) Face detection.

Detect faces in those images. We directly use face detector in opencv/dlib.

(2) Feature extraction

Use faces as input, and collect their features. There are two main types of features HOG and LBP. We first consider the LBP.

(3) Smile classification.

Train a model for classification. Then cross-train the models, and choose the best one. We use SVM classifier.

(4) Real-time Smile Detection application using camera

Using the camera on computers to judge whether the person in front of the camera is smiling in the real-time.

2.3 Theories and Methods

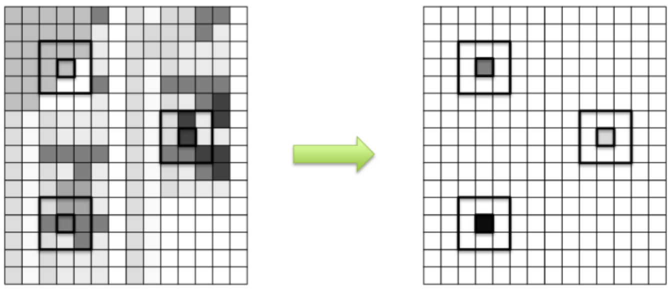
2.3.1 LBP

2.3.1.1 Introduction

LBP, Local binary patterns. LBP has been shown powerful and is often used in problems about human face. The simple LBP records the contrast information between the pixel and the surrounding pixels. LBP is used to describe local texture features.

2.3.1.2 Principle

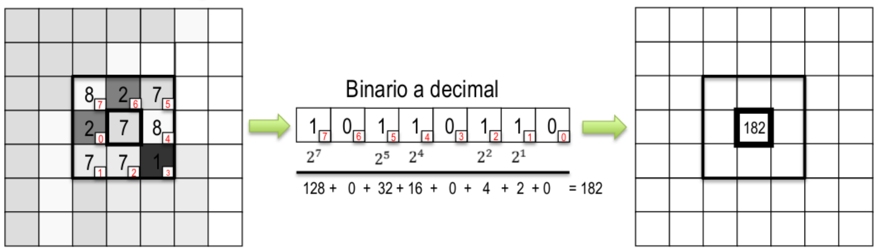
Let's first discuss how to calculate the LBP Descriptor. Firstly, we convert the input color image to grayscale, since LBP works on grayscale images. For each pixel in the grayscale image, a neighborhood is selected around the current pixel and then we calculate the LBP value for the pixel using the neighborhood. After calculating the LBP value of the current pixel, we update the corresponding pixel location in the LBP mask which is of same height and width as the input image with the LBP value calculated as shown below. In the image, we have 8 neighboring pixels.



To calculate the LBP value for a pixel in the grayscale image, we compare the central pixel value with the neighboring pixel values. We can start from any neighboring pixel and then we can transverse either in clockwise or anti-clockwise direction but we must use the same order for all the pixels. Since there are 8 neighboring pixels – for each pixel, we will perform 8 comparisons. The results of the comparisons are stored in a 8-bit binary array.

If the current pixel value is greater or equal to the neighboring pixel value, the corresponding bit in the binary array is set to 1 else if the current pixel value is less than the neighboring pixel value, the corresponding bit in the binary array is set to 0.

The whole process is shown in the image below (Figure 2). The current (central) pixel has value 7. We start comparing from the neighboring pixel where the label 0. The value of the neighboring pixel with label 0 is 2. Since it is less than the current pixel value which is 7, we reset the 0th bit location in the 8 bit binary array to 0. We then iterate in the counter-clockwise direction. The next label location 1 have value 7 which is equal to the current pixel value, so we set the 1st bit location in the 8 bit binary to 1. We then continue to move to the next neighboring pixel until we reach the 8th neighboring pixel. Then the 8-bit binary pattern is converted to a decimal number and the decimal number is then stored in the corresponding pixel location in the LBP mask.

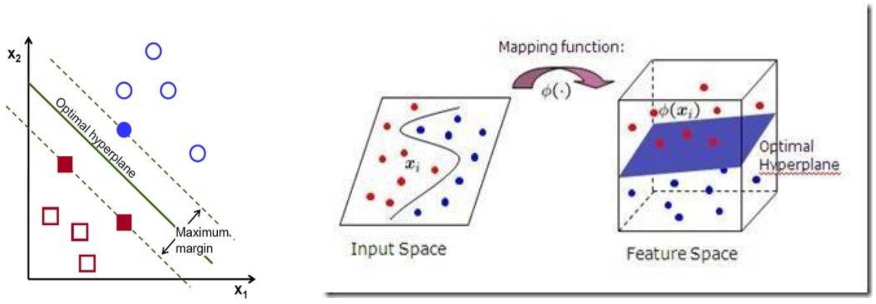


Once we have calculated the LBP Mask, we calculate the LBP histogram. The LBP mask values range from 0 to 255, so our LBP Descriptor will be of size 1x256. Lastly, we should connect the histograms of each cell into one feature vector.

