TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Kristina Õim 163445IAPM

LEARNING AND RECOGNITION OF FACIAL EXPRESSION WITH DECISION TREES

Master's thesis

Supervisor: Jüri Vain
PhD
Co-supervisor: Sven Nõmm
PhD

Tallinn 2018

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Kristina Õim 163445IAPM

NÄOILMETE ÕPPIMINE JA TUVASTAMINE OTSUSTUSPUUDE ABIL

Magistritöö

Juhendaja: Jüri Vain
PhD
Kaasjuhendaja: Sven Nõmm
PhD

Tallinn 2018

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Kristina Õim

07.05.2018

Abstract

Recognizing facial emotions has a wide area of application domains such as security, marketing, health monitoring. It would be the next challenge in research after face recognition. Goal of this thesis is to study and apply emotion recognition and analyse how good of a result it gives using decision trees - a tool popular in machine learning. All the data is gathered and preprocessed manually. Emotion recognition and features calculated from data are based on the output of 3D Kinect feature HD Face tracking. Selected features were put in decision tree classifier and the results were evaluated. Five features give predictability of 89%. The use of decision tree gave good results in predicting three different emotions. Results are trustworthy and were expected according to the similar experiments found in research literature.

This thesis is written in English and is 41 pages long, including 6 chapters and 13 figures.

Annotatsioon

Näoilmete õppimine ja tuvastamine otsustuspuude abil

Näo ja näoilmete tuvastusel on palju võimalikke rakendusalasid nagu näiteks turbekontroll, sotsiaalvõrgustikud, tervise jälgimine. Näoemotsioonide tuvastamine on näotuvastuse uuringutele järgmine loogiline samm. Käesolev lõputöö eesmärk on uurida ja rakendada otsustuspuudel põhinevaid emotsioonidega seotud näoilmete õppimis- ja tuvastusalgoritme, samuti hinnata nende algoritmide sobivust antud ülesande lahendamiseks. Lähteandmed näoilmetest on kogutud ja eeltöödeldud autori enda poolt käsitsi. Emotsioonide tuvastuseks vajalikud tunnuste alusandmetena on kasutatud 3D Kinect seadme näotuvastuse väljundandmeid, mis on esitatud näokujutise punktipilve ja punktide vaheliste kauguste kaudu. Näoilmete andmestiku analüüsi tulemusena saadud tunnuste alusel konstrueeriti otsustuspuu ja hinnati kontrollandmete põhjal selle tuvastusprotsenti. Mudel viie tunnusega tuvastab kolme näoilmet täpsusega 89%. Tulemus on usaldusväärne ja vastab teaduskirjanduses esitatud analoogsete katsete põhjal ootustele. Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 41 leheküljel, 6 peatükki 13 joonist.

List of abbreviations and terms

DPI	*Dots per inch*
TUT	Tallinn University of Technology
CNN	Convolutional Neural Network
FERA	Facial Expression Recognition and Analysis
AU	Action Unit
FACS	Facial Action Coding System
GEMEP	The Geneva Multimodal Emotion Portrayals
LBP	Local Binary Pattern
SVM	Support Vector Machines
SDK	Software Development Kit
USB	Universal Serial Bus
CSV	Comma Separated Values
JSON	JavaScript Object Notation
RGB	Red Green Blue
DTW	Dynamic Time Warping

Table of contents

List of figures

1 Introduction

In today's modern technology, companies are competing to create more attractive intelligent products and home entertainment systems. There are hands free systems that react to spoken commands, they understand speech and what is asked by them. Such amenities as calendar reminder, traffic supervision, music and smart home control are just a few to point out [1] [2]. Next step of human adaptation for being closer to user comfort is facial recognition and facial emotion recognition. Face recognition is already implemented in our social networks and other products. Emotion recognition is used in advertising based on the target audience response to a video [3]. Recognizing human emotions has a wide area of application domains such as customer-attentive marketing, health monitoring, and emotionally intelligent robotic interfaces, and security. Institute of Software Science, TUT is collaborating with Estonian Police and Border Guard Board on the development of face recognition and identification software. This work is an extension to referred project where the technical tools developed will be applied also in this thesis for collecting necessary training data and for arrangement of facial expression recognition experiments.

The goal of this thesis is to study, apply and tune facial expression learning and recognition methods for improving the face recognition algorithms applied in high traffic establishments, i.e. border crossing and passport checks. It would be an extra feature to pay more attention to passengers.

Thesis will consist of the following tasks:

- First of all extraction of facial features from the pointcloud of ~1350 3D-points from Kinect [4].
- Then building a classifier from distance metrics which are calculated from those exact points.

- The training set for the algorithm should consist about 100 images with 3 facial expressions: happy, sad, neutral. This thesis will concentrate on these 3 emotions from total of 7 which are universal for different cultures and races.
- Emotion learning should be conducted using Decision Trees [5]. They are helpful for deciding on different problems. The model is a tree like structure (also known as decision graph) with nodes and branches. The decision trees are widely used for analysing decisions but are also used in machine learning [5].
- At last, the approach must be validated using a sample set including 20 instances.

This thesis includes 6 chapters on 41 pages and 13 figures.

2 Methodology

The methods for solving the tasks stated in the Introduction are chosen by following the two principles:

1. The methods must be relevant for addressing the problem with satisfying precision and certainty.

2. The methods selected must not be too complex to be implemented and used under the time frame of preparing the thesis.

2.1 Distinguishability of Emotion

Sample data will be manually gathered with Kinect 3D [6]. It will be used to scan faces and different emotions. Three emotions will be scanned from one subject. All emotions will be saved separately referring to their emotion. Kinect gives out 1347 facepoints from where the 35 main ones are named for better usability [7]. All the points are used to find the best features to work with. The three selected emotions happy, sad and neutral are to be distinguished from a sample size of 20 or more.

2.2 Fisher score

Fisher's scoring is a form of Newton's method used in statistics to solve maximum likelihood equations numerically, named after Ronald Fisher [8] For Fisher score the author will use all Kinect [7] output points. The goal here is to measure distances between points and between different emotions, to choose the most useful ones for building the classifier. Some features (certain points or distances) may give out more information than others, because of that it is wise to use Fisher score. Fisher score is used to calculate the importance of features [9]. This helps to determine the usefulness of feature and whether to include the feature in the classifier. The score will be calculated for each feature separately but considering all three emotions. Author has developed her own program (see Subsection 4.4 for details) to calculate the Fisher score.

