UNIVERSITÀ DEGLI STUDI DI CAGLIARI

FACOLTÀ DI SCIENZE

Corso di Laurea Magistrale in Informatica

Progetto Computer Vision: **FACE RECOGNITION**

Docente:
Prof. **Giovanni Puglisi**

Studente:
Andrea Corriga (65088)

ANNO ACCADEMICO 2017-2018

Summary

1. Introduction

The goal of this project was to develop a Face Recognition application using a **Local Binary Pattern** approach and, using the same approach, develop a real time Face Recognition application.

At high level the system is able, with a set of familiar faces, to recognize with a certain accuracy a new face. To do this, different methods were used in order to compare the results in terms of accuracy.

This project was developed for Computer Vision course at University of Cagliari, supervised by prof. Giovanni Puglisi.

1.1 Used tools

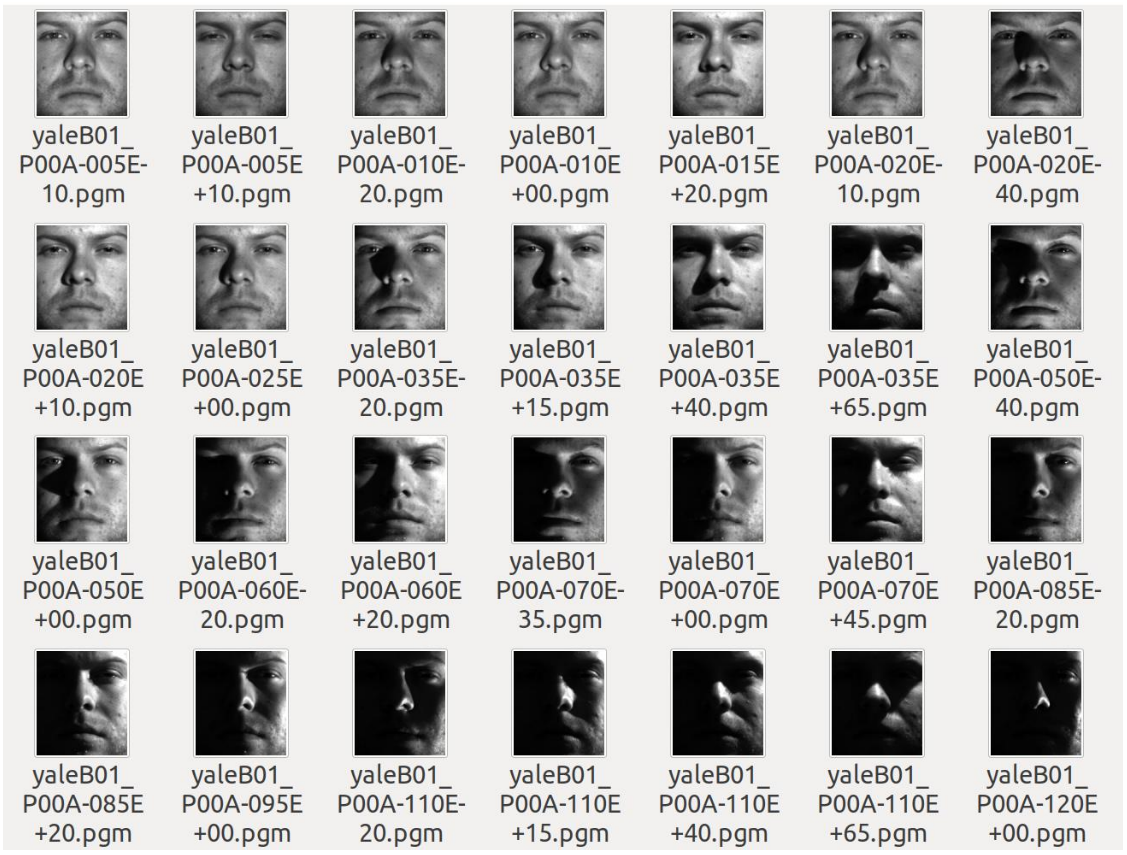
Most of the project has been developed using Python as programming language, open source libraries and open source datasets. In particular it's been used:

- Python 2.7.14
- OpenCV
- Sklearn
- Extended Yale B Faces Dataset
- Pil

Extended Yale B Faces Dataset includes 2414 frontal pictures of faces belonging to 38 different subjects. For each subject there are 64 photos taken in different light conditions.

