Abstract:
Given 3 Face datasets (data.mat, illumination.mat and pose.mat), I applied two primitive classification methods (Maximum Likelihood Bayesian and KNN) on them. After dividing the datasets randomly into training and testing, I calculated their accuracy (%) and compared their results.

I. Classification Techniques:

A. Maximum Likelihood Bayesian classification: -

- Segregate the datasets into training and testing data. Usually the training dataset is 70% or higher and testing dataset is 30% or lesser.
- Since we have a dataset which states that every class has 3 face images for data.mat, 21 face images for illumination.mat and 13 face images for pose.mat; the prior probabilities $P(x)$ will be the same for every class.
- Calculate the mean (μ) per class for the training dataset.
- Calculate the variance ($\sum$) and variance inverse ($\sum^{-1}$) for every class. Add a constant (α) along the diagonal if $|\sum| = 0$.
- To make a decision among two classes, w1 and w2; we calculate $g(x)$.

$$g(x) = X^T W X + w^T X + w_0$$
$$where,\ W = -\frac{1}{2}\sum^{-1}$$
$$w = \sum^{-1}\mu$$
$$w_0 = -\frac{1}{2}\mu^T \sum^{-1}\mu - \frac{1}{2}ln\left|\sum^{-1}\right| + ln\big(P(x)\big)$$

- Calculate these for all values in the test dataset and find the maximum value of $g(x)$. As we find the max value, we assign the label of that class value, thus making a decision per test sample.
- Finally, we compare the labels we assigned to the labels of the test dataset. Thus, we can find the accuracy of our classifier.

B. K Nearest Neighbor Classification: -

- Segregate the datasets into training and testing data. Usually the training data is 70% and testing is 30% or lesser than the training set.
- Since we have a dataset which states that every class has 3 face images for data.mat, 21 face images for illumination.mat and 13 face images for pose.mat; the prior probabilities $P(x)$ will be the same for every class.

- User is prompted to enter the value of *'K'*.
- For every sample in the testing dataset, we calculate the '*Euclidean distance'* to every sample in the training dataset. I have declared a '*distances'* vector which stores the Euclidean distance from a sample point in the testing dataset to every sample point in the training dataset.

$$d = \sqrt{\left((x-X)^T(x-X)\right)}$$
$$distances = d$$

- I have then pulled out *'K'* smallest distances (using *'mink'* function) from the '*distances'* vector.

$$[Dk, I] = mink(distances, K)$$

- As we find the min value/s, we assign the label of that class value; we then add the first *'K'* labels and take an average, thus making a decision per test sample.
- Finally, we compare the labels we assigned to the labels of the test dataset. Thus, we can find the accuracy of our classifier.

C. <u>**LDA-Bayesian Classification**</u>**: -**

- Segregate the datasets into training and testing data. Usually the training data is 70% and testing is 30% or lesser than the training set.
- Since we have a dataset which states that every class has 3 face images for data.mat, 21 face images for illumination.mat and 13 face images for pose.mat; the prior probabilities $P(x)$ will be the same for every class.
- Calculate the mean per class (μ_i) for the training dataset.
- Calculate the mean (μ_a) over all the classes i.e. find mean over all the training dataset instead of class-wise (μ_0).
- Now calculate the within class scatter (S_w) i.e. variance ($\sum$), and the between class scatter (S_B) matrix. Add a constant (α) along the diagonal if $|\sum| = 0$.

$$S_w = \sum_{i=1}^{c} \Sigma_i$$
$$S_B = \sum_{i=1}^{c} n_i\,(\mu_i - \mu_a)(\mu_i - \mu_a)^T$$

- I have used a MATLAB function to calculate the eigen values. The eigen values are stored in the *'EigVal'* vector and *'W'* is the new transformed matrix.

$$[W, EigVal] = eigs(S_B, S_w, C-1)$$

- Using the generated *'W',* we calculate the new transformed training and testing dataset by multiplying it to the original respective datasets; the new training and testing datasets are then fed to the Bayesian classifier for decision.
- Calculate the mean (μ_{new}) per class for the new training dataset.
- Calculate the variance ($\sum_{new}$) and variance inverse (${\sum_{new}}^{-1}$) for every class. Add a constant (β) along the diagonal if $|\sum_{new}| = 0$.
- To make a decision among two classes, w1 and w2; we calculate $g(x)$.