2.3 Features and feature engineering

For building a good classifier it is expected that the features used for classification have high Fisher score. The features of facial expression classifier are the distances between different facepoints and contextual information about the emotions. Feature engineering consists of calculating extra angles or curves on the set of facepoints i.e. Bezier curves [10] which could be used in the classifier as well. There is also a chance to use both, the features that Kinect [7] gives out directly and additional features that are computed by feature engineering methods for developing extra input to classifier.

2.4 Classifier and Implementation in Python

Data pre-processing, feature calculation, classifier and implementation will all be written using Python programming language. The choice is purely made on the authors preference. The model of facial expression will be constructed using decision tree. Last step will be training the model and determining how accurate the emotion recognition is. It is expected that the model can separate happy emotion from a neutral one easily. Work is finished with results analysis and with identifying the usability boundaries for the method at hand.

3 Preliminaries

Machine learning and especially facial recognition including facial expression recognition has been a topic of intensive research for some time now. Machine learning has considerable advantages in a lot of businesses i.e. marketing, estimating the user preferences or feelings towards a product or just simplifying everyday tasks and life in general. This is what every industry is aiming towards for. Large number of publications in the area suggesting different methods has been referred in [11]. Emotion recognition is possible from images, speech [12] and videos [3]. Massive amount of methods is accessible on Ibug [13] courtesy of Department of Computing, Imperial College London.

3.1 Ibug

Ibug is short for Intelligent Behaviour Understanding Group which analyses human behaviour including face analysis, audio analysis and biometrics analysis [13]. There are many publications on facial expression recognition where the features are extracted on the baseis of Fischer score. Though, the tasks addressed in the thesis are carried out with different methods [14] [15]. The overview in [11] is perfect for understanding what has been achieved in this field and what are the different results with different methods. Many publications are based on recognising emotions from images, using Bezier curve and they are working with huge datasets [16]. The accuracy for different methods is benchmarked in numbers using one of the well-known facial expression databases [17] [18] [19].

3.2 Face API

Face API was formerly included into Microsoft Project Oxford [20] but it has a separate functionality now. There are few options to test methods on their webpage. Face verification analyses provides the possibility of having the same person on two different images while API returns its decision with percentage how confident the system is in the output. Face detection searches human faces on images, and there is no limit how any faces could be oriented. In addition to detecting a face or multiple faces, the API also predicts the features from the image including emotion, age, position in image, gender,

smile and glasses [21]. Face verification and face detection are both publicly available for testing without any need of downloading the application into a local repository. Though, the option to search for similar faces and grouping faces is only possible in your local repository.

3.3 Public challenges

EmotiW challenge or Facial Expressions in the Wild challenge is a competition held every year since 2013. The challenge is to estimate group emotion from photos uploaded to social media [22]. There is also an audio-video based emotion recognition challenge that is continuation from previously held EmotiWs. Analysis takes place in the real world with the presence of "noise" like head movement, etc. EmotiW competitions have been held for a few years by now, each one in different country. 2017 winners used two types of Convolutional Neural Networks (CNN) to validate their approach. Combining different algorithms gave them accuracy overall of 80.9% [23].

9th IEEE International Conference on Face and Gesture Recognition was held in 2011. The Facial Expression Recognition and Analysis (FERA) challenge consisted of 2 sub-challenges: action unit (AU) detection and discrete emotion detection [15]. AU-s are muscle movements that are adequate to a facial emotion, e.g. jaw drop [24]. They belong to a Facial Action Coding System (FACS). At the moment using FACS is the only solution to estimate the emotion in real life situation [24]. At the challenge they used the GEMEP database of audio-visual recordings [25]. The goal for the first sub-challenge was to detect the presence of AU in every frame of the video material. For the second sub-challenge to detect which emotion out of five (anger, fear, joy, relief, sadness) was present. Local Binary Pattern, Support Vector Machines were used to detect the changes in AU and emotion detection. It was concluded that the task was difficult but not entirely impossible [15].

4 Facial expression classifier

Machine learning starts from choosing the algorithm for classifier which works the best with the given parameters and ultimately gives the best result in the end. There is also an option to choose multiple algorithms and select the best amongst them via cross-validation [26].

4.1 Usability

Since facial recognition is gaining popularity it has been tested on images, video, group events, etc. The scale of opportunities and usability is wide starting from user friendly services to security checks on boarder corssing. For instance, when client goes to the phone operator office, then the client's background regarding their services and client complains is immediately known. As referred above, the facial emotion recognition could have an immense impact also at border crossing and in airports when detecting suspicious travellers. Of course, there are also more specific applications like grouping photos according to emotions of people on photos.

4.2 Technical requirements

Specific technical requirements to facial expression recognition depend on the area of application. A list of functional and non-functional requirements needs to be specified before any actual coding starts. A correct requirements specification saves a lot of time in the long run if everything is clear and decided before the actual work begins. When some aspect of the task changes the adjustments in the presence of modluar and unambiguous specification can be made without extensive redesign and implementation effort.

Regarding the requirements to facial expresson classifier, the program should find facepoints, at first, that move the most between emotions using output from Kinect [4]. Then the program should calculate Fisher score and construct Bezier curves from mouth points indicating a smile. Next step is finding the points that move the most due to the

emotion under study and by calculation their Fischer score the best of them can be extracted and used as input for the classiffier.

Predicting an emotion. Humans can detect already slight changes in face very accurately. Whether it is a jaw drop or brow lowerer the person opposite detects the emotion. These slight muscle movements are classified as micro expressions [27]. Computer, on the other hand, needs formal rules to classify an emotion. Happiness (Figure 1) can be detected from pushed up cheeks and rising lip corners but also many other facepoints are observed.

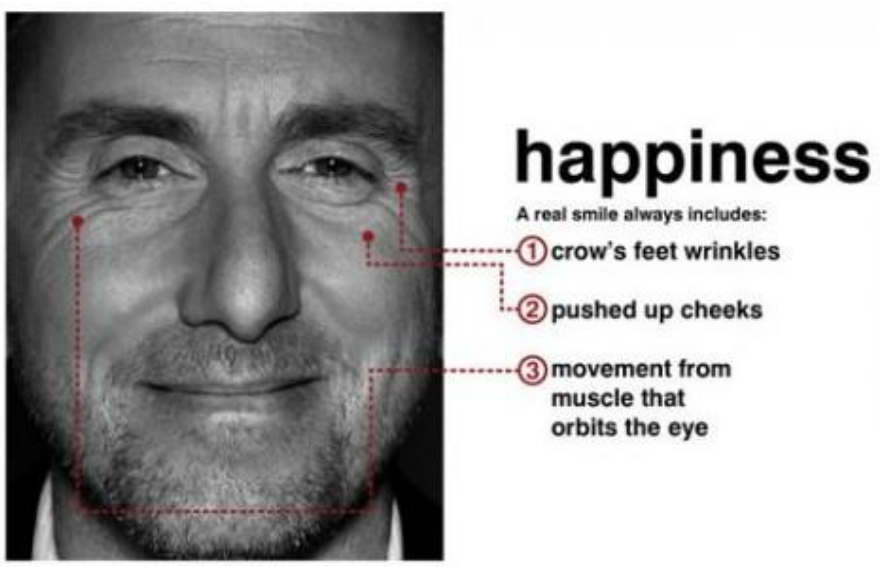

Figure 1. Happiness example [28].

