Date matching using Face Recognition

Davide Berdin, Tobias Famulla
{davide.berdin.0110},{tobias.famulla.7003}@student.uu.se

Machine Learning Project
Department of Information Technology

June 30, 2015

Contents

Abstract

In this project we present a Date Matching application using Face Recognition. For each user that took part of our experiment we created different models using different features extractions and classifiers as well. Our analysis shows a comparison between four different algorithms for human face features extraction, such as Eigenfaces, Fisherfaces, Local Binary Patterns Histograms and Identity. Moreover, for each feature extraction algorithm we applied the K-Nearest Neighbour classifier with Euclidean and Chi-square distances.

Chapter 1

Introduction

The general idea of our project is based on the question how predictable humans are. As mentioned in the Artificial Intelligence and Machine Learning courses, artificial intelligence in general is associated with a touch of magic, which is hard to understand by humans. The same picture is drawn concerning another topic: love and attraction. Despite some clear research results in biology and psychology humans try to believe that these topics are not rational explainable and not calculable.

We thought it might be an interesting project to bring these two topics together. Although we assume that attraction is a quite complicated field and can not be solved in a small project, a very successful app, Tinder, breaks the properties of attraction down to a simple binary question: Like or Don't-Like based just on the look of the persons.

The condensed question for our project concerning the further will be whether human taste concerning the face of a person is predictable, and how is it predictable. As we found out, we are not the first people asking and an application called "Tinderbox"[1], which does face recognition to automatically rate persons on Tinder, exist. To focus more on the face recognition and machine learning aspect of this question, we gathered data concerning pictures and tested different algorithm for the purpose to select people the user might like.

In the following we describe how we gathered data to test on, which algorithms we looked at and how we tested the data.

Chapter 2

Data Gathering

To test Face Recognition and Machine Learning algorithms we needed data. To get these data we decided to program a webapplication, which reimplements the functionality of Tinder. For this app we used the framework Flask[2] in connection with SQLAlchemy[3], SQLite[4] and WTForms[5]. Testers created a user, got pictures of a person shown and decided whether they liked the person. After that the next person appeared. We saved the data to a database and to split these data in training and test-sets to challange the algorithms.

Because we needed a high amount of profile pictures of a certain type (people in age-range which is the preferences of the test-users), we decided to gather these from Tinder directly. For this we used a public description of the reverse-engineered Tinder-API[6] and wrote a script to download pictures from people of different cities using our own Facebook-Tokens used by Tinder [1]

After preprocessing the photos (see Chapter 5) we got around 300 male and the same amount of female people, which had three pictures each to use for face recognition.

The webapp has been used by a pool of 8 testers and got 5 complete datasets including our own, meaning that each user rated all pictures according to their sexual preference. The uncomplete datasets contained a certain amount of rated persons, but not all people of a gender.

In the following we describe the different algorithms applicable for our problem and later on how we used them.

[1] Because the pictures are freely available to all Tinder-users, we assume that using them in a scientific project should not be problematic. The web application was just accessable by a small group of users to generate the data for the project. Due to legal concerns we will not make the picture-set publicly available, but provide a script with the sourcecode of the project.

Chapter 3

Face Detection and Face Recognition

Face detection is a system that tries to identify human faces in digital pictures. In general, object detection is used in multiple real-world applications nowadays, such as: localization of objects in images or videos, identification of pedestrians or cars, etc. Although, face detections algorithms focus primarily on the detection of frontal human faces [7]. In our approach we used **Haar Feature-based Cascade Classifier**, a famous classifier for object detection (see 3.1).

A **facial recognition** system is an application for automatically identifying a person from either a picture or a video. One of the ways to do this is by comparing selected facial features from the image and a facial database [8]. Given an image in which the face of a subject is present, a facial recognition algorithm tries to identify human facial features by extracting features from the image itself. In general, it will analyze the relative position of eyes, nose, mouth, etc. The position is not the only variable involved: in fact, size and shape are taken in consideration for a more accurate analysis. All these features will be then used to search for other images in a dataset that will match the original face.

In general, recognition algorithms can be of two kinds: geometric and statistical. The first approach distinguish features, whereas the second approach converts a picture into values and then compare those values with templates in such a way to eliminate the variance.

General methods such as *Principal Component Analysis* (PCA), *Hidenn Markov Model* (HMM) and *Linear Discriminate Analysis* (LDA) are used in face recognition systems. Commons systems are *Eigenfaces* using PCA (see 3.2) and **Fisherfaces** using LDA (see 3.3).

3.1 Haar Feature-based Cascade Classifier for Object Detection

The following sections describe the classifier algorithm used to detect specific human face features in an image. **Haar features** are simple and inexpensive image features based on intensity differences between rectangle-based regions that share similar shapes to "square-shaped" functions [9][10][11].

3.1.1 Local invariant feature detectors

Local feature-based methods are an often used group of object recognition methods[12]. Haar features are built on the notion of trying to define a set of data for *processing* as well as for providing invariance to different transformations. In the case of Local invariant features, global features are not considered, instead, local feature detection approaches search for **anchor** points in the picture where local are defined by differences in either texture or intensity. An anchor point, or insterested point, is the type of shape that the algorithm is looking for and it can be an edge or a corner. To find these textures or intesity changes, a **kernel** runs over the image. A kernel is a set of pixels computations of computing the gradient change value (the derivative) within this set. These features can be corners, T-junctions, circles, etc., depending on what kind of detector is used [13][12].

3.1.2 Haar classification

The Haar feature-based cascade classifier has been initially proposed in [14] and improved in [15]. In fact, the face detection algorithm seeks for specific Haar features of a human face. When a feature is found, the algorithm sends the face candidate to the next step of detection. Basically, a face candidate is a rectangular section of the initial image named **sub-window**. According to [16], a sub-window has typically a fixed size of 24x24 pixels. Moreover, the sub-window is often scaled in order to get a set of different size faces [17].