2.3.2 SVM

2.3.2.1 Introduction

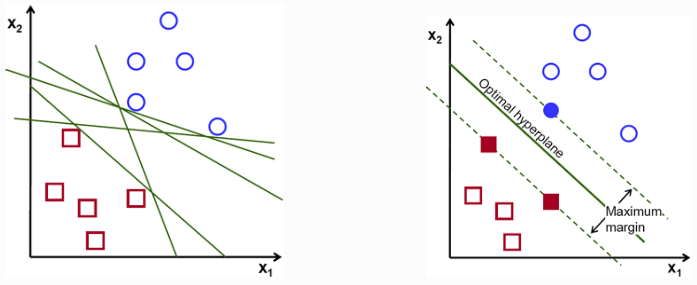
SVM, Support vector machine. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. An SVM model is representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible, as the following figure shows.



New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces, as the figure shows above.

2.3.2.2 Linearly Separable Data

Consider the image below which has two types of data, red and blue. Our goal is to find a line, $f(x) = ax_1 + bx_2 + c$ which divides both the data to two regions. When we get a new test data X, just substitute it in $f(x)$. If $f(X) > 0$, it belongs to blue group, else it belongs to red group. We can call this line as Decision Boundary.



Then, we need to find a line that should be passing as far as possible from all the points. What SVM does is to find a straight line or hyperplane with largest minimum distance to the training samples. See the bold line in the second image passing through the center. To find this Decision Boundary, we need training data. We don't need all but just the ones which are close to the opposite group to help with the data reduction. In the image, they are the one blue filled circle and two red filled squares. We can call them Support Vectors and the lines passing through them are called Support Planes.

What happened is, first two hyperplanes are found which best represents the data. For example, blue data is represented by $w^T x + b_0 > 1$ while red data is represented by $w^T x + b_0 < -1$ where w is weight vector ($w = [w_1, w_2, ..., w_n]$) and x is the feature vector ($x = [x_1, x_2, ..., x_n]$). b_0 is the bias. Weight vector decides the orientation of decision boundary while bias point decides its location. Now decision boundary is defined to be midway between these hyperplanes, so expressed as $w^T x + b_0 = 0$. The minimum distance from support vector to the decision boundary is given by $distance_{support\ vectors} = \frac{1}{||w||}$. Margin is twice this distance, and we need to maximize this margin. i.e. we need to minimize a new function $L(w, b_0)$with some constraints which can expressed below:

$$\min_{w,b_0} L(w, b_0) = \frac{1}{2}||w||^2 \text{ subject to } t_i(w^T x + b_0) \geq 1 \ \forall i$$

where t_i is the label of each class, $t_i \in [-1, 1]$.

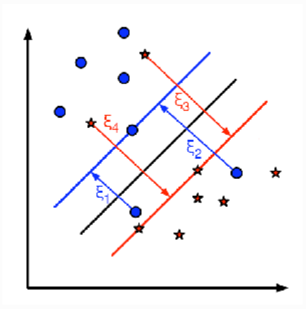
2.3.2.3 Non-Linearly Separable Data

There are some data which can't be divided into two with a straight line. So, we need to map this data set with a function. But obviously, there are more choices for lines in this case. In general, it is possible to map points in a d-dimensional space to some D-dimensional space $(D > d)$ to check the possibility of linear separability. There is an idea which helps to compute the dot product in the high-dimensional (kernel) space by performing computations in the low-dimensional input (feature) space. We can calculate higher dimensional features from lower dimensions itself. Once we map them, we get a higher dimensional space.

In addition to all these concepts, there comes the problem of misclassification. We need to modify our model such that it should find decision boundary with maximum margin, but with less misclassification. The minimization criteria is modified as:

$$min\ ||w||^2 + C(distance\ of\ misclassified\ samples\ to\ their\ correct\ regions)$$

Below image shows this concept. For each sample of the training data a new parameter ξ_i is defined. It is the distance from its corresponding training sample to their correct decision region. For those who are not misclassified, they fall on their corresponding support planes, so their distance is zero.



