Interacting with Software Using Facial Expressions Gestures

Abhinav Jain - 163050012
Akshat Jaiswal - 163050069
Kalyani Dole - 163050022
Sanket Gandhare - 163050058

Indian Institute Of Technology Bombay

November, 2016

Overview

Objective

The Objective of the project is to detect the facial expression of a person in the most optimal way using different classification models. Based on the accuracy levels of the results we will compute the best model.

Motivation

1. The automatic recognition of facial expressions has been an active research topic since the early nineties.
2. There have been several advances in the past few years in terms of face detection and tracking, feature extraction mechanisms and the techniques used for expression classification.
3. This proposal aims to build upon the existing research and apply the work in real life.
4. Facial Expressions are the facial changes in response to a person's internal emotional states, intentions and social communications.
5. Till now, the de-facto method of interacting with softwares has been through various pointing and typing devices.
6. With this project, we aim to bridge that gap. We aim to build a mechanism through which we can interact with software using facial expressions and gestures.

Project Description

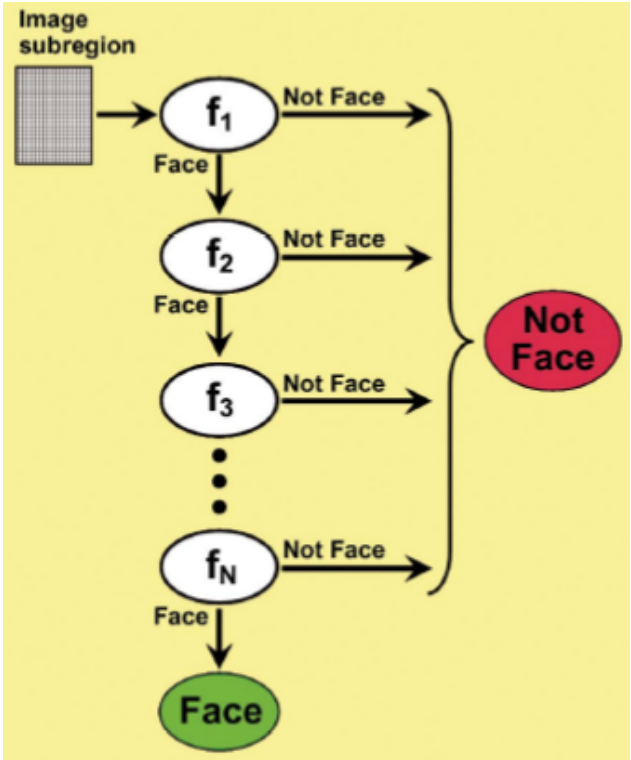
The project has been divided into four parts:

Data Preprocessing

1. First step was to take out 2 images out of the various shots, one containing the neutral face and the second containing the expression.
2. Next step consisted of sorting the images into categories based on the expression. These sorted images were put into respective folders viz-a-viz. **anger contempt disgust fear happiness neutral sadness surprise**.
3. This generated the preprocessed data.

Face Detection

A facial recognition system is a computer application capable of identifying or verifying a person from a digital image or a video frame from a video source.It is a process to detect face from an image, a set of images or a video. This phase of our project makes use of the Haar Feature-based Cascade Classifier for Object Detection which comes under Open CV. OpenCV already contains many pre-trained classifiers for face, eyes, smile etc.



We are using following 4 xml files for face detection :

1. haarcascade_frontalface_alt2.xml
2. haarcascade_frontalface_alt_tree.xml
3. haarcascade_frontalface_alt.xml
4. haarcascade_frontalface_default.xml

Then, we need to load the image into grayscale mode.We use **v2.CascadeClassifier.detectMultiScale()** to find various facial features and it is defined like this: *faceDet.detectMultiScale(frame, scaleFactor=1.1, minNeighbors=10, minSize=(5, 5), flags=cv2.CASCADE_SCALE_IMAGE)*

Here the different parameters are :

- **Image**:Matrix of the type CV_8U containing an image where objects are detected.
- **ScaleFactor**:Parameter specifying how much the image size is reduced at each image scale.
- **minNeighbors**:Parameter specifying how many neighbors each candidate rectangle should have to retain it.
- **flags**:Parameter with the same meaning for an old cascade as in the function cvHaarDetectObjects.
- **minSize**:Minimum possible object size. Objects smaller than that are ignored.
- **maxSize**:Maximum possible object size.

If faces are found, it returns the positions of detected faces as Rect(x,y,w,h).

Feature Selection and Representation

Features are little bits of information that describe the object or object state that we are trying to divide into categories.

1. For all detected face instances we individually draw Facial Landmarks with the predictor class.
2. Then we store the two landmarks into lists. The mean of both axes is calculated to determine the Center of Gravity.
3. The next step is to get distance between each point and the central point in both axes.