The classifier is called **cascade** (a degenerate decision tree), because the resultant classifier consists of multiple weak classifiers, called stages, that are applied one after another to a region of interest. They are called weak classifiers because if they are taken singularly they cannot classify an image, instead, if combined with others, they can form a strong classifier.

The detection process starts with the evaluation of the first classifier. If its result is positive then the classifier triggers the evaluation of the second one. Given a positive evaluation of the second classifier, a third one is triggered, and so on. As soon as a negative outcome is given, the sub-window is rejected. The evaluation of each classifier is done by a **stage comparator** where all the previous stages will be summed up and compared to a stage threshold to determine if the stage should be passed or not. If all the stages are passed, the face candidate is concluded to be a face [17].

3.2 Eigenfaces

In computer vision problem, the *eigenfaces* in human face recognition is the name given to a set of **eigenvectors**. An eigenvector of a linear transformation between vector spaces is a vector where the image is the vector itself moltiplied by a scalar vector, called **eigenvalue**. This can be written as follow:

$$A\mathbf{v} = \lambda\mathbf{v} \tag{3.1}$$

where λ is the eigenvalue associated with the eigenvector $\mathbf{v}$ and $\mathbf{A}$ is the square matrix. This approach was for the first time used in face classification in [18]. Each eigenvector is extracted from the **covariance matrix**, that is a matrix that represents the variation of each variable respects all the others (including the variable itself). The covariance is the value that represents the variation of two variables together. In other words, the eigenface is a set of values of linearly uncorrelated variables of a distribution of faces where all the pixels in each image is a vector in N-dimensional space. This leads to reduce the number of dimension by using a smaller set of basis pictures to represent the original images for the training set. To classify faces, the algorithm simply compare the faces representation in the basis set.

To generate the eigenfaces' set, the process called *principal component analysis (PCA)* is used over a large set of images with different faces. When an eigenface is created, the resultant image will appear with some light and dark areas arranged to a specific pattern. Figure 3.1 shows how an image is transformed when the eigenfaces are created. Patterns are selected to be evaluated and scored on different features of a human face. Many patterns are involved in facial recognition: evaluation of the face symmetry, facial hair style, position of the hairline, nose size, and so on. Due to the nature of the algorithm, the dimension reduction method allows the system to represent many subjects with a small dataset. The sensitivity of the recognition system depends on how large are the reductions applied to the image because when the variation between the image we want to recognize and the examined picture is large, the system starts to fail.

For each subject's face in the dataset, a weight describing the contribution of each eigenface gives to the image, is stored in a structure. When we need to classify a face, a new set of weights of the candidate image is projected onto the collection of eigenfaces providing a collection of weights representing the examined face. All these weights are then compared with all the weights in the previous set for finding the closest match. A typical method for classification used with eigenfaces is *K-NN (nearest neighbour method)* using *Euclidean Distance* between two vectors where the closest subject is given by the minimum distance [18].

(a) Training set of pictures

(b) Eigenfaces representation

Figure 3.1: Transofrmation from normal pictures to the eigenfaces

3.3 Fisherfaces

The usage of *eigenvectors* is indeed a good way to represent images, because it guarantees that the data variance is kept whilst the algorithm eliminates unnecessary existing correlations between the original features (dimensions) in the sample vectors [19]. When we need to deal with classification instead of representation, eigenfaces may not yield the most desirable results, and since for our project we want the best classification possible the **Fisherfaces** is introduced. The algorithm uses a techinique called *Linear Discriminant Analysis (LDA)*. When the LDA is used to find the subspace representation of set of pictures, the resulting basis vectors that define this space are known as *Fisherfaces* [19]. According to [20], Fisherfaces appears to be the best at extrapolating and interpolating over variation in lighting. Figure 3.2 shows the face features highlights of the Fisherfaces algorithm.

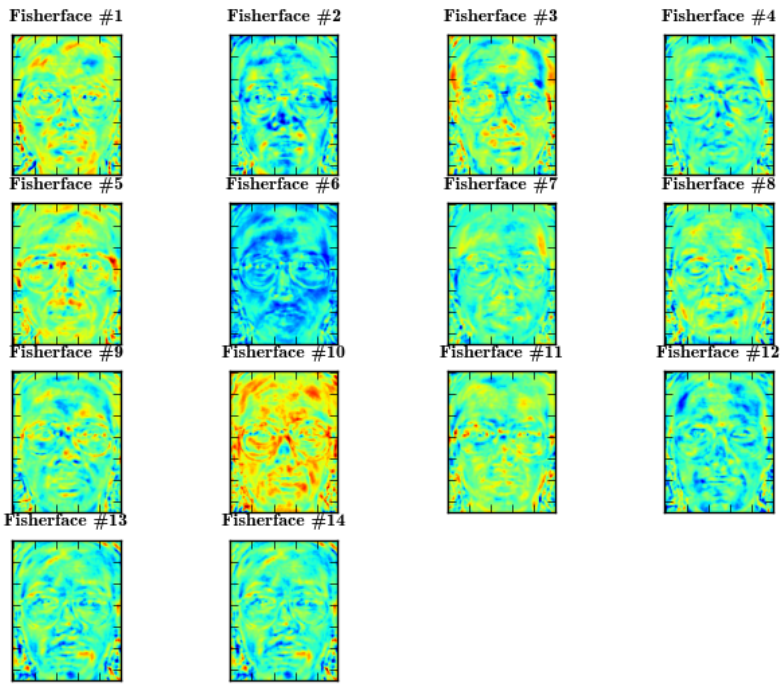

Figure 3.2: Fisherfaces representation of the Yale Facedatabase A

3.4 Local Binary Patterns Histograms

Local binary patterns (LBP) method is a texture operator that assigns a label to each pixel of an image by converting an image from greyscale to binary with a non-linear operation. The conversion is called **thresholding** meaning that if a pixel's value is lower then a specific value (threshold) the value 0 is assigned, 1 otherwise [21]. LBP uses a **neighbourhood thresholding** which means that a section of 3x3 pixels for each frame in the picture is taken in consideration when assigning a label. The result is then transformed into a binary number. Figure 3.3 shows the transformation process.