Non open source tools:

- Matlab



Picture 1: the Yale Faces Dataset B

1.2 Face Recognition problem

Over the last ten years or so, face recognition has become a popular area of research in computer vision and one of the most successful applications of image analysis and understanding. A facial recognition system is a technology capable of identifying or verifying a person from a digital image or a video frame from a video source. For a human is very easy to perform the face recognition process but the same process is not easy for a computer. Computers have to deal with numerous interfering factors related to treatment of images such as: wide chromatic variations, different and different angles of view. Beyond this there are other factors that can be affect the face recognition process like: occlusion, hair style, glasses, facial expression etc.

There are multiples methods in which facial recognition systems work, but in general, they work by comparing selected facial features from given image with faces within a database.

1.3 Local Binary Pattern

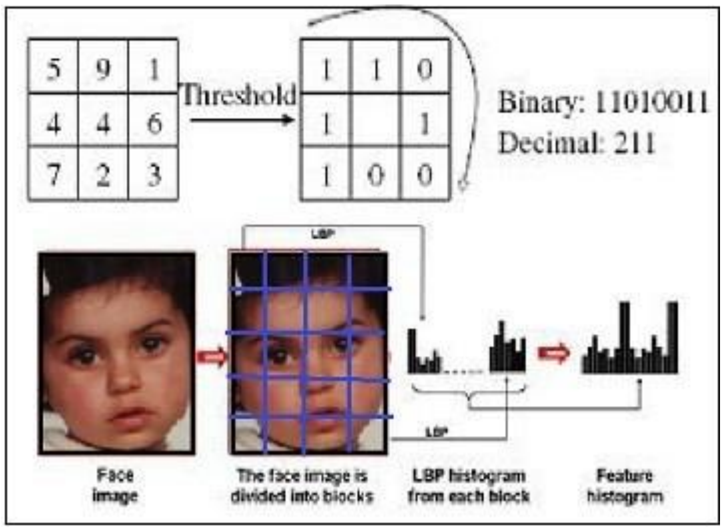
Local binary patterns (LBP) is a type of visual descriptor used for classification in computer vision. In this project **LBP** operator was used to filter the features of facial textures.

LBP transforms image blocks into an array of labels. Such labels (or their statistics, for example histograms) are used as features. Various versions have been developed in the state of the art.

In the basic version of the LBP we consider the values of a 3x3 pixel neighborhood. For each pixel in a cell, compare the pixel to each of its 8 neighbors (on its left-top, left-middle, left-bottom, right-top, etc.). Follow the pixels along a circle, i.e. clockwise or counter-clockwise. Where the center pixel's value is greater than the neighbor's value, write "0". Otherwise, write "1". This gives an 8-digit binary number (which is usually converted to decimal for convenience).

In its generalized version, each pixel is considered in a circular neighborhood of **P** points with radius **r**. If the elements do not fall on a single pixel, their values are obtained by interpolation. Generally the number of neighboring pixels chosen is 8.

In this project a multi-blocks LBP were used.



Picture 2: an example of multi block Local Binary Pattern

2. Project structure

In this chapter we'll be discuss the project structure and his general function.

```
Computer-Vision-Project
│   LICENSE
│   README.md
│   main.py
│   rotate.py
│
└─── algorithms
│     LBP.py
│
└─── model
│     (here will be knn/naivebayes/svm.pkl)
│
└─── utils
│     dataset.py
│     utils.py
│
└─── RealTime (subproject)
│           utils
│     │     utils.py
│           ------------------
│     create_data.py
│     face_recognize.py
│     haarcascade_frontalface_default.xml
```

The folder that starting with a capital letter represents a sub-project, the other folders are used to store the different libraries.