1. Distance calculation
   $xmean = np.mean(xlist)$
   $ymean = np.mean(ylist)$
   $xcentral = [(x - xmean) for x in xlist]$
   $ycentral = [(y - ymean) for y in ylist]$
2. Angle Caalculation :
   $anglenose = int(math.atan((ylist[26] - ylist[29])/(xlist[26] - xlist[29])) * 180/math.pi)$

Using the distance and angle information, we can get the vector using numpy functions.

Facial Expression Detection

The last thing after getting the images in vectorized for is to classify the data with respect to facial expression. We have 8 different expressions for classification -

1. anger
2. contempt
3. disgust
4. fear
5. happiness
6. neutral
7. sadness
8. surprise

Sampling: The data is sampled into ratio of 80:20(training vs prediction). We use the training data using different models to train the models and then use it for getting the output for the prediction input. The output is then compared with its classes. Ans using this accuracy is calculated for the model.

DataSet

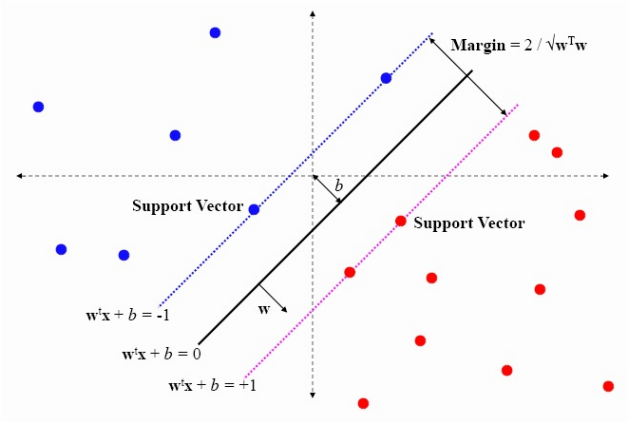
The dataset that we used goes by the name **'Cohn-Kanade AU Coded Expression'**. DataSet includes 486 sequences from 97 posers. Each sequence begins with a neutral expression and proceeds to a peak expression. The peak expression for each sequence in fully FACS coded and given an emotion label. The emotion label refers to what expression was requested rather than what may actually have been performed.

Approaches

Here are the approaches we tried:

SVM

Support vector machines are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis.
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.
We have used it in this way :
$clf = SVC(kernel =' linear', probability = True, tol = 1e-3, verbose = True)$
Using the above line of code we set the classifier as a support vector machines with polynomial kernel.



Fisherface Classifier

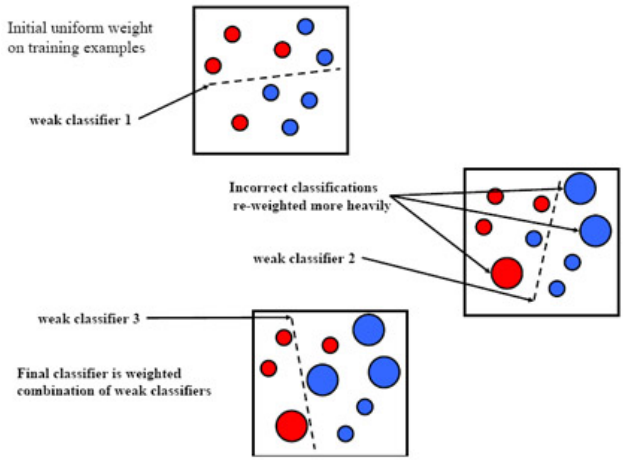
A key problem in Face and pattern recognition is to define an appropriate data representation for the task at hand. One way to represent the input data is by finding a subspace which represents most of the data variance. This can be obtained with the use of Principal Components Analysis (PCA).
Following are the steps followed:

- When PCA is applied to face images, PCA yields a set of eigenfaces.
- These eigenfaces are the eigenvectors associated to the largest eigenvalues of the covariance matrix of the training data.
- The eigenvectors thus found correspond to the least-squares (LS) solution.

This is a powerful way to represent the data because it ensures the data variance is maintained while eliminating unnecessary existing correlations among the original features (dimensions) in the sample vectors.

AdaBoost

1. The output of a number of learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier.
2. AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers.
3. In some problems it can be less susceptible to the overfitting problem than other learning algorithms. The individual learners can be weak, but as long as the performance of each one is slightly better than random guessing (e.g., their error rate is smaller than 0.5 for binary classification), the final model can be proven to converge to a strong learner.



$$H(x) = sign(\alpha_1 h_1(x) + \alpha_2 h_2(x) + \alpha_3 h_3(x))$$

Results

When the above methods were used we got different accuracy results:

Model	**Accuracy**
SVM	83% – 84%
Fisherface classifier	71%
AdaBoost	80%

Conclusion

The above use of different models and different techniques affected the prediction accuracy to a significant level. Of all the models SVM proved to be the best one as it gave the best accuracy for the model.