A more sofisticated usage of the neighbourhood section has been introduced later in [22] using a circular neighborhood and bilinearly interpolating values at non-integer pixel coordinates [23]. One the most important feature of LBP is its robustness to *monotonic gray-scale* changes due to the illumination variations of pictures.

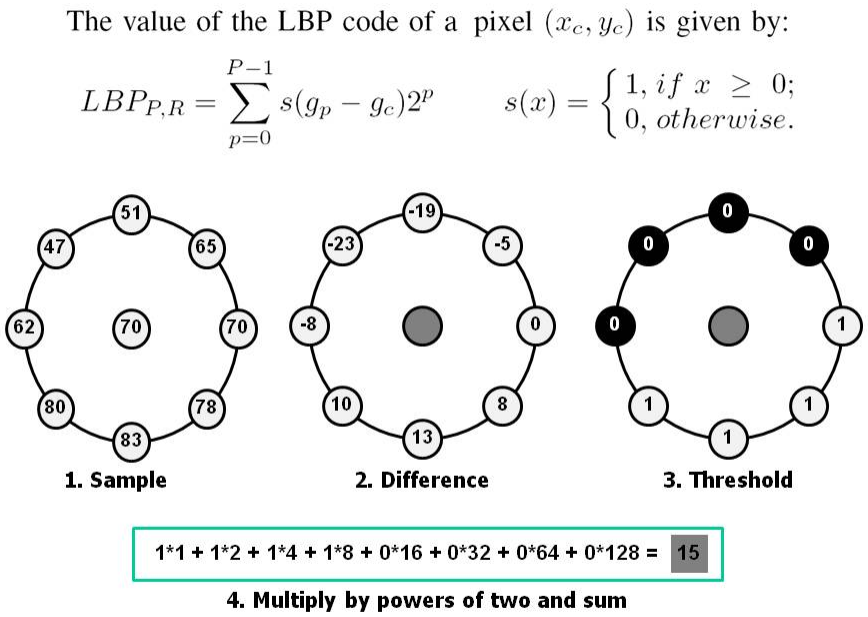

Figure 3.3: An example of LBP computation[23]

The LBP approach for classifying a texture is made by collecting the occurrences into a histogram. The classification itself is then done by computing the similarity among all the histograms. Although, for facial image recognition the approach loses spatial information and therefore the algorithm should also take the location of the texture into account. An effective method to accomplish this task is using the LBP texture *descriptors* to construct quite a variety of *local descriptor* of the face. After that these local descriptors can be combined in a unique descriptor called *global descriptor*. The usage of these local descriptors has been found to be more robust against the variation of poses of the face and illumination compared to holistic appreaches like eigenfaces and fisherfaces [24].

The basic methodology for Local Binary Patterns in face recognition was first introduced in [25] and the procedure is described as follows:

- The picture with the face is divided into local regions (neighbourhood approach)
- The LBP texture descriptors are then extracted from the each region
- The descriptors are then concatenated to build the global descriptor

Figure 3.4 shows the procedure described above. The resultant histogram contains a description of the face split on three levels of locality: the labels contain the information related to the patterns on fragment level, labels from a

small region are then summed up to produce information on a regional level and these regions are then combined to construct the global descriptor.

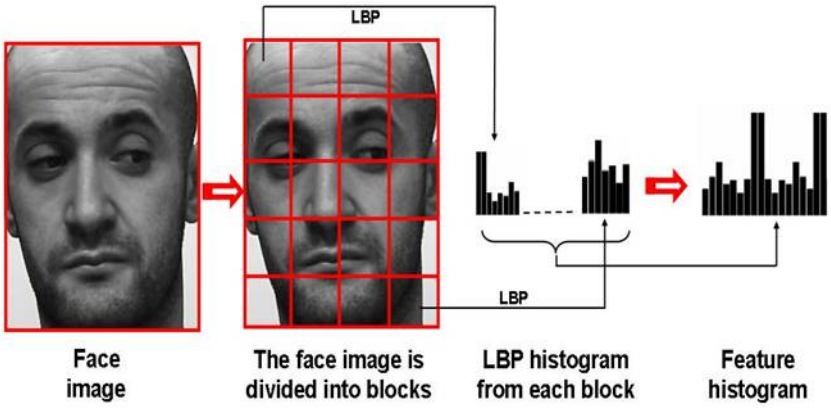

Figure 3.4: Face description with local binary patterns [23]

Chapter 4

Model Prediction

To help the user to auto-select a suitable mate, we introduced a *model prediction system* based on the taste of the user. In fact, during the *training phase*, the user selects which are the persons he/she would more likely go out with and which not. With this we create a dataset that we will use for our prediction system.

A simple definition of model prediction system is the ability of predicting the future through the usage of one or more classifiers. The classifier will try to determine the probability of a set of data belonging to another set. Applied to our case, when the training phase is completed, a testing phase will determine the *accuracy* of our model.

4.1 Classification models

Facerec provides two classifiers: *K-Nearest Neighbor* and *Support Vector Machines*. Despite the limited amount of solutions, these classifiers are both suitable for our project.