So the new optimization problem is :

$$\min_{w,b_0} L(w,b_0) = ||w||^2 + C\sum_i \xi_i \text{ subject to } y_i(w^T x_i + b_0) \geq 1 - \xi_i \text{ and } \xi_i \geq 0 \ \forall i$$

Then we need to decide the parameter C. It is obvious that the answer to this question depends on how the training data is distributed. Although there is no general answer, it is useful to take into account these rules:

Large values of C give solutions with less misclassification errors but a smaller margin. Consider that in this case it is expensive to make misclassification errors. Since the aim of the optimization is to minimize the argument, few misclassifications errors are allowed. Small values of C give solutions with bigger margin and more classification errors. In this case the minimization does not consider that much the term of the sum so it focuses more on finding a hyperplane with big margin.

3. SOLUTION DESIGN

3.1 Collecting training data

3.1.1 Detect Faces

We use the Haarcascade classifier already trained and embedded in the *opencv* module to detect faces in a given image. To get the detection work done more effectively, we first convert the image into a grey scale image. It's also helpful in reducing the storage size of the picture. Then, we use Gaussian Blur to filter out some disturbance. To get a higher accuracy, we use the classifier's function *detectMultiScale* and take the largest output frame as our target face.

3.1.2 Get LBP Features

For this part, we write our OWN function.

(1) With a grey scale image of face which is already obtained by our last step, we change it into a particular size, say, 60*60.

(2) Pick up a particular pixel and have a further discussion over it. Compare the eight neighboring pixels around it in grey level and denote it as '1' if the grey level is larger than the one in center, denote it as '0' otherwise. In this way we can derive a binary number in length of 8 to represent the LBP value of the center pixel. We

create a new image to represent the values.

(3) We then divide the image into several cells (such as 10*10), and determine the step (such as the exact size of the cell). And we calculate a histogram that displays the frequency of each LBP number occurring within the cell. It's a 256-dimentional feature vector.

(4) Then we concatenate all the feature vectors in a fixed order, getting the feature vector of the whole image.

Clearly, there are two parameters to resolve.

We also made an improvement of the LBP feature extraction, that is, just take the LBP values of the lower half part of the face. We will explain this in the analysis part.

3.2. Creating our model

3.2.1 Partition dataset

To partition the original dataset into dataset for training and for testing, we divide all the samples into ten folders, with the positive and negative ones distributed evenly. We write a program to do the job, and use the remainder of the number of the image over 10 as the number of its new group (n mod 10). This method also randomly distributes the data.

3.2.2 Model Training

We use the SVM class embedded in the *opencv.ml*. We chose the RBF kernel function with default Gamma, that is, one over the number of samples. We then feed the normalized LBP feature dataset and labels to the model to get it trained. As to the choice of the two parameters of the LBP feature and the different operators of LBP, we made some different combinations to train the model, then adjust them according to the model performance.

3.3 Application for Real-time Detect

We integrate our previous work to implement this real-time detect through the computer's camera. We capture the frame of the computer's camera using opencv. Face detection is the next step. Having captured the face, we put it to the model for classification and show the result on the screen. The new feature we add is that it can classify multiple faces between smiling and not smiling at the same time.

4. ANALYSIS AND OUR IMPROVEMENTS

In this part, we will discuss what factors affect the performance of the model and how to improve its performance through parameters, operators and other methods we come up with.

4.1 Performance Metrics

We use the following metrics to assess our model's performance:

- **F1 Score**: For classification tasks, the terms *true positives, true negatives, false positives, and false negatives* compare the results of the classifier under test with trusted external judgements. The terms positive and negative refer to the classifier's prediction (expectation), and the terms true and false refer to whether that prediction corresponds to the external judgement (observation)

$$F_1 = \frac{2TP}{2TP + FP + FN}$$

We also take two other metrics for reference:

- **Precision**: Also known as reliability. The fraction of samples classified as smiling that are truly smiling
- **Accuracy**: the fraction of correct classification

4.2 Parameter Adjusting

In LBP feature extraction, there are two parameters: image size, cell size, while in the classifier, there are two kernel functions to select from: linear and rbf. The following table shows the performance of models with some combination of the above parameters:

Table 1

Model Performance based on different parameters						
F1 Score		LBP feature				SVM
best	average	Image size	Cell size	Operator	dimension	Kernel
0.932	**0.913**	36	6	None	9216	rbf
0.936	0.907	60	10	None	9216	rbf
0.918	0.902	60	10	None	9216	linear
0.922	0.905	30	5	None	9216	rbf
0.927	0.902	40	5	None	16384	rbf
0.919	0.892	40	10	None	4096	rbf

As shown is the table, the model with image size 36 and cell size 6 performs most stable and comparatively well. Although its best performance falls behind that with 60 and 10, its average performance is much higher than the latter, suggesting its stability. Besides, the former one has feature vectors of far less dimension, so this makes the computation more quickly and consumes less computing power.