Sadness, on contrary, pulls lip corners down and people are sad with their eyebrows and eyelids slightly dropped (Figure 2).

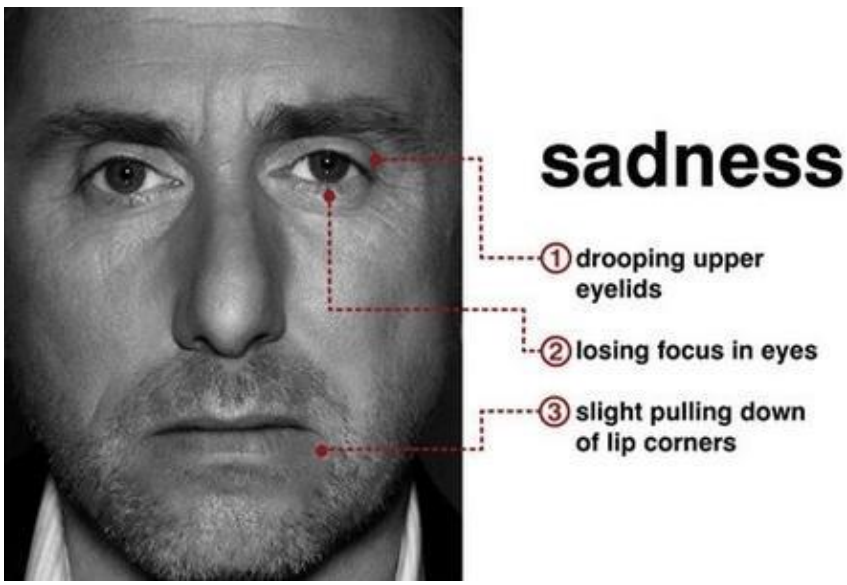

Figure 2. Sadness example [29].

4.3 Kinect and data collection

The image data of facial expressions are recorded by Microsoft Kinect. It is a motion sensing device which allows the user to communicate using motions and speech commands [4] (Figure 3). It consists of RGB camera, infrared emitter, infrared depth sensor, multi-array microphone and tilt motor. It is possible to capture a colour image and a depth image [6]. Kinect is moderately accurate when scanning movement, it is used to play games without holding on to the remote. Device detects the movements of objects monitored and transfers them to the screen image.

Figure 3. Kinect machine in TUT.

For capturing and collecting the images some software needs to be installed. It is necessary to download Microsoft Visual Studio and from there the Kinect for Windows SDK. After that Kinect can be plugged in using USB port. Most of the theis work with Kinect was conducted in the Software science department laboratory where experimental data were collected (Figure 4).

All the data used in this work have been gathered and preprocessed using Kinect that has been mounted in the laboratory. The images were captured from a group of people who volunteered for face scanning. Test subjects were male and female Caucasian people of age 9 to 63. For each person the data were gathered in the same conditions. In a fully lighted room, Kinect placed approximately a meter away from the subject. Each subject has been captured with 3 different emotions: happy, sad, neutral.

Figure 4. Laboratory.

Kinect saves the data as series of values to an Excel file (Figure 6). Values represent the 1347 points that Kinect records. They are represented as 3 dimensional vectors. The points recorded always refer to the same point in the face and the more used 35 of them are named accordingly [7]. Face that appears on the screen (Figure 5) is always the same but the data behind every face changes.

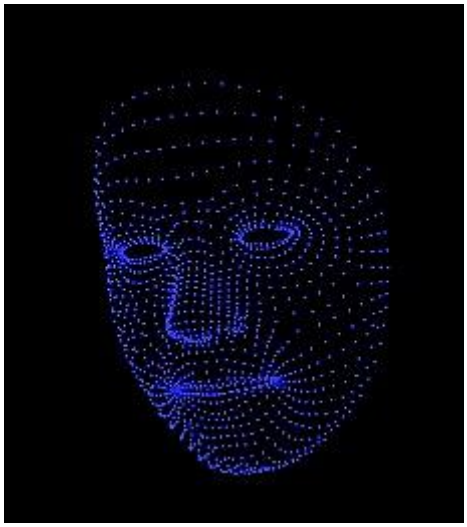

Figure 5. Kinect Face Tracking output.

The idea is to scan subject's facial expressions of three different emotions, analyse them and determine which characteristics to use for the model. That means using all of the 1347 points from Kinect [6], filtering out the most useful points and extracting most important

features. They will be used to train the algorithm to distinguish sad from happy and happy from neutral.

The program that records the facepoints has been written by a fellow student. The recorded data is saved in CSV (Comma Separated Values) file format. Since Kinect calibration stabilizing takes takes some time at the start of the recording author did not use the data of first nine lines to avoid getting false data at the begging of the datafile. Before applying the algorithm the data needs to be organized. Excel should save Kinect output data as text (Figure 6). One should follow that the data cathegory is determined correctly before saving, e.g. if the numbers are like a calendar date, then Excel changes the values to date format. Since the datafiles are huge and there are many of them the setup should be made at the very beginning of data processing.

	A	B	C	D	E	F	G	H	I
1	0			1			2		
2	X	Y	Z	X	Y	Z	X	Y	Z
3	0.1788462	0.03569789	1.017122	0.176989	0.03676163	1.010242	0.1727294	0.03963186	0.9940195
4	0.1790285	0.03542202	1.016914	0.1772699	0.03654441	1.010041	0.1731848	0.03951194	0.99384
5	0.1790113	0.03514466	1.016565	0.1772694	0.03627807	1.009717	0.173201	0.03921364	0.9935613
6	0.1810537	0.03555641	1.025462	0.179552	0.0364445	1.018486	0.1760083	0.03893591	1.002039
7	0.1814251	0.03615557	1.026487	0.1798851	0.03698069	1.019513	0.1762411	0.03932701	1.003065
8	0.1816589	0.03612522	1.025428	0.1800836	0.03702195	1.018468	0.1763736	0.03953685	1.002056
9	0.1813165	0.0367713	1.02645	0.1798171	0.03755672	1.019463	0.1762629	0.03980841	1.002985
10	0.1819082	0.03660354	1.026543	0.1803425	0.03737577	1.019566	0.1766565	0.03959596	1.003107

Figure 6. Subjects one emotion datafile example.