K-Nearest Neighbor is a **non-parametric** method. It means that it makes no assumptions about the probability distributions of the variables being assessed for either classification or regression. In both cases, the input consists of the *k closest training examples* in the *feature space*. In classification, the output of K-NN is a class membership where each object is classified using the *majority vote* of its neighbors. In the case of *k=1*, the object is simply assigned to the class of that single nearest neighbor. In our project we used two different methods for calculating the distances: **Euclidean** and **Chi-square**. The first one simply computes the distance between two points in the Euclidean Space. Basically, the distance is the *length* of the line segment that connects the two points.

The second method instead, is the distance between two *histograms* having the same length. Both histograms are then normalized within a range between 0 and 1. The distance measure is defined as follows:

$$\frac{\sum \frac{(x_i - y_i)^2}{(x_i + y_i)}}{2} \tag{4.1}$$

Support Vector Machine is a supervised learning model with an associated learning algorithm that analyzes data and tries to recognize patterns. It is mostly used for both classification and regression analysis [26]. Each example in the training dataset is *marked* as belonging to one of two different categories and the training algorithm will build a model that assign new samples to one category or the other one. This technique makes SVM a non-probabilistic binary linear classifier. The resultant model is basically a representation of the samples as points in the space. Those points are mapped in such a way that the examples the categories are splitted by a clear gap that is as large as possibible. When a new sample is introduced in the model, they are mapped into that same space and placed (or predicted) to a category based on which side of the gape they fall on [26].

Chapter 5

Test and Implementation

After gathering the data (see Chapter 2) we used a self-written program to compare the described algorithms. In the first step, the program read in data from the database for a single user and preprocesses the pictures using Haar Feature-based Cascade Classifier to detect the face, crop it and resize the result to 30 by 30 pixels. Afterwards the dataset is splitted in two parts, the first containing 60% of the rated people and the second one the remaining. The first set is used to train a model using a feature extraction and classifier combination. Then the set with the remaining persons is used to test the model on its pictures and compare the predicted value from the actual vote from the user.

For this test, two variants were implemented. In the first one, the pictures are rated individually and the prediction is compared with the vote of the user. In the second one, all three pictures of a person are individually predicted by the model and afterwards a majority decision is used to generate the like-value for the person and compared to the users input.

5.1 Implementation

To implement the software, we decided to use the Python programming language, because its ecosystem provides a wide variety of machine learning and face recognition libraries and is easy to use. Speed of execution was not a concern to us in such a research project.

We wrote a program, that executes the above described steps using different frameworks. The face recognition and cropping was implemented using the library OpenCV 2[27] and its Python bindings. It provieded basic classifiers as a decision-tree classifiers with at least 2 leaves, where Haar-like features are the input to the basic classifiers as described in Section 3.1. Figure 5.1 shows the features tipically used depending on the classifier. From the OpenCV documentation: *"The feature used in a particular classifier is specified by its shape (1a, 2b etc.), position within the region of interest and the scale."* [28].

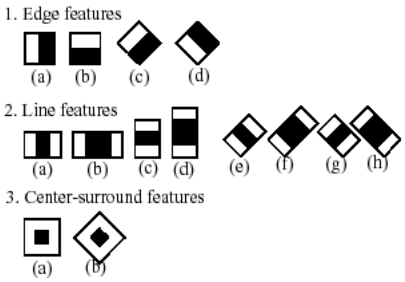

Figure 5.1: Features in Cascade Classifier [28]

For the implementation of the feature extraction and classification we used a library called "facerec"[29]. This provided support for Fisherfaces, Eigenfaces and Local Binary Pattern Histogram as Feature Extraction algorithms and k-Nearest Neighbours and Support Vector Machines as classifiers.

We scripted the program in such a way, that it automatically trains different combinations of the above and provide csv-tables and graphs for the results of the evaluation.

5.2 Evaluation of the results

In figure 5.2 the average of the accuracy over the different users is shown against the different feature extraction algorithms and classifiers. Additionally it differentiates between majority decision (see above) and decision on pictures alone, marked with "_vote" and "_nonvote".

As shown in the picture, we got the best combination for all users using PCA (Eigenfaces) as a feature extraction and 5-nearest neighbours with euclidean distance as a classifier.

Assuming random guessing of the algorithm would create an accuracy of 50%, the graphs show that at least some combinations are significantly better than random, even having the small dataset we tested on in mind.

Shown in the next figures 5.3, 5.4, 5.6 and 5.5, the accuracy of a certain feature is drawn for different classifiers for some selected users. We assume that these users are representative of the range of errors we got. The figures clearly state out that the accuracy is highly dependent on the user. In some cases, like for the user "BabyMonkey", the results of the prediction are even worse then randomly guessing if we assume a rate of 50%. In other cases, like for user "mylovelyml", the accuracy is up to over 90% and shows that the algorithm is capable to select the matching pictures with relatively high accuracy.

The graphs state further, that Fisherface is the worst feature extraction algorithms of the competitors and 5-Nearest Neighbours with Euclidean distance has in general quite high accuracy compared to the competitors.

Nevertheless, we have problems explaining the special behaviour of Fisherface for user "GoTobias" and "mylovelyml".

5.3 Guessing and Significance

To interpret whether the algorithms work successfully, we have to assume a certain probability for successful random guessing to which we can compare our results.

Assuming the user liked a person with a possibility of 50%, we should statistically be able to get an accuracy of 50% using random guessing. The same holds if a person just likes 10% and dislikes 90% of the people. Nevertheless, in the last case it would be possible to just dislike all persons and get an accuracy of 90% which would be on the line with our best results.

Concerning that, it is not quite possible for us to state whether the algorithms are better than *educated guessing* or not.

As shown in figure 5.7 the testcases create false positive as well as false negative results. The graphs for other combinations of feature extraction and distance functions are comparable. Because of this behaviour, we assume that 50% as an error rate for random guessing can be used and the provided graphs show, that at least some combinations are working significantly better then guessing.