$$g(x) = {X_{new}}^T W X_{new} + w^T X_{new} + w_0$$
$$where,\ W = -\tfrac{1}{2}{\textstyle\sum_{new}}^{-1}$$
$$w = {\textstyle\sum_{new}}^{-1}\mu_{new}$$
$$w_0 = -\tfrac{1}{2}{\mu_{new}}^T{\textstyle\sum_{new}}^{-1}\mu_{new} - \tfrac{1}{2}ln\left|{\textstyle\sum_{new}}^{-1}\right| + ln\big(P(x)\big)$$

- Calculate these for all values in the new test dataset and find the maximum value of $g(x)$. As we find the max value, we assign the label of that class value, thus making a decision per test sample.
- Finally, we compare the labels we assigned to the labels of the test dataset. Thus, we can find the accuracy of our classifier.

D. <u>PCA-Bayesian Classification</u>: -

- Segregate the datasets into training and testing data. Usually the training data is 70% and testing is 30% or lesser than the training set.
- Since we have a dataset which states that every class has 3 face images for data.mat, 21 face images for illumination.mat and 13 face images for pose.mat; the prior probabilities $P(x)$ will be the same for every class.
- I have used a MATLAB function to compute the '*W, U, S, V'* matrices which are used as follows:

$$[W, S, V] = svds(Dataset_{train}, C - 1)$$
$$A = USV^T$$

- Using the generated *'W',* we calculate the new transformed training and testing dataset by multiplying it to the original respective datasets; the new training and testing datasets are then fed to the Bayesian classifier for decision.
- Calculate the mean (μ_{new}) per class for the new training dataset.
- Calculate the variance ($\sum_{new}$) and variance inverse (${\sum_{new}}^{-1}$) for every class. Add a constant (θ) along the diagonal if $|\sum_{new}| = 0$.
- To make a decision among two classes, w1 and w2; we calculate $g(x)$.

$$g(x) = {X_{new}}^T W X_{new} + w^T X_{new} + w_0$$
$$where,\ W = -\tfrac{1}{2}{\textstyle\sum_{new}}^{-1}$$
$$w = {\textstyle\sum_{new}}^{-1}\mu_{new}$$

$$w_0 = -\frac{1}{2}\mu_{new}{}^T\Sigma_{new}{}^{-1}\mu_{new} - \frac{1}{2}ln\left|\Sigma_{new}{}^{-1}\right| + ln\big(P(x)\big)$$

- Calculate these for all values in the new test dataset and find the maximum value of $g(x)$. As we find the max value, we assign the label of that class value, thus making a decision per test sample.
- Finally, we compare the labels we assigned to the labels of the test dataset. Thus, we can find the accuracy of our classifier.

E. <u>LDA-K Nearest Neighbor Classification</u>: -

- Segregate the datasets into training and testing data. Usually the training data is 70% and testing is 30% or lesser than the training set.
- Since we have a dataset which states that every class has 3 face images for data.mat, 21 face images for illumination.mat and 13 face images for pose.mat; the prior probabilities $P(x)$ will be the same for every class.
- Calculate the mean per class (μ_i) for the training dataset.
- Calculate the mean (μ_a) over all the classes i.e. find mean over all the training dataset instead of class-wise (μ_0).
- Now calculate the within class scatter (S_w) i.e. variance (Σ), and the between class scatter (S_B) matrix. Add a constant (α) along the diagonal if $|\Sigma| = 0$.

$$S_w = \sum_{i=1}^{c} \Sigma_i$$

$$S_B = \sum_{i=1}^{c} n_i\,(\mu_i - \mu_a)(\mu_i - \mu_a)^T$$

- I have used a MATLAB function to calculate the eigen values. The eigen values are stored in the *'EigVal'* vector and *'W'* is the new transformed matrix.

$$[W, EigVal] = eigs(S_B, S_w, C - 1)$$

- Using the generated *'W',* we calculate the new transformed training and testing dataset by multiplying it to the original respective datasets; the new training and testing datasets are then fed to the Bayesian classifier for decision.
- User is prompted to enter the value of *'K'*.
- For every sample in the new testing dataset, we calculate the '*Euclidean distance'* to every sample in the new training dataset. I have declared a '*distances'* vector which stores the Euclidean distance from a sample point in the testing dataset to every sample point in the training dataset.