In the **root** folder there are the `algorithms` folder, where there is the basic Local Binary Pattern implementation made by myself. The `datasets` folder contain the zips of the datasets used to test the project. In the `model` folder will be saved training data values, in order to perform the prediction without training the classifier again. In the `utils` folder there are some helper function.

The **Realtime** folder contain the sub-project that perform a real time face recognition using the pc camera. It contain two main scripts: `create_data.py` that generate the dataset shooting some photos using the camera. `face_recognize.py` train the classifier and perform the real time face recognition. More details will be given in the next chapters.

3. Run the project

3.1 Install dependencies

In order to run, test and modify the source code must be installed the following packages.

```
#CV2
sudo apt-get install python-opencv

#PIP
sudo apt-get install python-pip
pip install Pillow

# SKLEARN
pip install -U scikit-learn

sudo apt-get install build-essential python3-dev python3-setuptools
sudo apt-get install python3-numpy python3-scipy
sudo apt-get install libopenblas-dev

# DLIB
sudo apt-get install build-essential cmake
sudo apt-get install libgtk-3-dev
sudo apt-get install libboost-all-dev

# OTHER
sudo pip install scikit-image
sudo pip install dlib
```

3.2 Run Main project

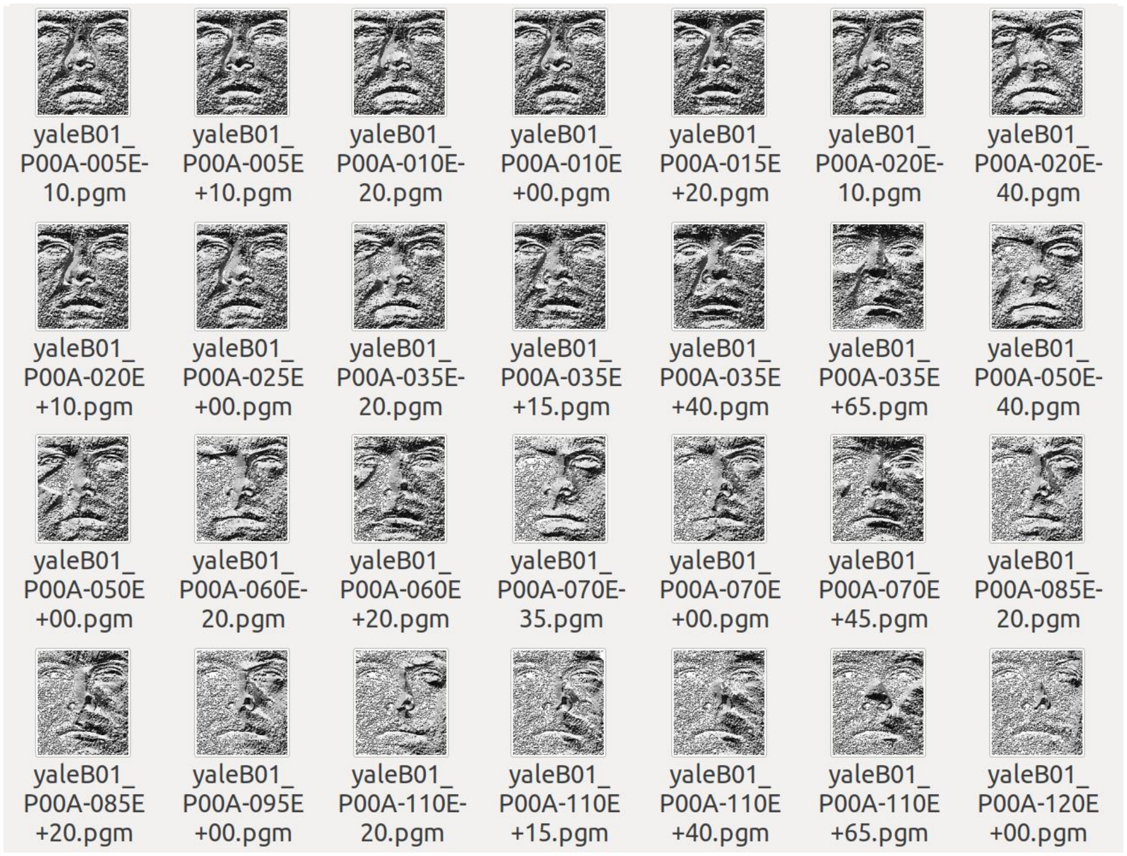
First of all you have to unzip the YaleFaces.zip and YaleFaces_small.