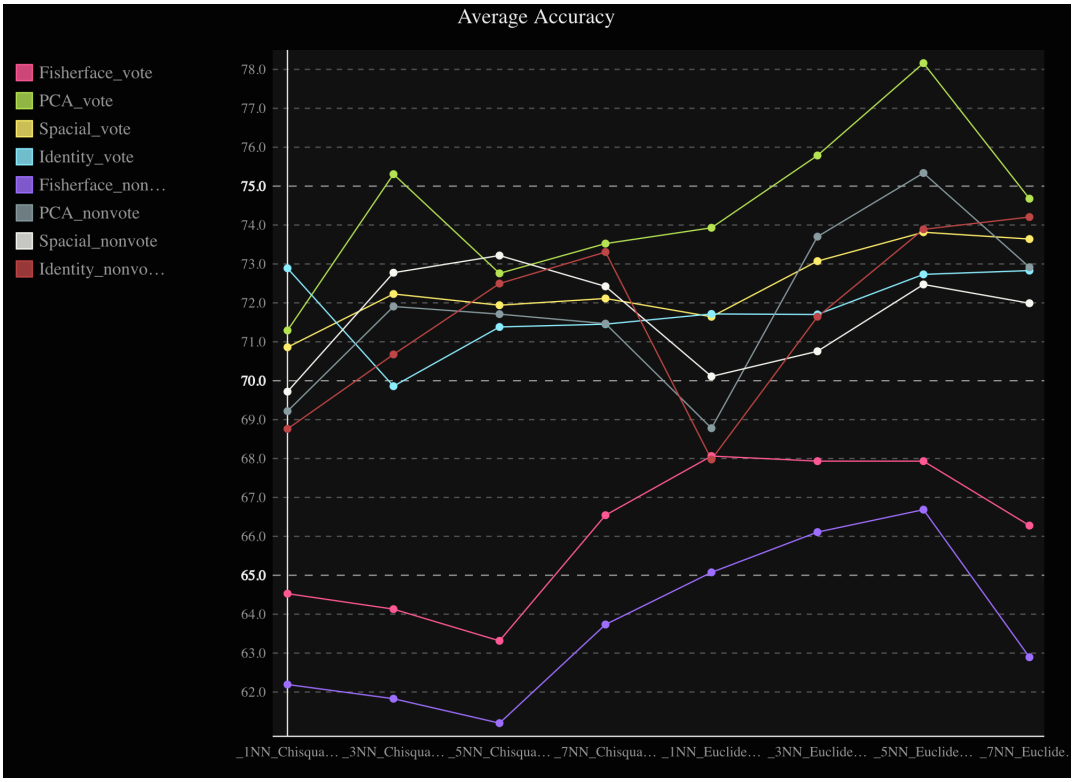

Figure 5.2: Average Accuracy

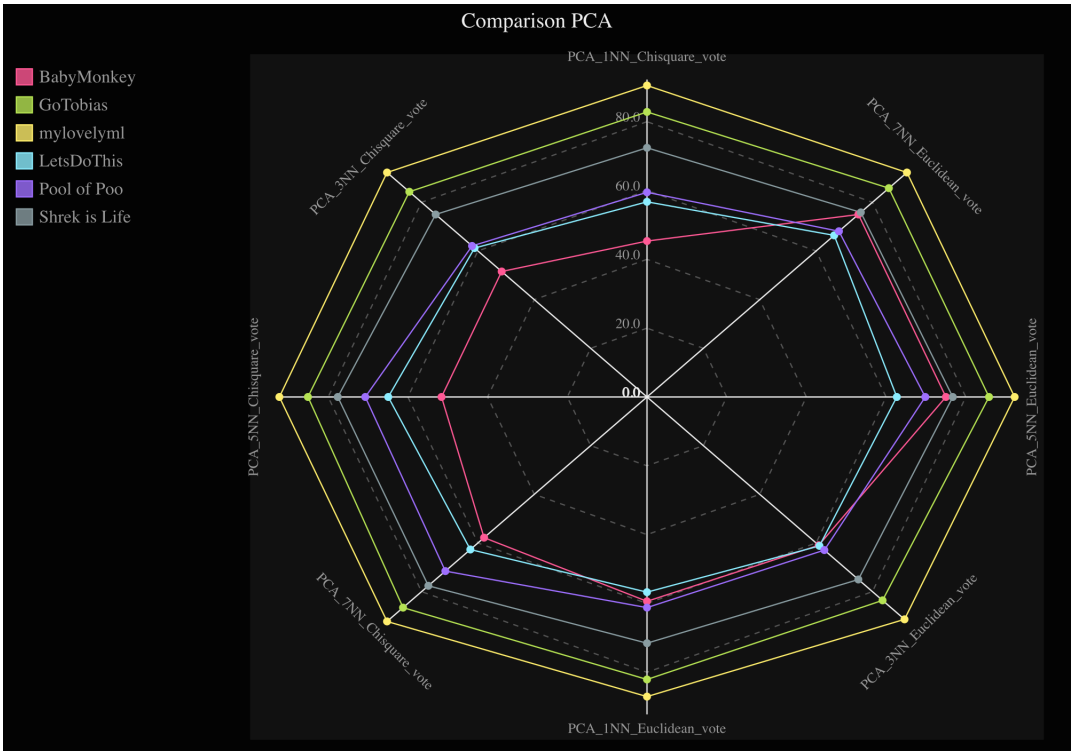

Figure 5.3: Eigenfaces accuracy

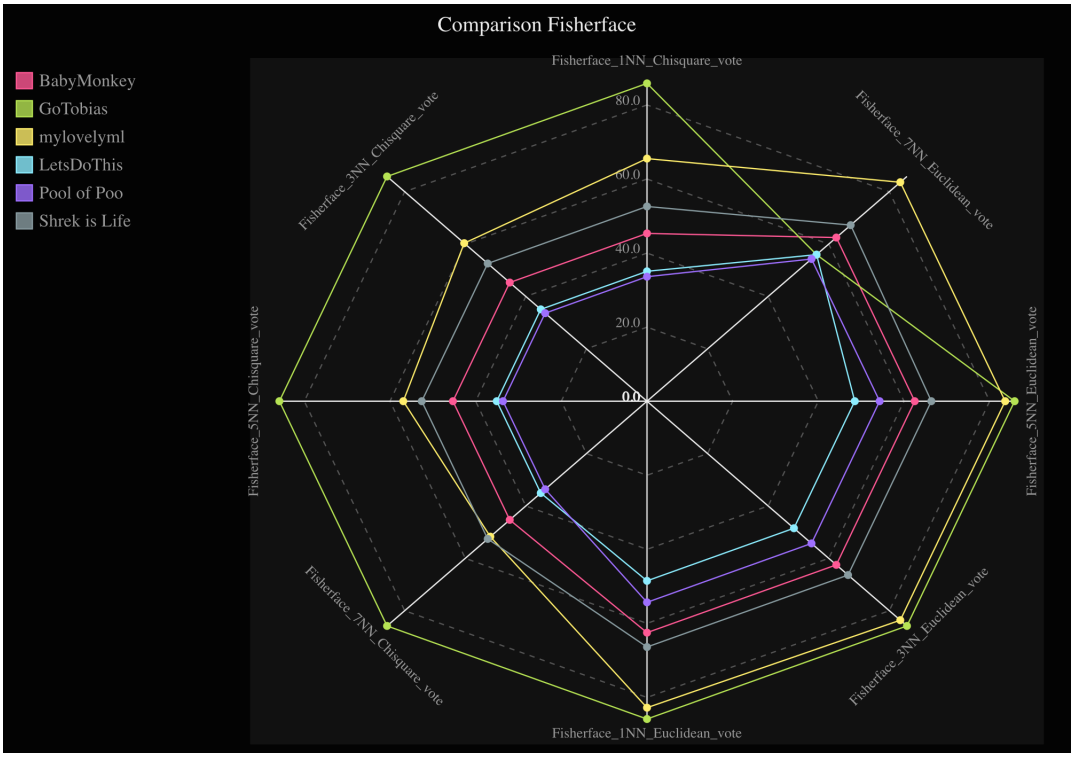

Figure 5.4: Fisherfaces accuracy

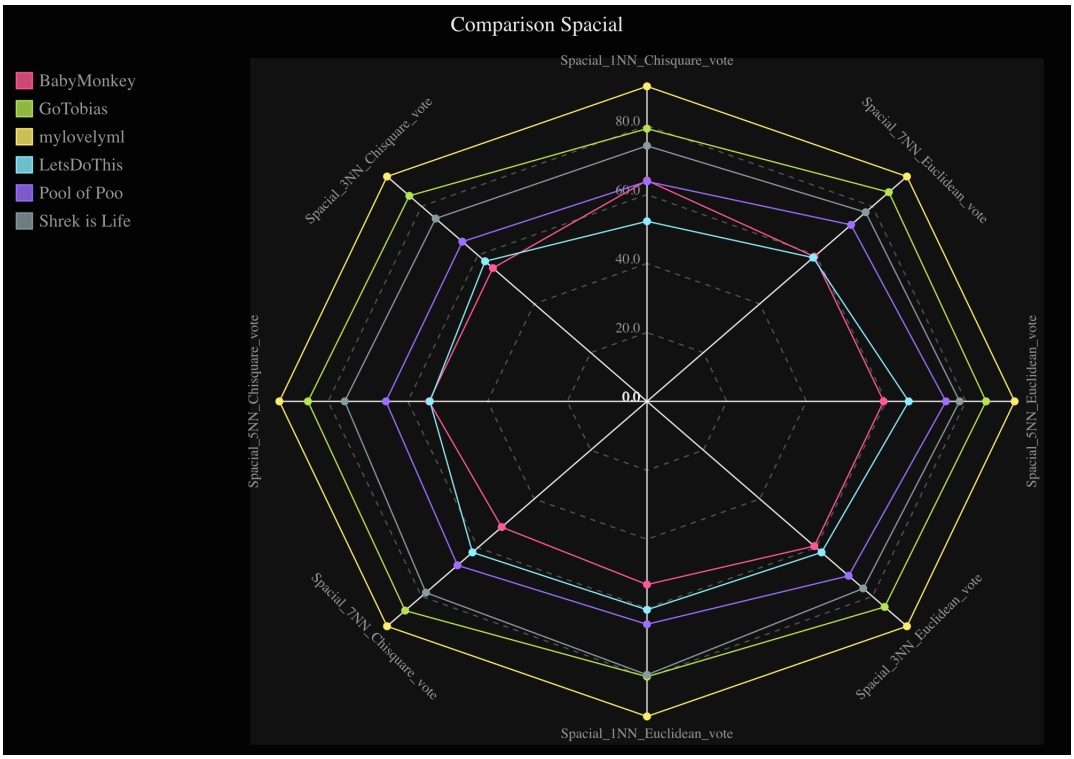

Figure 5.5: Local Binary Pattern accuracy

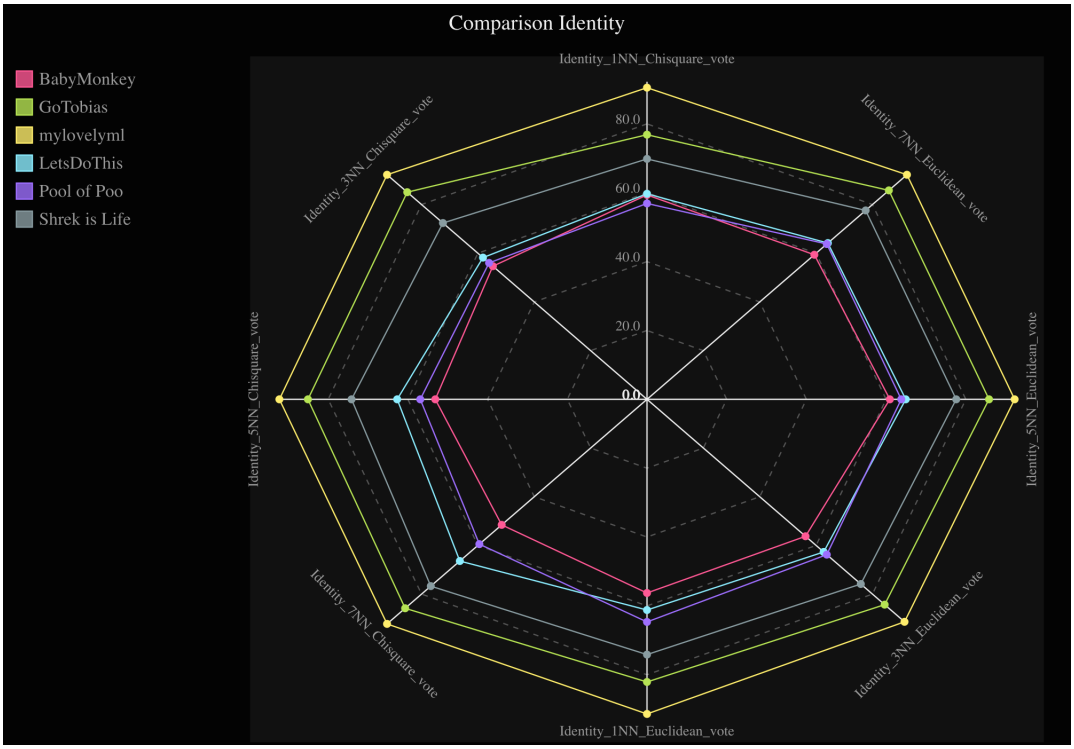

Figure 5.6: Identity accuracy

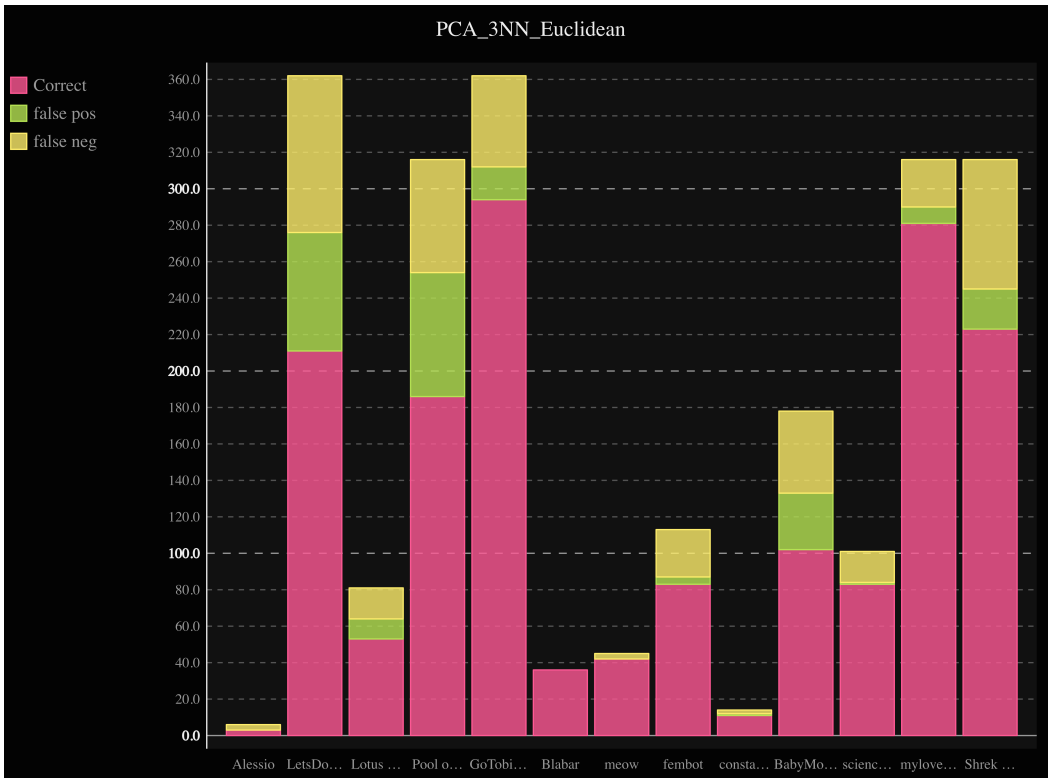

Figure 5.7: accuracy PCA 3NN Euclidean

5.3.1 Conclusion

We know from some persons of the test group, that they had quite some problems to just rate the face that was acutally used by the algorithm and not the person as a whole, including clothes and appearance. Furthermore, by looking at the cropped images, we found some pictures, where the wrong person was cropped out or the picture was slightly turned.

Concerning these drawbacks in preprocessing and data gathering, we assume, that the data at least for some of the users are quite promising and a basis for further research.

5.4 Evaluation of the project

Overall we think we made a good progress during the project and have some results, which can be used for further investigation.

Nevertheless we had some struggles during the project concerning different versions of Python, support of libraries and the libraries themselves. The part of gathering the data took a little bit longer then expected and left us less time for optimisation. Additionally we wanted to also evaluate the use of Support Vector Machines as a classifier and tested the algorithm with different parameters, the results we got back from the library were invalid for all parameters and we were not yet able to fully investigate the error. Because the algorithm is not just implemented in the library itself but uses also a the bigger libsvm library this might be an encounter which takes us some time, we don't have anymore.