4.4 The algorithm of extracting facial expression features

In the first phase the program analyses all faces with 3 different emotions. That means calculating all point distances from zeroPoint which in this case is one of the points named by Kinect, i.e. NoseTop point number 24. For calculation of distances between points Eucleidean distance [30] is used. Calculated distances are grouped in the file based on the emotions (Figure 7).

```
current_point = 0

for all_points in datafile:
      point = datafile(current_point)
      distance = euclidean(NoseTop, point)
      add array(distance, emotion)
      if file ends:
        stop
      current_point += 1
```

Figure 7. All point distances calculation.

Current_point in the example stands for one point in datafile and it starts from zero. It is used to calculate the distance from NoseTop. The point value increases with each cycle. This loops for all the datafiles and all 1347 points.

Next step is to take all the neutral emotion distances calculated above and subtrackt them from “happy face” distances. The same point distance is taken from both neutralFace and happyFace distances. Output from them is absolute value that indicates how much points move when emotin changes, in this case it shows the distance of point locations between neutral and happy emotion (Figure 8).

```
position = -1

for i in neutralFace:
  for j in happyFace:
    if i == j:
      position = position + 1
      distance = abs(neutralFace[i]-happyFace[j])

      add array(distance, position)
```

Figure 8. Calculation for how much points move between emotions.

This gives a good dataset to start looking for the points useful for detecting different emotions. All the distances and their point references (positions) are stored in a separate column and then sorted in ascending order by distance length. Then it is possible to get the positions of the facepoints which moved the most between emotions. Position list is reversed for extracting the most influential features, the first 100 positions are selected for further analysis (Figure 9) and saved into positions.json file. The same has been done also with “sad face” distances.

```
positions = []
distance = 0
distanceListForHappyFace = sorted(distance)

for distance in distanceListForHappyFace:
  distance += 1
  if distance >= distanceListForHappyFace:
    stop
  add positions(distanceListForHappyFace.position)


positionsReversed = []
for position in reversed(positions):
  add positionsReversed(position)

firstHundred = positionsReversed[:100]
write_to_file(positions, firstHundred)
```

Figure 9. First 100 points selection.

When positions are selected they have to be converted to right point references. The following method is applied in the begging of the calculation of Fisher score. Points are read in from positions.json file where they were previously saved into. Because all the distances are saved in the same list their position enumeration starts from the multiple of 1346, i.e. this is how many facepoints are in use for each datafile. When the program takes a new face, the numbers start again from 0. That means, the original point reference of the face point can be restored by dividing the point’s serial number in the file by greatest multiple of 1346 such that the reminder remains nonnegative (Figure 10). The reminder is then the original reference. They are returned to Fisher score algorithm.

```
def calculate_points_from_positions(points):
  new_points = []
  for point in points:
    if point > 1346:
      multiplier = point // 1346
      score = point - (multiplier * 1346)
      new_points.append(score)
    else:
      score = point
      new_points.append(score)
  return new_points
```

Figure 10. Point numbers from positions.

After that, Fisher score can be calculated for each of the 100 points. Fisher score calculation program implementation is made by the author of this thesis. Importance of Fisher score is explained in chapter 2.2. Algorithm was constucted by mathematical formula (1) in "Data Mining" written by Charu C. Aggarwal. In Formula (1) μ_j and σ_j stand for mean [31] and standart deviation [32] of datapoints in *j-th* class in a feature, p_j represents fraction of data points in *j-th* class and μ global mean of feature.

$$F = \frac{\sum_{j=1}^{k} p_j (\mu_j - \mu)^2}{\sum_{j=1}^{k} p_j \sigma_j^2} \quad (1)$$

For Fisher score only the selected point distances are calculated. Calculation takes place between zero point and points sorted in previous step. After that the distance is normalised. It is standard tehnique since faces differ in size. Normalisation means taking the calculated distance and dividing it with that subjects face length. For face length the distances of chin center and forehead center given by Kinect are used. Score is calculated for each feature separately and all three emotions are taken into account. They are sorted by value and saved with point reference to a different file. From there most influential points for the model are selected.

Since Kinect outputs extensive amount of information to be used for facial expression recognition, incorporating Bezier curves [10] into a set of recognition features is a reasonable step forward to get more meaningful information. In this thesis Bezier curve is being used to form a curve between lip corners. For the algorithm there is used 3 facepoints: MouthRightCorner, MouthLeftCorner, MouthLowerLipMidBottom (Figure 11). There is also an opportunity to use MouthUpperLipMidBottom but in this thesis it is not used to calculate the Bezier curves because using lower lip point has generally given better results. The program fragment in Figure 11 returns the 100 points for each curve to be constructed of. Curves formed from neutral face is compared to curves formed from

happy face and curves formed from sad face. This comparison is made with Dynamic Time Warping (DTW) algorithm [33] that compares two curves and gives the minimal distance between the two of them. Distances are saved into separate file.

```
for i in range(0, 100):
  P0 = face_matrix[10:11, (687 * 3):((687 * 3) + 3)]
  P1 = face_matrix[10:11, (10 * 3):((10 * 3) + 3)]
  P2 = face_matrix[10:11, (91 * 3):((91 * 3) + 3)]
  B=(1 - t) ** 2 * P0 + 2 * (1 - t) * t * P1 + t ** 2 * P2

curveFeatures.get(n).get(file_to_read).append(B.flatten().
tolist())
  t += 1
```

Figure 11. Bezier curve calculation.

Files for all point distances, Fisher score and Bezier curve are equipped with control point if the file already exists and they are not over written.

4.5 Decision Tree

Decision Tree is a non- cyclic oriented flowchart like graph consisting of leaves and branches [5]. The decision tree gets a JSON file as an input with selected features. The file consists of distances between emotions and Bezier curves which were selected using Fisher score. Program converts JSON to dataframe type to make it easier for the selection of different columns and splitting the training and test sets. Training and test set are split with the ratio of 80:20. Decision Tree Classifier comes from the sklearn library. After creating the training and test sets they are put into Decision Tree Classifier and the results are tested.