Launch the `python main.py` with the following parameters:

```
--dataset [datasetName]
--classifier [svm, knn, naivebayes]
--training
--histEq
--output
```

By default launching only `python main.py` the script use YaleFaces dataset without training (loading from model folder). If there are not model in your model folder you must use `--training`. `--histEq` is helpful because perform an histogram equalization before calculating the LBP. If `--output` is setted the script will produce inside

`./datasets/your_chosen_algorithm_/your_chosen_dataset/` the PNG of the LBP calculated for each image.



Picture 3: the LBP result on Yale Faces Dataset

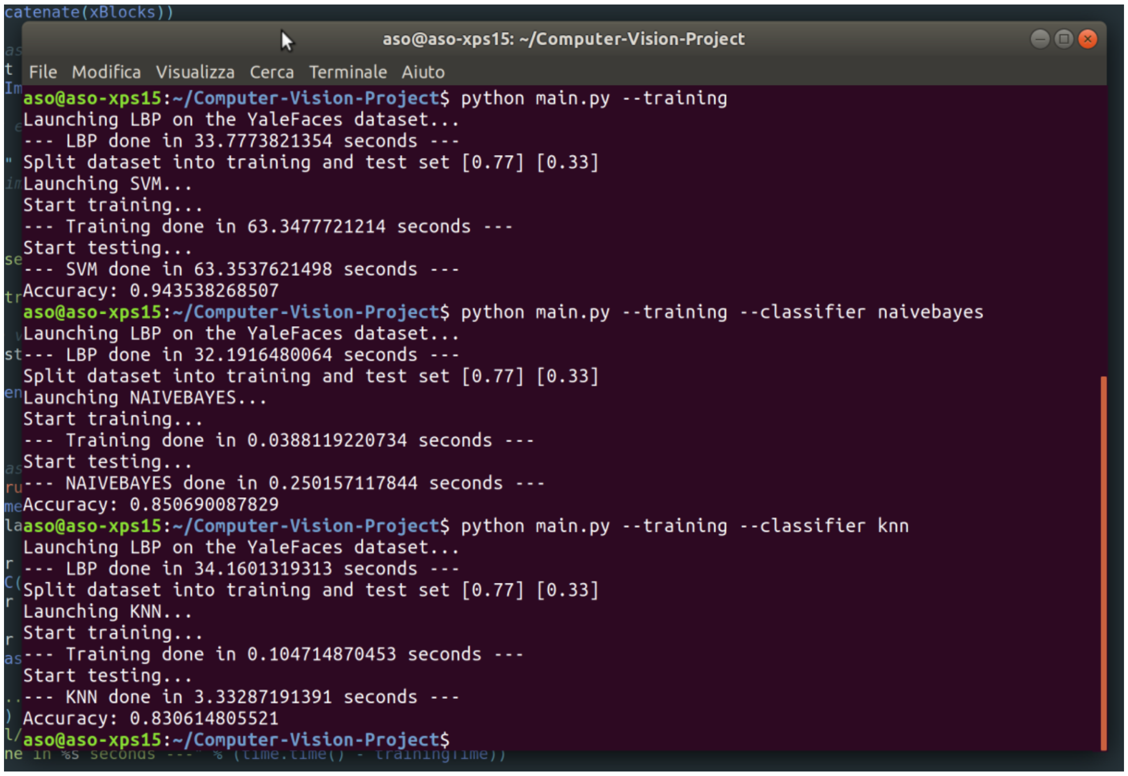
With `--classifier` you can choose which classifier to use between SVM, KNN and NaiveBayes (obviously for each classifier you must perform the training of the model).