We liked the project and think we learned the basics of how face recognition works in general and how the differnt layers including the machine learning part work together. We see a lot potential for further improvement we descibed in the next chapter.

Chapter 6

Related and future works

This project explores just a small part of the available possibilities. The face recognition algorithms we used were easy to use from the programming point of view as well as well tested. Although, we are aware of the fact that nowadays, more accurate and sophisticated methods are available. For instance, Facebook uses *Deep Learning* system to create a 3D representation of a face given a 2D image [30], but this goes beyond our knowledge and from a project point of view, this technique would be too to complicated to implement.

As mentioned in the Introduction, *Tinderbox* is a program that autoselect people on Tinder using face recognition. The application exploits Eigenfaces as features extraction and for each person it detects the face from every picture and computes the mean. After the user has trained a model with ca60 people, the program starts selecting people automatically [1].

Different improvements can be applied to this project. One of those could be the use of Support Vector Machine as classification method. Despite our efforts to use this algorithm, we were not able to get the best out of it because of lack of time. The number of parameter that can be tuned and kernel functions in SVM is so large that it could be a potential project itself. Another enhancement could be the use of different cropping sizes for each image. This could potentially improve the features extraction algorithms because they would have a broader area to work on.

As said above, K-means is still in beta and we do not know when it will be released. Although, it could be a potential feature to use in the future.

Bibliography

[1] "Automatic tinder with eigenfaces," 2015. accessed 2015-05-08. Available: http://crockpotveggies.com/2015/02/09/automating-tinder-with-eigenfaces.html.

[2] "Flask framework," 2015. accessed 2015-05-08. Available: http://flask.pocoo.org.

[3] "Sqlalchemy - the database toolkit for python," 2015. accessed 2015-05-08. Available: http://www.sqlalchemy.org.

[4] "Sqlite," 2015. accessed 2015-05-08. Available: https://www.sqlite.org.

[5] "Wtforms - a flexible forms validation and rendering library for python," 2015. accessed 2015-05-08. Available: https://github.com/wtforms/wtforms.

[6] "Tinder api documentation," 2015. accessed 2015-05-08. Available: https://gist.github.com/rtt/10403467.

[7] "Face detection definition," 2015. accessed 2015-05-08. Available: http://en.wikipedia.org/wiki/Face_detection.

[8] "Face recognition definition," 2015. accessed 2015-05-08. Available: http://en.wikipedia.org/wiki/Facial_recognition_system.

[9] M. Oren, C. Papageorgiou, P. Sinha, E. Osuna, and T. Poggio, "Pedestrian detection using wavelet templates," in *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pp. 193–199, IEEE, 1997.

[10] C. Papageorgiou and T. Poggio, "A trainable system for object detection," *International Journal of Computer Vision*, vol. 38, no. 1, pp. 15–33, 2000.

[11] "Haar-like features and integral image representation," 2015. accessed 2015-05-08. Available: http://www.cvc.uab.es/adas/site/userfiles/files/haarfeatures.pdf.

[12] T. Tuytelaars and K. Mikolajczyk, "Local invariant feature detectors: a survey," *Foundations and Trends® in Computer Graphics and Vision*, vol. 3, no. 3, pp. 177–280, 2008.

[13] S. Reinius, "Object recognition using the opencv haar cascade-classifier on the ios platform," 2013.

[14] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1, pp. I–511, IEEE, 2001.

[15] R. Lienhart and J. Maydt, "An extended set of haar-like features for rapid object detection," in *Image Processing. 2002. Proceedings. 2002 International Conference on*, vol. 1, pp. I–900, IEEE, 2002.

[16] B. Heisele, P. Ho, and T. Poggio, "Face recognition with support vector machines: Global versus component-based approach," in *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, vol. 2, pp. 688–694, IEEE, 2001.

[17] R. Tripathy and R. Daschoudhury, "Real-time face detection and tracking using haar classifier on soc,"

[18] M. A. Turk and A. P. Pentland, "Face recognition using eigenfaces," in *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference on*, pp. 586–591, IEEE, 1991.

[19] "Fisherfaces definition," 2015. accessed 2015-05-08. Available: http://www.scholarpedia.org/article/Fisherfaces.

[20] P. N. Belhumeur, J. P. Hespanha, and D. Kriegman, "Eigenfaces vs. fisherfaces: Recognition using class specific linear projection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 19, no. 7, pp. 711–720, 1997.

[21] "Image thresholding and segmentation," 2015. accessed 2015-05-08. Available: http://www.bogotobogo.com/python/OpenCV_Python/python_opencv3_Image_Global_Thresholding_Adaptive_Thresholding_Otsus_Binarization_Segmentations.php.

[22] T. Ojala, M. Pietikainen, and T. Maenpaa, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, no. 7, pp. 971–987, 2002.

[23] "Local binary patterns," 2015. accessed 2015-05-08. Available: http://www.scholarpedia.org/article/Local_Binary_Patterns.

[24] D. N. Parmar and B. B. Mehta, "Face recognition methods & applications," *arXiv preprint arXiv:1403.0485*, 2014.

[25] T. Ahonen, A. Hadid, and M. Pietikainen, "Face description with local binary patterns: Application to face recognition," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 28, no. 12, pp. 2037–2041, 2006.

[26] "Support vector machine definition," 2015. accessed 2015-05-08. Available: http://en.wikipedia.org/wiki/Support_vector_machine.

[27] "Opencv - open source computer vision," 2015. accessed 2015-05-08. Available: http://opencv.org.

[28] "Cascade classification in opencv," 2015. accessed 2015-05-08. Available: http://docs.opencv.org/modules/objdetect/doc/cascade_classification.html.

[29] "facerec library," 2015. accessed 2015-05-08. Available: https://github.com/bytefish/facerec.

[30] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pp. 1701–1708, IEEE, 2014.