$$d = \sqrt{\big((x_{new} - X_{new})^T(x_{new} - X_{new})\big)}$$
$$distances = d$$

- I have then pulled out *'K'* smallest distances (using *'mink'* function) from the '*distances'* vector.

$$[Dk, I] = mink(distances, K)$$

- As we find the min value/s, we assign the label of that class value; we then add the first *'K'* labels and take an average, thus making a decision per test sample.
- Finally, we compare the labels we assigned to the labels of the test dataset. Thus, we can find the accuracy of our classifier.

F. <u>PCA-K Nearest Neighbor Classification</u>: -

- Segregate the datasets into training and testing data. Usually the training data is 70% and testing is 30% or lesser than the training set.
- Since we have a dataset which states that every class has 3 face images for data.mat, 21 face images for illumination.mat and 13 face images for pose.mat; the prior probabilities $P(x)$ will be the same for every class.
- I have used a MATLAB function to compute the '*W, U, S, V'* matrices which are used as follows:

$$[W, S, V] = svds(Dataset_{train}, C - 1)$$
$$A = USV^T$$

- Using the generated *'W',* we calculate the new transformed training and testing dataset by multiplying it to the original respective datasets; the new training and testing datasets are then fed to the Bayesian classifier for decision.
- User is prompted to enter the value of *'K'*.
- For every sample in the testing dataset, we calculate the '*Euclidean distance'* to every sample in the training dataset. I have declared a '*distances'* vector which stores the Euclidean distance from a sample point in the testing dataset to every sample point in the training dataset.

$$d = \sqrt{\left((x_{new} - X_{new})^T (x_{new} - X_{new})\right)}$$
$$distances = d$$

- I have then pulled out *'K'* smallest distances (using *'mink'* function) from the '*distances'* vector.

$$[Dk, I] = mink(distances, K)$$

- As we find the min value/s, we assign the label of that class value; we then add the first *'K'* labels and take an average, thus making a decision per test sample.
- Finally, we compare the labels we assigned to the labels of the test dataset. Thus, we can find the accuracy of our classifier.

II. Running the codes:

- I have 8 codes named:
  - *ML_Bayesian.m*
  - *LDA_Bayesian.m*
  - *PCA_Bayesian.m*
  - *KNN.m*
  - *LDA_KNN.m*
  - *PCA_KNN.m*
  - *Final1.m* (**All classifiers are integrated**)
  - *Final2.m* (**data.mat with C=2 integrated for all classifiers**)

- All codes are independent. For ease of running and testing, I have integrated all the datasets and classifiers under **Final1.m** and **Final2.m** code-file.
  You can run Final1.m/Final2.m or run each code individually.
  Final1.m:
  - data.mat with C=200 for all classifiers.
  - illumination.mat for all classifiers.
  - pose.mat for all the classifiers.

  Final2.m:
  - Designed specifically to meet the first type of experiment with C=2 for data.mat.
  - All classifiers are designed for it.

- You will be prompted to test on any of the datasets (data.mat, illumination.mat and pose.mat). Enter ***'1'** for data.mat*, ***'2'** for illumination.mat* and ***'3'** for pose.mat*.

- In case you plan to manually assign dataset index, you can do so to the Dt_test but make sure you change the value of the variables, train_set and test_set, since it keeps a count of how many samples are being used for training and testing.
  - E.g. Say you want to test the pose.mat
    - You want to assign 4 values to testing, j = 1, 3, 7, 9 to Dt_test, and rest 9 to training, Dt_train, you can do so.
    - Just change the variable: test_set = (4 values) * 68 = 272 and train_set = (9 values) * 68 = 612.
    - When you change the number of testing indices, you will have to change the variable named test_set and train_set too. Else accuracy won't be calculated correctly/accurately.
    - After this when you run, enter '3' when prompted for 'pose.mat'.