By default LBP is performed splitting the images in 12x12 blocks.

Usage example

```
python main.py --dataset YaleFaces --classifier svm

python main.py --training --output --histEq
```


aso@aso-xps15: ~/Computer-Vision-Project
File Modifica Visualizza Cerca Terminale Aiuto

```
aso@aso-xps15:~/Computer-Vision-Project$ python main.py --training
Launching LBP on the YaleFaces dataset...
--- LBP done in 33.7773821354 seconds ---
Split dataset into training and test set [0.77] [0.33]
Launching SVM...
Start training...
--- Training done in 63.3477721214 seconds ---
Start testing...
--- SVM done in 63.3537621498 seconds ---
Accuracy: 0.943538268507
aso@aso-xps15:~/Computer-Vision-Project$ python main.py --training --classifier naivebayes
Launching LBP on the YaleFaces dataset...
--- LBP done in 32.1916480064 seconds ---
Split dataset into training and test set [0.77] [0.33]
Launching NAIVEBAYES...
Start training...
--- Training done in 0.0388119220734 seconds ---
Start testing...
--- NAIVEBAYES done in 0.250157117844 seconds ---
Accuracy: 0.850690087829
aso@aso-xps15:~/Computer-Vision-Project$ python main.py --training --classifier knn
Launching LBP on the YaleFaces dataset...
--- LBP done in 34.1601319313 seconds ---
Split dataset into training and test set [0.77] [0.33]
Launching KNN...
Start training...
--- Training done in 0.104714870453 seconds ---
Start testing...
--- KNN done in 3.33287191391 seconds ---
Accuracy: 0.830614805521
aso@aso-xps15:~/Computer-Vision-Project$
```

Picture 4: screenshot of main.py script

3.3 Run Real Time project

This section basically work in two steps:

- Creation of the dataset
- Training and real time classification

First of all launch the `python create_data.py` with the following parameters:

