PRE PROCESSING STEPS

1. Grayscale version
2. Face and eyes detection
3. Face straightening
4. Face Cropping
5. Image resizing
6. Normalization : We can use the normalize() function to apply visual normalization in order to fix very dark/light pictures (can even fix low contrast)

linear image transform (LIT) : ignores scanning a number of non-face windows.
regional minima (RM) : to reject non-face windows.
modified adaptive thresholding (ADT) technique : convert input image into a binary representation and perform an exclusion process on the latter form.

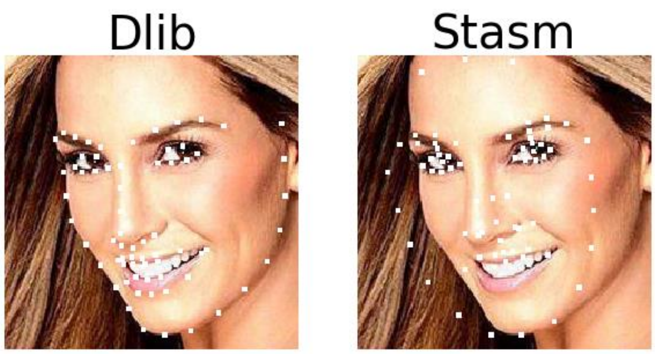
FACE DETECTION

1. OpenCV Haarcascade
2. OpenCV DNN (Deep Neural Network)
3. Detecting faces using Dlib

Related **What are the different types of face landmark detection algorithms used to date, and which one is best?**

If you are using Python, PyStasm and Dlib are freely available. However, PyStasm is not supported anymore, and its performance is lower than DLib. So I would recommend using DLib for face landmark detection. DLib can be easily installed using `pip`.

The following shows the landmarks detected using DLib and Stasm, and you can see that landmarks detected by Stasm are a bit off.



4. Mtcnn in Python
5. Viola Jones algorithm.

TECHNIQUES

Date

KNN && Decision Tree

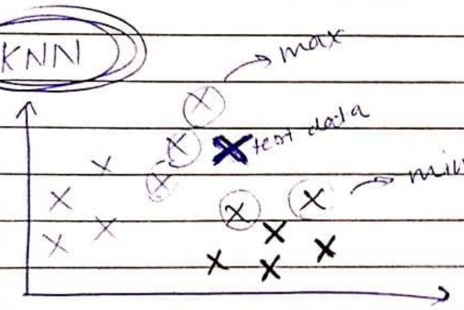
Haar like features → detect face & eyes using Viola Jones adaboost classification method

FAUS → Facial Action Units (10 virtual markers)
FACS → Facial action coding system

Lucas Kande optical flow algo → trace marker positions

dist between FAU at centre of the subject face to other markers are calculated and used as FEATURE

KNN



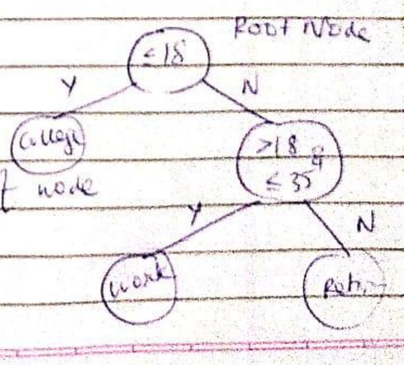
K = hyperparameter = 5 → nearest neighbours

↓ vary

highest accu. → ✓

Euclidean, Manhattan, Minkowski, Chebyshev

Decision Tree




```
if (age ≤18) :
    print ("college");
elif (age >18 && age ≤ 35):
    print ("Work")
else :
    print ("Retire")
```

chances of overfitting

Pure Split (All yes / 0 No)
(0 Yes / All No)

Purity → Pure Split
→ Entropy $H(S) = -(P_+)\log_2(P_+) - (P_-)\log_2(P_-)$
→ Gini Impurity $GI = 1 - \sum_{i=1}^{n} (P)^2$

Information Gain
→ $H(S) - \sum_{v \in n} \frac{|S_v|}{|S|} H(S_v)$
root node

A hierarchical tree is built using a bottom-up approach by recursively clustering and merging the classes at each level. This process is based on a similarity matrix, see Table 1, which represents how similar are the different log-likelihood facial expressions. For example, the lowest distance (i.e., 7.94) corresponds to neutral and anger expressions, so both are joined in the same node (i.e., node 1), and so on. The similarity matrix is then recalculated at each level of the tree with the resulting new classes. In this point it is worth mentioning that there are different topologies for the hierarchical tree. After testing several of them, the best results were reached with the structure depicted in Fig. 2.