Selected features are the 4 with the highest Fisher score [9] calculated and Bezier curve [10] from lip corners. The four features refer to four points on the face. The exact location cannot be named beacause they are not named by Kinect although the region they are located can be looked up on 2D face model (Figure 12) [34].

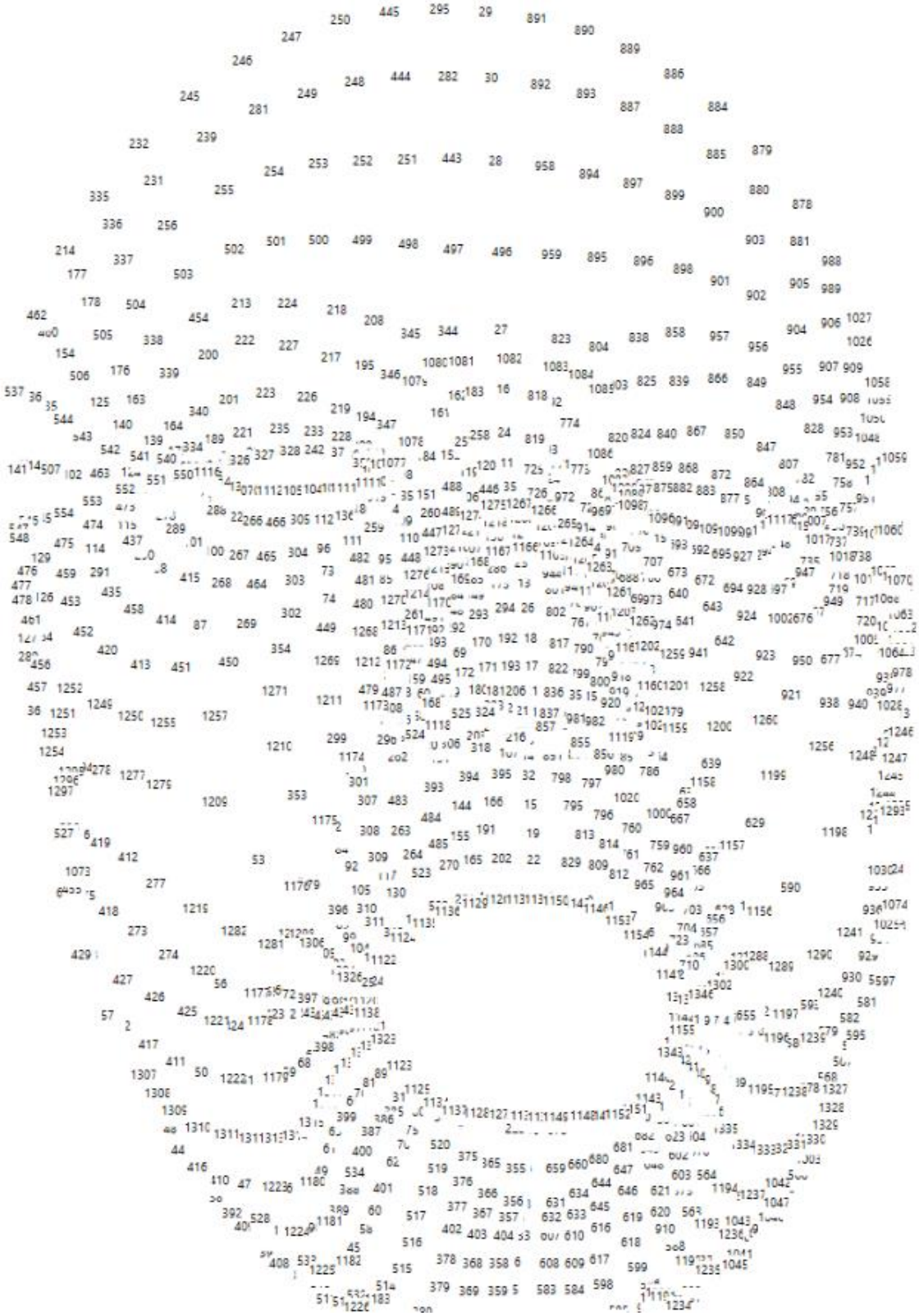

Figure 12. 2D face model of Kinect Face Tracking feature [34].

Face model shows that point 557 (Feature 1) should belong to the right eye region since point 554 is visible. Feature 2 point 1333 can be found on the left side of the face between lower lip corner and jaw line. The face model in Figure 13 is not ideal covering many points at transition regions such as surroundings of lips, eyes and nose. Since there isn't a full list available, of where each point is, the model is the best source to derive that information. Feature number 3 point 558 should also belong to the right eye region but looking at the model the numbers don't run in succsession. Feature 4 point 296 is not to be seen but point number 294 is on the nose. Points used to calculate Bezier curve 687 and 91 are lip corner points and named accordingly. Point number 10 belongs to lower lip mid top and is used to get the curve shape out of Bezier curve. Using only two points draws a straight line.

Experiments with different selections of points show that there is no need to select more features to classifier because they don't give new information. The data sets fall into the same cathegory as the first 4 features and the accuracy score stays about the same. Adding more features just increases the computing time which is not needed.

The features are given in JSON file. First thing to do is to convert JSON to data frame type for combining the 4 distance features and Bezier curve data into one file. Then the data is split forming the training and test sets. Test set includes 20 instances. Even that gives surprisingly high accuracy score of 89%.

5 Validation of results and their usability analysis

A facial expressions leaning case study has been conducted using about 30 test subjects and Kinect [4] image capturing system for collecting the data. The feature calculation and selection has been programmed by the author in Python and the results were implemented in the classifier model.

Model accuracy of 89% is achived with 5 features. Running the tests with one person test data reveals that the model relies mostly to the 5th feature when deciding on an emotion. Feature number 5 is the Bezier curve connecting between lip corners and lower lip point. When the curve distances are similar, the other features weigh more in deciding on an emotion. Validation of the result was made in decision tree with 20 instances, in addition 15 manul tests were enforced. Failure rate for them was 13%, neutral emotion was detected with 100% accuracy. The model labeled one instance of happy emotion to neutral and one instance of sad emotion to happy. Classification report shows accuracy for each class separetly as well as combined (Figure 13).

	precision	recall	f1-score	support
0.0	0.83	0.83	0.83	6
1.0	0.86	0.86	0.86	7
2.0	1.00	1.00	1.00	6
avg / total	0.89	0.89	0.89	19

Figure 13. Model classification report.

Boundaries for the method created include not using data directly from the source. Data that is used, is processed previously to get the distances that define the features. Directly giving image data from Kinect to the model is not reasonable and presumably won't work in expected way . The model would interprete the point data as distance data and give a false prediction. Files from Kinect are usually big and would take also more computing time.