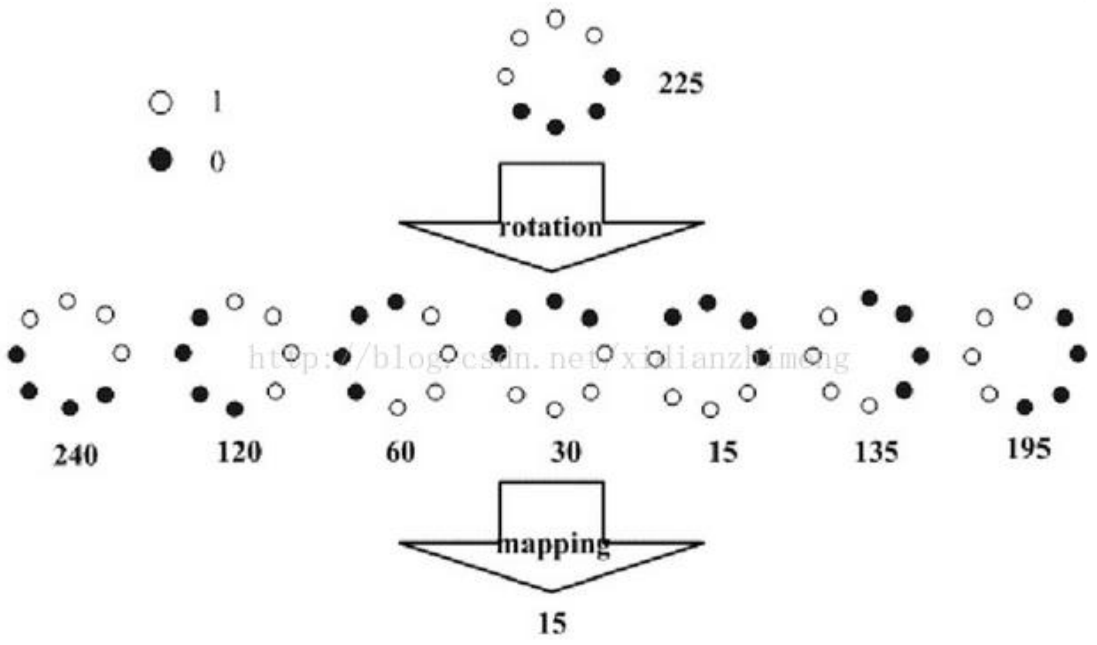
In terms of the kernel function of the SVM classifier, we found that the rbf kernel performs a bit better than the linear kernel. The reason maybe that, the training set is large enough and all the samples are of comparatively high quality.

4.3 LBP Operators

We try two kinds of operators of LBP: rotation-invariance operators and the uniform pattern. They are used to conduct dimension reduction.

- **Uniform Pattern**: if an LBP number in a binary form has at most two jumps from 1 to 0 or 0 to 1, it is called a uniform pattern class. All the other LBP numbers not belonging to any uniform pattern class is classified as one class. For example, 00000000 (0 jump), 10001110 (one jump from 0 to 1 and two jumps from 1 to 0) are both uniform pattern class. This operator was first proposed by Ojala and others, and is able to reduce dimension of feature vectors from 256 to 58.

- **Rotation-invariance**: for the LBP number of a pixel, rotate its neighborhood to look for the minimum binary number, and let it be the actual LBP number of the pixel. For example, the following numbers are the same in rotation:



This operator was first proposed by Maenpaa and others. It can reduce the dimension from 256 to 36.