III. Accuracy & Analysis:

Bayesian Classification Test Results

S. No.	METHOD	DATASET	TESTING INDEX	TRAINING INDEX	ACCURACY	AVERAGE
A.	**ML-Bayesian**	data.mat (C=200)	❖ 3 (Illumination) ❖ 2 (Expression) ❖ 1 (Neutral)	❖ 1, 2 ❖ 1, 3 ❖ 2, 3	❖ 64% ❖ 66.5% ❖ 72%	67.5%
		data.mat (Neutral & Expression)	❖ 80 images (40 from each class)	❖ 320 images (160 from each class)	❖ 68.75%	68.75%
		illumination.mat	❖ 2, 5, 10, 12, 16, 19 ❖ 1, 3, 9, 13, 17 ❖ 1-15	❖ Remaining 15 ❖ Remaining 16 ❖ 16-21	❖ 100% ❖ 100% ❖ 100%	100%
		pose.mat	❖ 10-13 ❖ 2, 4, 7, 13 ❖ 1, 5, 9	❖ 1-9 ❖ Remaining 9 ❖ Remaining 10	❖ 75% ❖ 65.8% ❖ 99.5%	80.1%
B.	**PCA Bayesian**	data.mat (C=200)	❖ 3 ❖ 2 ❖ 1	❖ 1, 2 ❖ 1, 3 ❖ 2, 3	❖ 63.5% ❖ 66.5% ❖ 70.5%	66.8%
		data.mat (Neutral & Expression)	❖ 80 images (40 from each class)	❖ 320 images (160 from each class)	❖ 51.25%	51.25%
		illumination.mat	❖ 16-21 ❖ 1, 4, 9, 11, 17, 21 ❖ 2, 6, 10, 20	❖ 1-15 ❖ Remaining 15 ❖ Remaining 17	❖ 100% ❖ 100% ❖ 100%	100%
		pose.mat	❖ 10-13 ❖ 2, 5, 9, 13 ❖ 3, 4, 10, 13	❖ 1-9 ❖ Remaining 9 ❖ Remaining 9	❖ 79.8% ❖ 77.2% ❖ 91.2%	82.7%
C.	**LDA Bayesian**	data.mat (C=200) (alpha: 1, beta: 1)	❖ 3 ❖ 2 ❖ 1	❖ 1, 2 ❖ 1, 3 ❖ 2, 3	❖ 61% ❖ 62% ❖ 67.5%	63.5%
		data.mat (C=200) (alpha: 0.05, beta:1)	❖ 3 ❖ 2 ❖ 1	❖ 1, 2 ❖ 1, 3 ❖ 2, 3	❖ 55.5% ❖ 77% ❖ 88.5%	73.6%
		data.mat (C=200) (alpha: 0.02, beta:1)	❖ 3 ❖ 2 ❖ 1	❖ 1, 2 ❖ 1, 3 ❖ 2, 3	❖ 50.5% ❖ 79.5% ❖ 88%	72.6%
		data.mat (C=200) (alpha: 0.05, beta:0.05)	❖ 3 ❖ 2 ❖ 1	❖ 1, 2 ❖ 1, 3 ❖ 2, 3	❖ 55% ❖ 79.5% ❖ 88.5%	74.3%
		data.mat (Neutral & Expression) (alpha: 1, beta: 1)	❖ Class 1 (200 images)	❖ Class 2 (200 images)	❖ 96%	96%
		data.mat (Neutral & Expression)	❖ Class 1 (200 images)		❖ 82%	82%

		(alpha: 0.05, beta: 0.05)		❖ Class 2 (200 images)		
		data.mat (Neutral & Expression) (alpha: 0.05, beta: 0.05)	❖ 130 images (65 from each class)	❖ 270 images (135 from each class)	❖ 60.71%	60.71%
		illumination.mat (alpha: 1, beta: 1)	❖ 16-21 ❖ 1, 4, 11, 18, 21 ❖ 2, 4, 10, 14, 18, 20	❖ 1-15 ❖ Remaining 16 ❖ Remaining 15	❖ 100% ❖ 100% ❖ 100%	100%
		illumination.mat (alpha: 0.05, beta: 1)	❖ 16-21 ❖ 1, 4, 11, 18, 21 ❖ 2, 4, 10, 14, 18, 20	❖ 1-15 ❖ Remaining 16 ❖ Remaining 15	❖ 100% ❖ 100% ❖ 100%	100%
		pose.mat (alpha: 1, beta: 1)	❖ 10-13 ❖ 9-13 ❖ 1, 3, 7, 10, 13	❖ 1-9 ❖ 1-8 ❖ Remaining 8	❖ 80.9% ❖ 83.5% ❖ 76.2%	80.2%
		pose.mat (alpha: 0.05, beta: 1)	❖ 10-13 ❖ 9-13 ❖ 1, 3, 7, 10, 13	❖ 1-9 ❖ 1-8 ❖ Remaining 8	❖ 80.9% ❖ 83.5% ❖ 76.2%	80.2%