`--name`

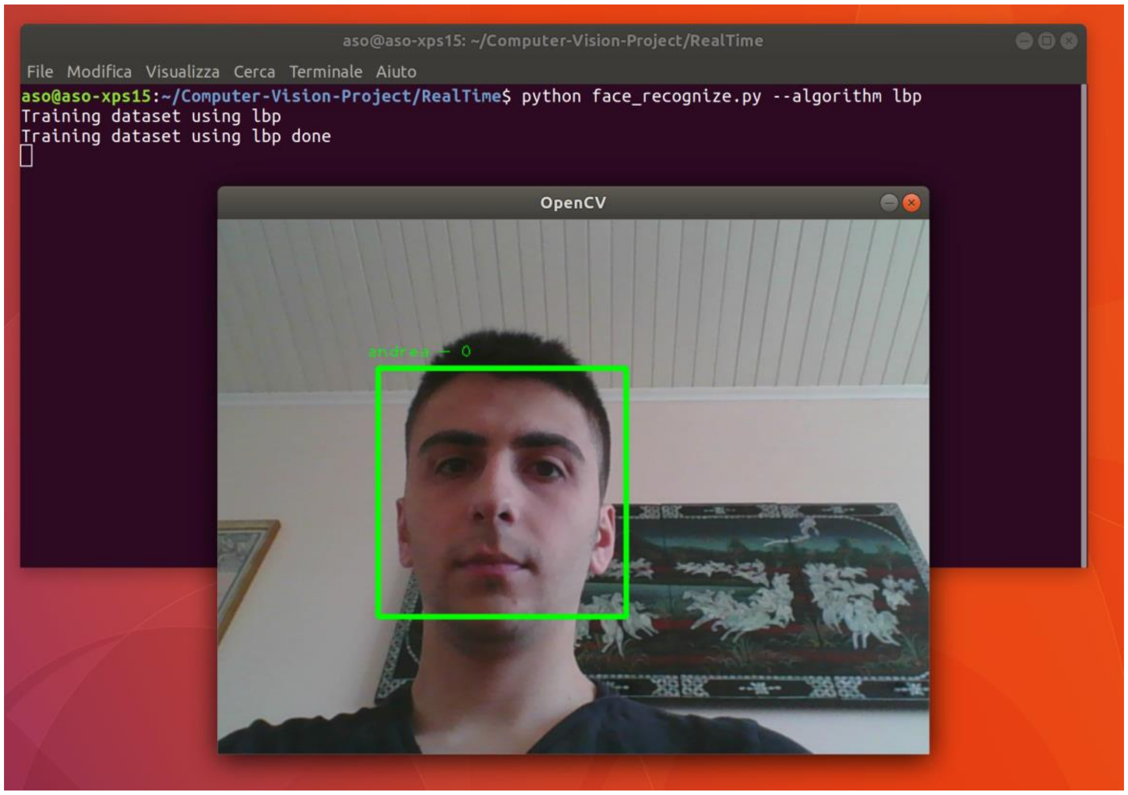
Launching the script it will create inside `./datasets/CHOOSEN_NAME/` some photos shooted using the PC camera. The script use `CascadeClassifier`in order to perform Face Detection and cut out the face rettangle. Currently I choose to save 30 photos per subject.

Now is it possible to perform Face Recognition in real time launching `python face_recognize.py` with the following parameters:

`--algorithm [lbp, fisherface]`

The script, first of all, use the previously created dataset to train the classifier. If LBP is chosen it will be used SVM as classifier, else is used a general default CV2 classifier.

After the training, `CascadeClassifier` is used to identify the face inside the scene and, when a face is identified, the classifier try to recognize it.

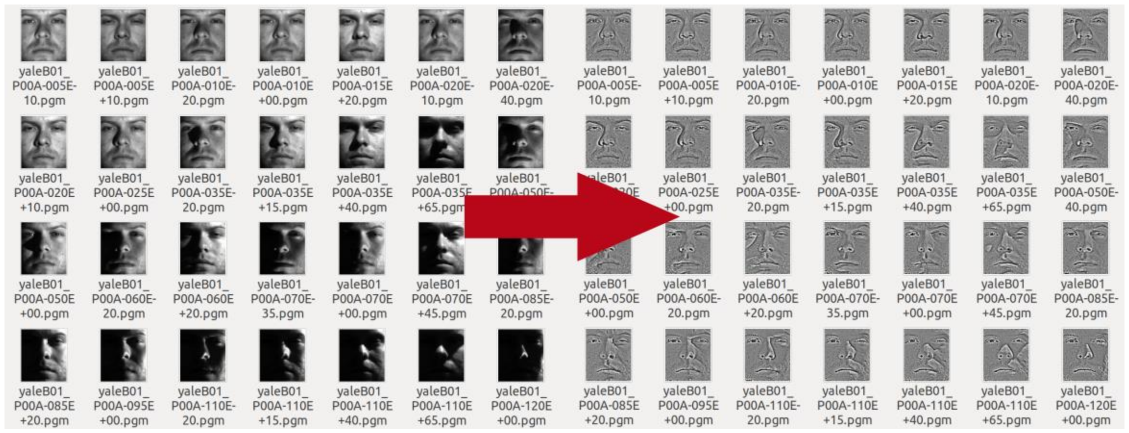

aso@aso-xps15: ~/Computer-Vision-Project/RealTime
File Modifica Visualizza Cerca Terminale Aiuto
OpenCV
andrea - 0

```
aso@aso-xps15:~/Computer-Vision-Project/RealTime$ python face_recognize.py --algorithm lbp
Training dataset using lbp
Training dataset using lbp done
```

Picture 5: a screenshot of real time application

3.4 Normalization Illumination

Starting by this paper: Enhanced Local Texture Feature Sets for Face Recognition under Difficult Lighting Conditions (authors page: Xiaoyang Tan's Publications) and his open source matlab code I perfomed a normalization of lighting as you can see in the picture.