6 Summary

The goal of this thesis is to study and apply facial expression recognition using decision trees. Thesis consist of data collecting, data pre-processing, feature calculation, classificator building and validation of the results. The model developed as a result of thesis work is capable of recognizing three facial expressions, sad, happy and neutral, with the accuracy score of 89%. This gives a trustworthy apprehension of the methods usability.

For future research there is an option to extend the facial expression recognition also for the other four main emotions and to test the scalability of the decision tree classifier in the presence of all of them. Another challenging task is to incorporate the body language recognition with facial expressions to estimate the mismatch between the facial expression and real emotion. Third potential research line would be using the facial emotions to interprete the spoken speech. The last is actual research topic in human-machine interfaces design domain.

References

[1] Google, "Google Home," [Online]. Available: https://store.google.com/gb/product/google_home?hl=en-GB.

[2] Apple Inc., "HomePod," [Online]. Available: https://www.apple.com/homepod/. [Accessed 28 04 2018].

[3] realeyes, "Case Studies," [Online]. Available: https://www.realeyesit.com/results. [Accessed 28 04 2018].

[4] Microsoft, "Kinect for Windows SDK Beta," 15 04 2011. [Online]. Available: https://www.microsoft.com/en-us/research/project/kinect-for-windows-sdk-beta/?from=http%3A%2F%2Fresearch.microsoft.com%2Fen-us%2Fum%2Fredmond%2Fprojects%2Fkinectsdk%2Fdownload.aspx. [Accessed 04 05 2018].

[5] scikit learn, "1.10 Decision trees," [Online]. Available: http://scikit-learn.org/stable/modules/tree.html. [Accessed 04 05 2018].

[6] Microsoft, "Kinect for Windows Sensor Components and Specifications," Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/jj131033.aspx. [Accessed 28 04 2018].

[7] V. Pternas, "How to use Kinect HD Face," 06 06 2015. [Online]. Available: https://pterneas.com/2015/06/06/kinect-hd-face/. [Accessed 10 01 2018].

[8] R. I. Jennrich and P. F. Sampson, "Newton-Raphson and related algorithms for maximum likelihood variance component estimation. Technometrics, 18, .," *Technometrics,* vol. 18, no. Feb., pp. 11-17, 1976.

[9] C. C. Aggarwal, Data Mining The Textbook, Springer.

[10] javascript.info, "Bezier curve," [Online]. Available: https://javascript.info/bezier-curve. [Accessed 04 05 2018].

[11] Imperial College London, "ibug," Department of Computing, [Online]. Available: https://ibug.doc.ic.ac.uk/publications/by_theme/face-analysis/. [Accessed 10 01 2018].

[12] Vokaturi, "Vokaturi," [Online]. Available: https://vokaturi.com/. [Accessed 15 03 2018].

[13] Imperial College London, "ibug," Department of Computing, [Online]. Available: https://ibug.doc.ic.ac.uk/home. [Accessed 10 01 2018].

[14] G. Sandbach, S. Zafeiriou, M. Pantic and D. Rueckert, "Recognition of 3D facial expression dynamics," 2012. [Online]. Available: https://ibug.doc.ic.ac.uk/media/uploads/documents/sandbach2012recognition.pdf. [Accessed 20 01 2018].

[15] M. F. Valstar, B. Jiang, M. Mehu, M. Pantic and K. Scherer, "The First Facial Expression Recognition and Analysis Challenge," [Online]. Available: https://ibug.doc.ic.ac.uk/media/uploads/documents/pdf17.pdf. [Accessed 20 02 2018].

[16] S. Bansal and P. Nagar, "Emotion recognition from facial expression based on Bezier curve," 2015. [Online]. Available: http://aircconline.com/ijait/V5N6/5615ijait01.pdf. [Accessed 25 03 2018].

[17] "Cohn-Kanade AU-Coded Expression Database," Affect Analysis Group, [Online]. Available: http://www.pitt.edu/~emotion/ck-spread.htm. [Accessed 28 04 2018].

[18] "The Japanese Female Facial Expression (JAFFE) Database," [Online]. Available: http://www.kasrl.org/jaffe.html. [Accessed 15 03 2018].

[19] "MMI Facial Expression Database," 2002. [Online]. Available: https://mmifacedb.eu/. [Accessed 15 03 2018].

[20] A. Linn, "Microsoft The AI Blog," Microsoft, 2015. [Online]. Available: https://blogs.microsoft.com/ai/microsofts-project-oxford-helps-developers-build-more-intelligent-apps/. [Accessed 15 03 2018].

[21] Microsoft, "Face API," [Online]. Available: https://azure.microsoft.com/en-us/services/cognitive-services/face/?cdn=disable. [Accessed 28 04 2018].

[22] "Facial Expressions Wild," 2017. [Online]. Available: https://sites.google.com/site/emotiwchallenge/challenge-details.

[23] L. Tan, K. Zhang, K. Wang, X. Zeng, X. Peng and Y. Qiao, "Group Emotion Recognition with Individual Facial Emotion," [Online]. Available: https://kpzhang93.github.io/papers/icmi.pdf. [Accessed 25 03 2018].

[24] B. Farnsworth, "Facial Action Coding System (FACS) – A Visual Guidebook," 6 December 2016. [Online]. Available: https://imotions.com/blog/facial-action-coding-system/.

[25] Affective Sciences, "Swiss Center for Affective Sciences," [Online]. Available: http://www.affective-sciences.org/gemep. [Accessed 15 03 2018].

[26] E. Chen, "Choosing a Machine Learning Classifier," [Online]. Available: http://blog.echen.me/2011/04/27/choosing-a-machine-learning-classifier/. [Accessed 01 04 2018].

[27] Paul Ekman Group LLC., "Micro Expressions," [Online]. Available: https://www.paulekman.com/micro-expressions/. [Accessed 28 04 2018].

[28] Body Language Experts, "Happiness as emotion," 06 06 2013. [Online]. Available: http://www.bl-expert.com/emotions/happiness-as-emotion/. [Accessed 28 04 2018].

[29] Body Language Experts, "Sadness as emotion," 06 06 2013. [Online]. Available: http://www.bl-expert.com/emotions/sadness-as-emotion/. [Accessed 30 04 2018].

[30] Wolfram MathWorld, "Distance," Wolfram Reasearch Inc., [Online]. Available: http://mathworld.wolfram.com/Distance.html. [Accessed 04 05 2018].