*** Table 1. Bayesian Results (PCA-Bayes and LDA-Bayes) ***

KNN Classification Test Results

S. No.	METHOD	DATASET	TESTING INDEX	TRAINING INDEX	ACCURACY	AVERAGE
D.	**KNN**	data.mat (K = 1) (C=200)	❖ 3 ❖ 2 ❖ 1	❖ 1, 2 ❖ 1, 3 ❖ 2, 3	❖ 60% ❖ 75.5% ❖ 60%	65.16%
		data.mat (Neutral & Expression) (K = 1)	❖ 20 images (10 from each class)	❖ 380 images (190 from each class)	❖ 80%	80%
		data.mat (Neutral & Expression) (K = 3)	❖ 20 images (10 from each class)	❖ 380 images (190 from each class)	❖ 80%	80%
		illumination.mat (K = 1)	❖ 16-21 ❖ 1, 4, 7, 14, 19, 21 ❖ 2, 5, 7, 12, 20	❖ 1-15 ❖ Remaining 15 ❖ Remaining 16	❖ 100% ❖ 95.34% ❖ 99.41%	98.25%
		pose.mat (K = 1)	❖ 8-13 ❖ 3, 6, 10, 13 ❖ 1, 5, 9	❖ 1-7 ❖ Remaining 9 ❖ Remaining 10	❖ 69.61% ❖ 80.15% ❖ 86.28%	78.68%
E.	**PCA KNN**	data.mat (K = 1) (C=200)	❖ 3 ❖ 2 ❖ 1	❖ 1, 2 ❖ 1, 3 ❖ 2, 3	❖ 58.5% ❖ 64.5% ❖ 55%	59.33%
		data.mat (Neutral & Expression) (K = 1)	❖ 140 images (70 from each class)	❖ 260 images (130 from each class)	❖ 55.71%	55.71%

		data.mat (Neutral & Expression) (K = 3)	❖ 140 images (70 from each class)	❖ 260 images (130 from each class)	❖ 43.57%	43.57%
		illumination.mat (K = 1)	❖ > 15 ❖ 2, 3, 6, 10, 11, 19, 21 ❖ 1, 7, 14, 19	❖ 1-15 ❖ Remaining 14 ❖ Remaining 17	❖ 99.5% ❖ 99.37% ❖ 99.27%	99.38%
		pose.mat (K = 1)	❖ > 9 ❖ 2, 4, 8, 10, 13 ❖ 1, 3, 9, 11	❖ 1-9 ❖ Remaining 8 ❖ Remaining 9	❖ 70.22% ❖ 75.29% ❖ 83.82%	76.44%
F.	**LDA KNN**	data.mat (K = 1) (C=200) (alpha: 0.05)	❖ 3 ❖ 2 ❖ 1	❖ 1, 2 ❖ 1, 3 ❖ 2, 3	❖ 57.5% ❖ 77.5% ❖ 88%	74.33%
		data.mat (Neutral & Expression) (K = 1)	❖ 120 images (60 from each class)	❖ 280 images (140 from each class)	❖ 60.83%	60.83%
		data.mat (Neutral & Expression) (K = 3)	❖ 120 images (60 from each class)	❖ 280 images (140 from each class)	❖ 41%	41%
		illumination.mat (K = 1) (alpha: 0.05)	❖ > 15 ❖ 1, 4, 9, 12, 18, 20 ❖ 2, 3, 8, 15, 16, 19, 21	❖ 1-15 ❖ Remaining 15 ❖ Remaining 14	❖ 99.76% ❖ 99.76% ❖ 98.32%	99.28%
		pose.mat (K = 1) (alpha: 0.05)	❖ > 9 ❖ > 8 ❖ 1, 3, 7, 10, 13	❖ 1-9 ❖ 1-8 ❖ Remaining 8	❖ 75.37% ❖ 77.65% ❖ 72.35%	75.12%