Picture 6: the result of Normalization Illumination

4. Experiments

The features extraction methods and the accuracy was tested changing:

- Classifier
- The blocks size of LBP
- Performing the normalization of illumination

The tests were done splitting the dataset 77% for training and 33% for tests. The final values is the result of the average value of 5 tests with different training/test set.

Below there are my results obtained with YaleFaces dataset using this PC: *Dell XPS 15 9550, with Intel i7 6700HQ Skylake @ 2.60GHz.*

Test with 3x3 blocks using LBP

Average time for calculating the LBP: 14 seconds.

Classifier	Accuracy without histogram equalization	Accuracy with histogram equalization	Accuracy with Normalization Illumination
LinearSVM	0.05	0.07	0.05
KNN	0.48	0.48	0.69
NaiveBayes	0.35	0.37	0.66

All classifiers get poor result without the Matlab scripts. With this blocks size LinearSVM always get the worst results.

Test with 6x6 blocks using LBP

Average time for calculating the LBP: 18 seconds.

Classifier	Accuracy without histogram equalization	Accuracy with histogram equalization	Accuracy with Normalization Illumination
LinearSVM	0.40	0.55	0.40
KNN	0.65	0.66	0.90
NaiveBayes	0.70	0.70	0.93

With normalization illumination the accuracy using SVM is very variable. It get an accuracy starting by 0.33 to 0.49. Also the other classifier is very variable, but not so much. In general NaiveBayes get the best result in all conditions.

Test with 12x12 blocks using LBP

Average time for calculating the LBP: 32 seconds.

Classifier	Accuracy without histogram equalization	Accuracy with histogram equalization	Accuracy with Normalization Illumination
LinearSVM	0.94	0.95	0.99
KNN	0.81	0.81	0.97
NaiveBayes	0.82	0.83	0.96

In this case linear SVM get the best results and using the normalization script we come close to the 100%.

In general the first thing that we can notice is that splitting the images into blocks in order to calculate the LBP is essential to obtain satisfactory results.

Splitting the images into 12*12 get the best results.

Normalize the lighting improve the classifier score in the most of the case, except for the LinearSVM with larger block size (3x3 and 6x6).

KNN get average score, but we must specify that this classifier is instance based, so it is not good in a realistic case. Some faces can't be classified if there are not a very similar element in the training set.

NaiveBayes at least, get a good result in many different case. Splitting the images in 12x12 blocks it get worse result than LinearSVM (without Normalization Illumination) yes, but still get good result with smaller blocks.

5. Conclusion

The goal of this project was to develop a Face Recognition application using a **Local Binary Pattern** approach and after that, using the same approach, develop a real time Face Recognition application.

The goal can be considered achieved with excellent results.

In general we can say that LinearSVM perform the best results but need to split the images in very small blocks (at least 12x12 blocks) and the scene must be under control. These compromises are burdensome in terms of computational load.

With different and smaller blocks size LinearSVM get the worse result if compared with KNN and NaiveBayes.

NaiveBayes work good in the most of the case, even if we split the images with large blocks (3x3).

6. References

- Python
- Matlab
- OpenCV
- Sklearn
- Enhanced Local Texture Feature Sets for Face Recognition under Difficult Lighting Conditions
- Multiresolution gray-scale and rotation invariant texture classification with local binary patterns
- Face Description with Local Binary Patterns: Application to Face Recognition