Emotion	joy	anger	surprise	sadness	disgust	neutral
joy	0.00	16.21	18.92	17.57	16.28	16.76
anger			13.73	10.26	9.09	7.94
surprise				12.27	15.55	11.53
sadness					13.70	9.40
disgust						11.56
neutral						

Table 1. *Similarity matrix: Euclidean distance between the log-likelihood maps.*

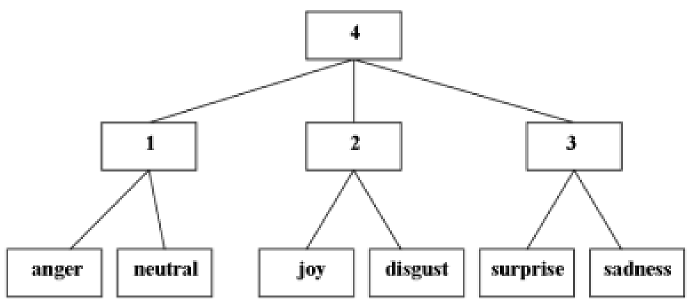


Fig. 2. *Hierarchical Decision Tree.*

Naive Bayes Classifier → Baye's Theor

↳ uses cauchy

Dependent $y \in \{0, 1, \ldots M\}$ class label

Independent $X \in R^n$ (feature vector) observed data

argmax → max value from target function

↳ commonly used to find class with largest predicted probability

$$\hat{y} = \underset{y}{\operatorname{argmax}} \; P(X/y)$$

$$\hat{y} = \underset{y}{\operatorname{argmax}} \prod_{i=1}^{N} P(x_i/y)$$

E = Class (emotions) = $c_1, c_2, \ldots C_n$

$$P(C_i/E) = \frac{P(C_i) * P(E/C_i)}{P(E)}$$

Using Bayes theorem

DATASET

Independent $x_1 \; x_2 \; x_3 \ldots x_n$

Output Dependent Y

$$L_i = \log\left(\frac{P(E, C_i)}{P(E)}\right)$$

log likelihood

↳ gives info on how discriminative features are

$$P(Y/x_1, x_2, x_3 \ldots x_n) = \frac{P(Y) * P(x_1 \; x_2 \; x_3 \ldots x_n / y)}{P(X_1 \; X_2 \; X_3 \ldots X_n)}$$

$$= \frac{P(Y) * P(X_1/y) * P(X_2/y) * \ldots P(X_n/y)}{P(X_1) * P(X_2) \ldots * P(X_n)}$$

since independent + constant

Gaussian Distribution ✗

Cauchy Distribution ✓

In gaussian case → only 2 parameters

Page No. | Date

$$L(x_i/y ; a_i, b_i) = \prod_{d=1}^{n} \left[\frac{b_i}{\pi \left(b_i^2 + (x_i^d - a_i)^2 \right)} \right]$$

for large $|x|$ mean variance, SD do not exist

median value & Interquartile range

a_i = location parameter
b_i = scale parameter
$i = 1, \ldots, N$
$\hat{a}_i, \hat{b}_i$ → max likelihood estimators for a_i, b_i

$$\sum_{d=1}^{n} \frac{x_i^d - \hat{a}_i}{\hat{b}_i^2 + (x_i^d - \hat{a}_i)^2} = 0$$

$$\sum_{d=1}^{n} \frac{\hat{b}_i^2}{\hat{b}_i^2 + (x_i^d - \hat{a}_i)^2} = \frac{n}{2}$$

Newton Raphson iterative method

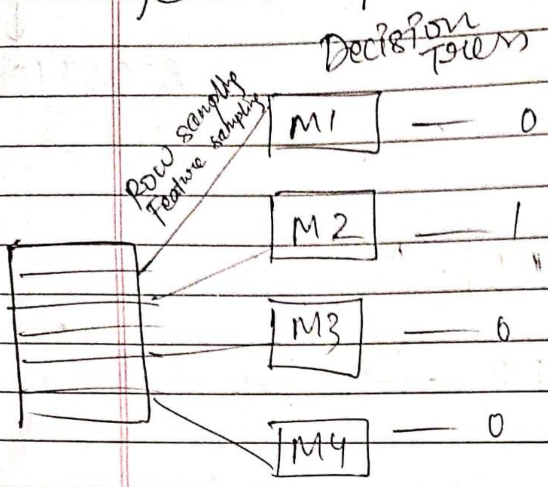
→ starting point as mean & variance of data