*** Table 2. KNN Results (PCA-KNN and LDA-KNN) ***

IV. Analysis:

Dataset	Bayesian	PCA-Bayesian	LDA-Bayesian	KNN	PCA-KNN	LDA-KNN	Average (%)
data.mat	68.13	60	74.67	75.05	52.87	58.72	**65%**
illumination.mat	100	100	100	98.25	99.38	99.28	**99.49%**
pose.mat	80.1	82.7	80.2	78.68	76.44	75.12	**78.87%**
Average (%)	**82.74%**	**80.9%**	**84.95%**	**83.99%**	**76.23%**	**77.71%**	**-**

Table 3. Shows the overall accuracy for each dataset and classifier

❖ **data.mat** (Refer Table 1 and 2)

➢ *Bayesian Classification* (ML-Bayesian, LDA-Bayesian or PCA-Bayesian), with training index as 2(expression), 3(illumination) and 1(neutral) as testing, we observe *greater accuracy* than any other combination.

As value of "α" (scaling factor used to prevent singularity) decreases the accuracy increases (*You can see in LDA-Bayesian, when* "$\alpha = 0.05$"*, the accuracy jumps to 88.5%).*

➢ *KNN Classification* (KNN and PCA-KNN), with "$K > 1$", causes smoothening and hence the accuracy increases for some combinations but decreases for the others.

For "$K = 1$", the accuracy is best achieved when index 1, 3 is used for training and 2 is used for testing. The value of "$\alpha = 0.05$", produces *maximum accuracy.*

For "$K = 2$" and "$K = 3$", the accuracy is best achieved when index 1, 2 is used for training and 3 is used for testing.

In *LDA-KNN*, the highest accuracy is obtained for "$K = 1$", "$K = 2$" and "$K = 3$", when index 2, 3 is used for training and 1 is used for testing.

➢ Thus, the illumination (3rd component in the data.mat dataset), if used in training will produce higher accuracy. This is because a brighter and illuminated image produces better results as we know.

❖ **illumination.mat** (Refer Table 1 and 2)

➢ *Bayesian Classification* (ML-Bayesian, LDA-Bayesian or PCA-Bayesian), produces 100% accuracy with or without any dependency on the dataset chosen for training or testing.

- *KNN Classification* (KNN, PCA-KNN, LDA-KNN), produces 98-99% accuracy when "$K = 1$" i.e. 1 Nearest Neighbor. But when "$K > 1$", the accuracy varies from 68-95% and causes smoothening effect.

- Most of the results/accuracies obtained lie between 98-100%. Which proves the fact that a well illuminated/brighter image will produce accurate results and hence classification.

- **pose.mat** (Refer Table 1 and 2)

  - *Bayesian Classification* (ML-Bayesian, LDA-Bayesian or PCA-Bayesian), with higher training set and smaller testing set, produces *greater accuracy* in any combination.

  - *KNN Classification* (LDA-KNN, PCA-KNN), the results are best when "$K = 1$", but smoothens/decreases by a bit as "$K > 1$".

V. Some Conclusions:

- As the size of the training data increases, the accuracy increases steadily and then becomes stable/constant.
- As the value of K increases (in KNN, PCA-KNN and LDA-KNN), the classification smoothens.
- In data.mat using illumination variations in training with either expressions or neutral images produces great accuracy; proving the point that an illuminated image produces better accuracy.
- This can also be supported with the fact that illumination.mat produces an accuracy of almost 98-100% for almost any classifier.
- As you increase the % of training data as an input to the classifier, the accuracy increases.
  - For e.g. for pose.mat, when you increase the % of training data, the accuracy increases up to a threshold and then it drops (for Bayesian and KNN).
  - For illumination.mat, the accuracy increases for almost all classifiers when we use 75+% of data for training.
- Finally, from **Table 3**, we can see that **"Bayesian"**, classifies the best.

- Thus, we conclude:
  - In this project we implemented the **Bayesian** and **KNN** classifiers with *dimensionality reduction* **(PCA and LDA)** techniques.
  - We see from *"Table 3"* that **LDA-Bayesian** produces the best results.
  - Higher training data increases the accuracy.
  - Greater variance in training dataset results in higher accuracy.
  - Illuminated images are best trained and classified.

****