[31] Wolfram MathWorld, "Mean Deviation," Wolfram Research Inc., [Online]. Available: http://mathworld.wolfram.com/MeanDeviation.html. [Accessed 04 05 2018].

[32] Wolfram MathWorld, "Standard Deviation," Wolfram Research Inc, [Online]. Available: http://mathworld.wolfram.com/StandardDeviation.html. [Accessed 04 05 2018].

[33] MathWorks, "dtw," The MathWorks Inc., [Online]. Available: https://se.mathworks.com/help/signal/ref/dtw.html. [Accessed 05 05 2018].

[34] "Kinect 2 Face HD," [Online]. Available: https://social.msdn.microsoft.com/Forums/getfile/668131. [Accessed 05 05 2018].

Appendix 1 – Code

constants.py

```
import csv
import glob
import json
import os.path
import numpy as np

sad = 0
happy = 1
neut = 2

emotionList = [sad, happy, neut]

files = glob.glob("faces/*")

distance = "distance"

# Results
all_points_result = "results/positions.json"
bezier_curves_result = "results/curves.json"
fisher_result = "results/fisher.json"
bezier_floats_result = "results/bezier_floats1.json"
bezier_floats_result_sad = "results/bezier_floats0.json"
bezier_floats_neut = "results/bezier_floats_neut.json"
distances_for_tree = "results/distances_for_tree.json"


def get_emotion_from_file(file):
  n = 3
  if file.endswith("_sad.csv") or
file.endswith("_kurb.csv"):
    n = sad
  elif file.endswith("_happy.csv"):
    n = happy
  elif file.endswith("_neut.csv") or
file.endswith("_neutral.csv"):
    n = neut
  return n


def calculate_face_matrix(file):
  data_rows = get_data_rows(file)
  lenght1 = len(data_rows) + 1
  lenght2 = len(data_rows[2]) + 1
  face_matrix = np.empty([lenght1, lenght2])
  for i in range(2, len(data_rows)):
    a = np.asarray(data_rows[i])
    for j in range(0, len(data_rows[2])):
      face_matrix[i - 2, j] = float(a[j])
  return face_matrix


def get_data_rows(file):
  with open(file, 'r') as file:
    data_rows = list(csv.reader(file, delimiter=';'))
```

```
    return data_rows


def calculate_points_from_positions(points):
  new_points = []
  for point in points:
    if point > 1346:
      multiplier = point // 1346
      score = point - (multiplier * 1346)
      new_points.append(score)
    else:
      score = point
      new_points.append(score)
  return new_points


def write_to_file(file_name, content):
  with open(file_name, 'w') as jsonFile:
    json.dump(content, jsonFile)


def load_file(file_name):
  with open(file_name) as data_file:
    data = json.load(data_file)
  return data


def is_file_exists(path):
  return os.path.isfile(path)


def is_action_verified(action):
  text = input("There is response file for " + action + "
do you need to replace the old one? Y/N")
  if text is "Y" or text is "y":
    return True
  else:
    return False
```

all_point_distances.py

```
      from scipy.spatial.distance import euclidean
      from constants import *

      features = {}

      features.update(
          {
            distance: [
              {
                sad: []
              },
              {
                happy: []
              },
              {
                neut: []
              }]
          }
        )


      class Distance(object):
        def __init__(self, distance, position):
          self.distance = distance
          self.position = position


      def get_distance(dist):
        return dist.distance


      def calculate_all_point_distances():
        for file_to_read in files:
          n = get_emotion_from_file(file_to_read)
          face_matrix = calculate_face_matrix(file_to_read)

          # NoseTop point 24
          zeroPoint = face_matrix[10:11, 72:75]

          x = 0
          z = 3
          file_lenght = len(get_data_rows(file_to_read)[2])
          for i in range(2, file_lenght):
            anotherPoint = face_matrix[10:11, x:z]
            dist = euclidean(zeroPoint, anotherPoint)
            features.get(distance)[n].get(n).append(dist)
            if point_ends == 4041:
              break
            point_starts += 3
            point_ends += 3


        zeroPointFace = features.get(distance)[2]
        happyPointFace = features.get(distance)[1]
```

```
        sadPointFace = features.get(distance)[0]

        difference = []
        m = -1

        print('pointface calculation')
        for i in range(0, len(zeroPointFace[2])):
          for j in range(0, len(happyPointFace[1])):
            if i == j:
              m = m + 1
              var1 = zeroPointFace[2][i]
              var2 = happyPointFace[1][j]
              happy_no = abs(var1 - var2)
              sad_no = abs(var1 - sadPointFace[0][j])
              difference.append(Distance(happy_no, m))
              difference.append(Distance(sad_no, m))

        positions = []
        i = 0
        distListFace=sorted(difference,key=get_distance)

        for i in range(0, len(distListFace)):
          i += 1
          if i >= len(distListFace):
            break

      positions.append(distListFace.__getitem__(i).position)

        positionsReversed = []
        for position in reversed(positions):
          positionsReversed.append(position)

        first_hundred = positionsReversed[:100]
        write_to_file(all_points_result, first_hundred)


      if is_file_exists(all_points_result) is not True:
        print("FIle is not there")
        calculate_all_point_distances()
      else:
        if is_action_verified("Calculate all point distances")
      is True:
          print("Start calculating point distances...")
          calculate_all_point_distances()
          print("Finished calculating point distances.")
```