if diverged → select new starting points

Training set : estimation of parameters
Test set : classification

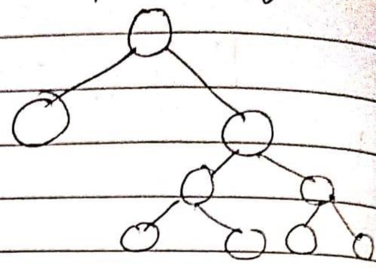
TP	FN
FP	TN

Random Forest Classifier & Regres



majority voting
↓ output = 0

main problm of DT



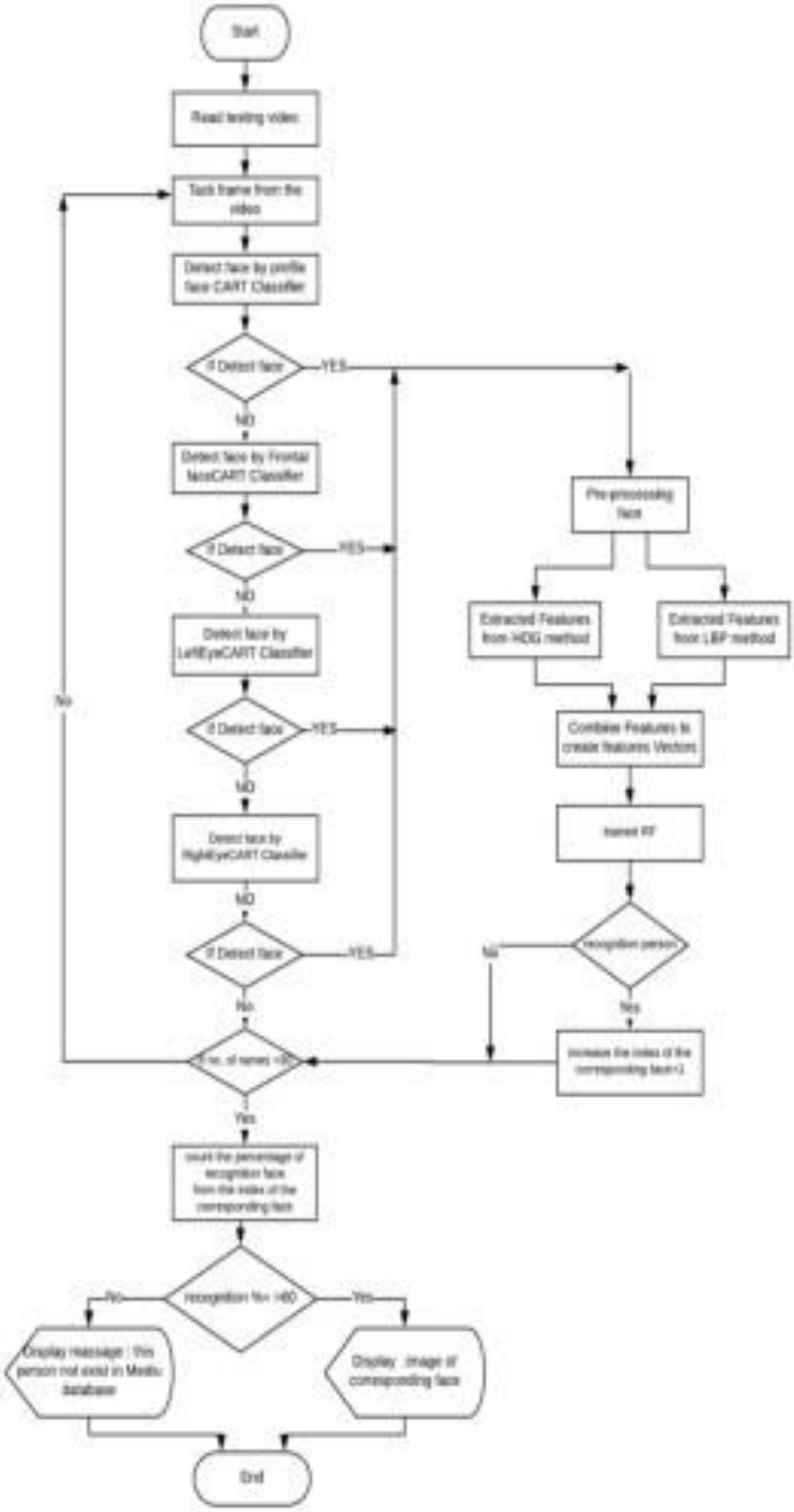
Overfitting

Low Bias
High Variance

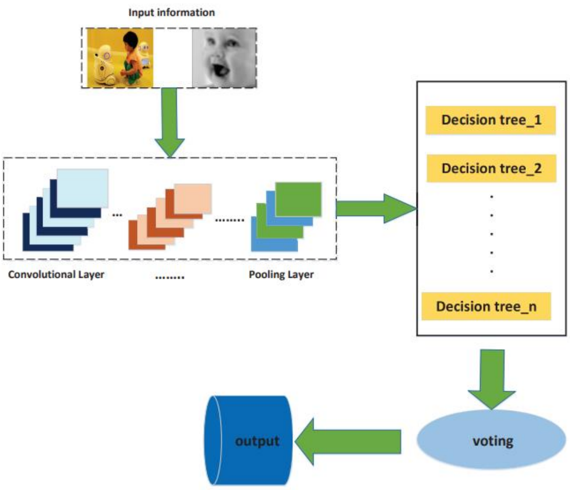
But in general, we need
Low Bias Low Variance

Differen, Random forest clas
reg → mean

Random Forest



CNN + Random Forest



Conclusion : Random Forest since it overcomes the problem of overfitting and gives accurate results