The following table shows the performance of the two operators:

Table 2

Model Performance based on different operators						
F1 Score		LBP feature				SVM
best	average	Image size	Cell size	Operator	dimension	Kernel
0.932	0.913	36	6	None	9216	rbf
0.931	0.912	36	6	uniform pattern	2088	rbf
0.936	0.907	60	10	None	9216	rbf
0.934	0.913	60	10	uniform pattern	2088	rbf
0.837	0.814	60	10	rotation-invariance	1296	rbf

We can see that the uniform pattern significantly reduce the dimension while maintaining the performance. On contrary, the rotating-invariant operator works badly. Our guess is that implementing the rotating-invariant operator in a 3*3 area, which is too small, diminish the differences of features, making the boundary for classification more vague and indefinite.

4.4 Data Normalization

The word "normalization" is used informally in statistics, and so the term normalized data can have multiple meanings. In most cases, when you normalize data you eliminate the units of measurement for data, enabling you to more easily compare data from different places. Some of the more common ways to normalize data include:

- **Rescaling data to have values between 0 and 1.** This is usually called feature scaling. One possible formula to achieve this is:

  $$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

- **Transforming data using a z-score or t-score**. This is usually called **standardization**. Standardization transforms data to have a mean of zero and a standard deviation of 1. This standardization is called a **z-score**, and data points can be standardized with the following formula:

  $$z_i = \frac{x_i - \bar{x}}{s}$$

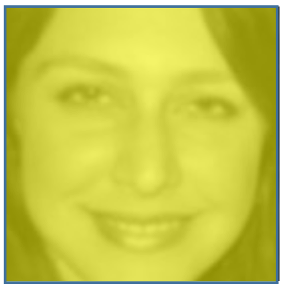
  Where:
  x_i is a data point ($x_1, x_2 \ldots x_n$).
  $\bar{x}$ is the sample mean.
  s is the sample standard deviation.
- **Standardizing residuals:** Ratios used in regression analysis can force residuals into the shape of a normal distribution.
- **Normalizing Moments** using the formula μ/σ.
- **Normalizing vectors (in linear algebra) to a norm of one.** Normalization in this sense means to transform a vector so that it has a length of one.

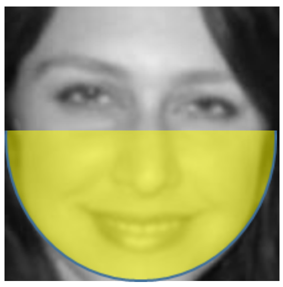
We choose to rescale data to have values between 0 and 1, and use the most common linear formula above.

4.5 Other Improvements

4.5.1 In Feature Extraction

In collecting features, we come up with an idea of 'half-face detect'. That means, we only calculate the feature of the lower half of the face in a semi-circle area.


Traditional Feature Area


Our Feature Area

We believe that not all parts of the face are changing while one is smiling. If we collect feature of the irrelevant parts, we may encounter a problem called overfitting, which will impair the performance. For example, some people in the training set wear sunglasses or wear hats, while others don't. And apparently hair style don't determine one is smiling or not. So if we put the eyes and hair into the comparison items, this may 'confuse' the classifier, which leads to the classifier determining the wrong border of positive and negative sample's feature vectors in the hyperspace.

4.5.2 In Real-time Detect

When we were testing our real-time smile detect, we found that the program is when the person is smiling, the prediction is not stable. It's frequently oscillating between smile and not smile. But in the case that one was not smiling, the prediction was quite stable. This phenomenon tells us that the prediction of the positive samples is not very reliable.

Our solution is to make it easier to respond to smiling faces, In other words, make it harder to judge it as not smiling. In detail, we reprogramed it such that only if the model predicts the person is not smiling for three times in a row, then it will show that he or she is indeed not smiling. This method work quite effectively

Another feature we add is to judge multiple faces at the same time. It just needs a little change, so we will not talk about it in detail.

5. OTHER POSSIBLE IMPROVEMENTS

5.1 Head pose

As peoples in different pictures may not have the same positions, so their facial expression may have some differences even though they are all smiling. But the computers can not directly think about the differences of the positions. So, the position of the head may have some influence to the result. Then it is possible for us to make some adjustment according to the Head Pose (yaw, pitch, and roll parameters, in radians). To certain extent, all the faces will be transform into the same position in this way and it will probably let the final results have a higher score.

5.2 other facial features

As we all know, the most significant features to judge a person is smiling or not is to observe its mouth. But except mouth, some of the other facial features may also help to judge whether the person is smile. The eyes and also the eyebrows will look a little different if the person is smiling. Besides, the cheeks and even the strings reflecting intensity of specific muscles on face may be one of the possible improvement. This is also called action unit, or AU. It's possible that all facial expressions can be expressed through AU or combination of AU.