fisher.py

```
      from scipy.spatial.distance import euclidean
      from constants import *

      features = {}
      fisherScores = []


      # Arvuta P (kui palju esineb klassi featuuri sees)
      # Eeldused: Klasse on 3
      def calculate_p(feature, emotion):
        divider = 0
        for emo in emotionList:
          divider = divider + len(feature[emo][emo])
        return len(feature[emotion][emotion]) / divider


      # Arvuta standardhälve (np.std)
      def calculate_standard_deviation(feature, emotion):
        return np.std(feature[emotion][emotion])


      # Arvuta featuuri ühe klassi aritmeetiline keskmine
      def calculate_mean_deviation(feature, emotion):
        return sum(feature[emotion][emotion]) /
      len(feature[emotion][emotion])


      # Arvuta featuuri kõikide klasside aritmeetiline keskmine
      def calculate_global_mean_deviation(feature):
        answer = 0
        count = -1
        for emotion in emotionList:
          answer = answer + sum(feature[emotion][emotion])

          if count == -1:
            count = len(feature[emotion].get(emotion))
          else:
            count=count+len(feature[emotion].get(emotion))

        return answer / count


      # Arvuta fisheri skoor!
      def calculate_fisher_score(feature):
        top_fraction = 0
        bottom_fraction = 0

        for emotion in emotionList:
          fraction = calculate_p(feature, emotion)
          mean_dev=calculate_mean_deviation(feature,emotion)
          global=calculate_global_mean_deviation(feature)
        stn_dev=calculate_standard_deviation(feature,emotion)
          top_fraction = top_fraction + (fraction *
      ((mean_deviation - global_mean_deviation) ** 2))
```

```
        bottom_fraction = bottom_fraction + (fraction *
    (standard_deviation)**2)

      fisher_score = top_fraction / bottom_fraction

      return fisher_score


    def calculate_fisher_from_points():
      for index, newPoint in
    enumerate(calculate_points_from_positions(load_file(all_p
    oints_result))):
        distance = 'feature' + str(index)
        features.update(
          {
            distance: [
              {
                sad: []
              },
              {
                happy: []
              },
              {
                neut: []
              }]
          }
        )

        for file_to_read in files:
         n = get_emotion_from_file(file_to_read)
         face_matrix=calculate_face_matrix(file_to_read)

         # NoseTop p 24
         zeroPoint=face_matrix[10:11,(24*3):((24*3)+3)]
         chinCenter = face_matrix[10:11,(4*3):((4*3)+3)]
        foreheadCent=face_matrix[10:11,(28*3):((28*3)+3)]

         feature=
    face_matrix[10:11,(newPoint*3):((newPoint*3)+3)]
         dist1 = euclidean(zeroPoint, feature)

         dist01 = euclidean(chinCenter, foreheadCenter)
         normalisationForMainPoint = dist1 / dist01
         norm = normalisationForMainPoint

         features.get(distance)[n].get(n).append(norm)


        score=
    str(calculate_fisher_score(features.get(distance)))
        print(distance + " Fisher score is " + score)
        fisherScores.append({"feature": distance, "score":
    score, "point": newPoint})

      fisherScores.sort(key=lambda x: x['score'],
```

```
        reverse=True)
          write_to_file(fisher_result, fisherScores)
          write_to_file(distances_for_tree, features)


        if is_file_exists(fisher_result) is not True:
          print("File is not there!")
          calculate_fisher_from_points()
          print("Finished calculating fisher score!")
        else:
          if is_action_verified("Calculate fisher score from
        points") is True:
            print("Start calculating fisher score...")
            calculate_fisher_from_points()
            print("Finished calculating fisher score!")
```

bezier_curve.py

```
    from scipy.spatial.distance import euclidean
    from constants import *
    from dtw import dtw

    curves = "curves"

    curveFeatures = {
        sad: {},
        happy: {},
        neut: {}
    }


    def calculate_bezier_curve(files_to_read):
        for index, file_to_read in enumerate(files_to_read):

            n = get_emotion_from_file(file_to_read)
            face_matrix = calculate_face_matrix(file_to_read)

            curveFeatures.get(n).update(
                {
                    file_to_read: []
                }
            )

            # Bezier curve for mouth 0<=t<=1
            t = 0
            for i in range(0, 100):
                P0=face_matrix[10:11,(687*3):((687*3) + 3)]
                P1=face_matrix[10:11,(10*3):((10*3) + 3)]
                P2=face_matrix[10:11,(91*3):((91*3) + 3)]
                B=(1-t)**2*P0+2*(1-t)*t*P1+t**2*P2


    curveFeatures.get(n).get(file_to_read).append(B.flatten()
    .tolist())
                t += 1
        return curveFeatures


    def get_person_emotion(emotion, file_name):
        person_name = file_name.split("_")[0]
        feature = curve_features.get(str(emotion))
        for file_name in feature:
            if person_name in file_name:
                return feature.get(file_name)


    if is_file_exists(bezier_curves_result) is not True:
        print("File is not there!")
        write_to_file(bezier_curves_result,
    calculate_bezier_curve(files))
    else:
        if is_action_verified("Calculate Bezier Curves") is
```

```
        True:
                print("Start calculating Bezier curve...")
                write_to_file(bezier_curves_result,
        calculate_bezier_curve(files))

        print("Load Bezier result..")
        curve_features = load_file(bezier_curves_result)
        neutral_curve = curve_features.get(str(neut))

        floats = []
        i=1
        for neutral_file_name in neutral_curve:
            x = neutral_curve.get(neutral_file_name)
            #y=neutral_curve.get(neutral_file_name)[i]
            y = get_person_emotion(happy, neutral_file_name)
            path = dtw(x, y, dist=euclidean)
            floatPoint = path[0]
            print(floatPoint)
            floats.append(floatPoint)
            i += 1
        #floats.sort(key=lambda x: x['float'], reverse=True)
        write_to_file(bezier_floats_result, floats)
```

decision_tree.py

```
import pandas as pd
from sklearn.cross_validation import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import json
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

def convertJSON(dataset):
  new_json = {"label": []}
  for index, data in enumerate(dataset):
    new_json.update({data: []})
    for index, distances in enumerate(dataset.get(data)):
      for float in distances.get(str(index)):
        new_json.get(data).append(float)
        if data == list(dataset.keys())[0]:
          new_json.get("label").append(index)
  return new_json


dataset =
json.load(open('results/distances_for_tree.json'))
sad=pd.DataFrame(json.load(open
('results/bezier_floats0.json')))
happy=pd.DataFrame(json.load(open
('results/bezier_floats1.json')))
neut=pd.DataFrame(json.load(open
('results/bezier_floats_neut.json')))

frames = [sad, happy, neut]
result = pd.concat(frames, ignore_index=True)

df = pd.DataFrame(convertJSON(dataset))

X = df.values[:, :4]
X = pd.concat([pd.DataFrame(X),result], axis=1)
Y = df.values[:, 6:]

# Train and test data split
X_train, X_test, y_train, y_test = train_test_split
( X, Y, test_size = 0.2, random_state = 100)

clf_gini = DecisionTreeClassifier(criterion = "gini",
random_state = 100, max_depth=3, min_samples_leaf=5)
clf_gini.fit(X_train, y_train)
```

```
DecisionTreeClassifier(class_weight=None,criterion='gini'
,max_depth=3,max_features=None,max_leaf_nodes=None,
min_samples_leaf=5,min_samples_split=2,min_weight_fractio
n_leaf=0.0,presort=False,random_state=100,
splitter='best')


clf_gini.predict(X_test)
y_pred = clf_gini.predict(X_test)
score = accuracy_score(y_test,y_pred)*100

# Make predictions on validation dataset
predictions = clf_gini.predict(X_test)
print(accuracy_score(y_test, predictions))
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))
```