GROUP 4
PROJECT REPORT

CH5019
MATHEMATICAL FOUNDATIONS OF DATA SCIENCE

BE18B004 - Kailash Lakshmikanth
BE18B011 - S.R. Nikitha
ED18B036 - Thammineni Harshith Reddy
ED18B034 - Swathi Veerabathraswamy

QUESTION 1

Face recognition is a kind of classification problem, which has two major parts:

- A database consisting of multiple images is used to generate a representative image for each subject.
- When any new image is to be classified into one of the given subjects, it is compared which each representative image, and the one with the least deviation is picked.

IMAGES AS A DATA MATRIX

Each of greyscale images given can be considered as a matrix of pixels, with each pixel taking a value from 0 to 1 based on the intensity of colour, 0 being white and 1 being black. Each image is now thus a 64×64 matrix of values.

This matrix can then be transformed into a column vector, which represents a data point. For every subject, we have 10 such images, forming ten different column vectors. These are combined to form the data matrix for each subject. This is shown in Fig.1.

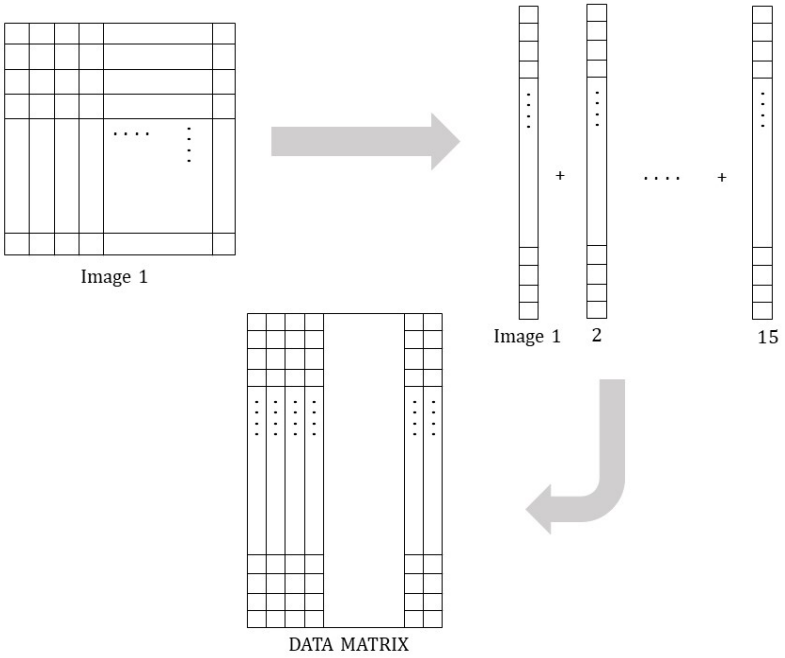

Figure 1: Representation of images as a data matrix

PART 1: FINDING THE REPRESENTATIVE IMAGE

Two approaches were tried on the data matrix to find the representative image, in order to find the most accurate representation for each subject.

Using the mean as a representative image

The **mean of all column vectors** in the data matrix for a particular subject was taken as the representative column vector. This mean column vector was then converted back to a 64×64 matrix, which was then converted into an image. Such images were created for each of the 15 subjects, named **M1.pgm** to **M15.pgm**.

The attached program **MEAN.m** was used for this purpose. The representative images obtained for each of the subjects are shown in Fig.2.

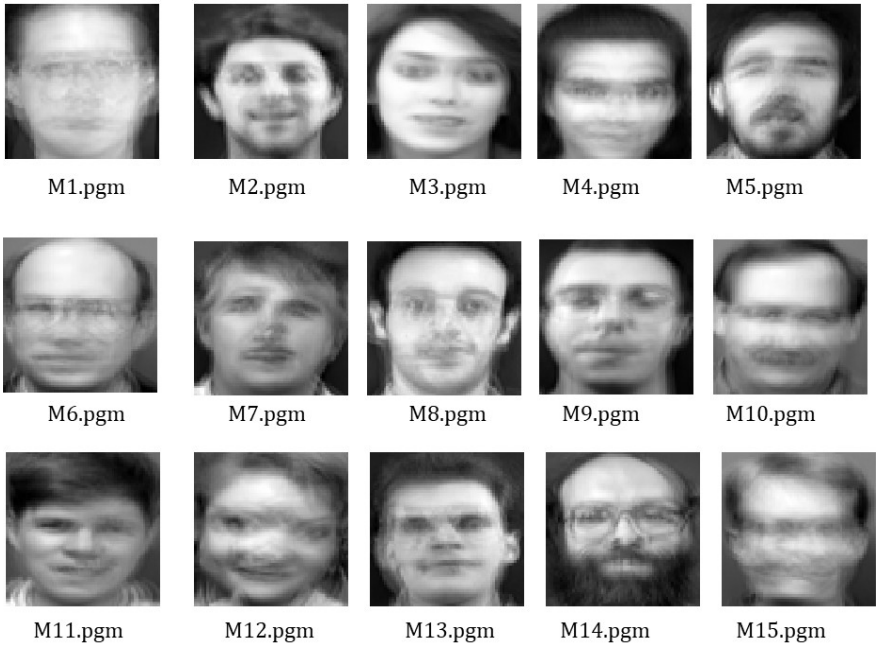

Figure 2: Mean image for each of the 15 subjects

Using the principal eigenvector as a representative image

THEORY

A data matrix A can be decomposed using Singular Value Decomposition (SVD) as:

$$A = USV^{-1}$$

Where:

the columns of U are the eigenvectors of matrix A.

the diagonal elements of S are the eigenvalues of A in decreasing order.

Thus the principal eigenvector is the first column of U. This eigenvector can be converted to a matrix and subsequently an image. This idea is used to find the principal **Eigenface**, the most accurate representation of the given images.

IMPLEMENTATION

Each column of the data matrix is first reduced by the mean of the matrix, which gives us the new data matrix. SVD is then applied on the covariance of this matrix, using the **svd()** function in MATLAB.

Since only one representative image can be stored for each subject, only the first eigenvector is considered. The weight of the eigenvector is calculated by its norm, and this eigenvector, scaled by its weight, is added to the mean, giving us the vector for the representative image. This vector is reshaped into a matrix, which is then converted to an image. The program **EIGENFACE.m** was used, which generated the images **R1.pgm** to **R15.pgm** shown in Fig. 3.

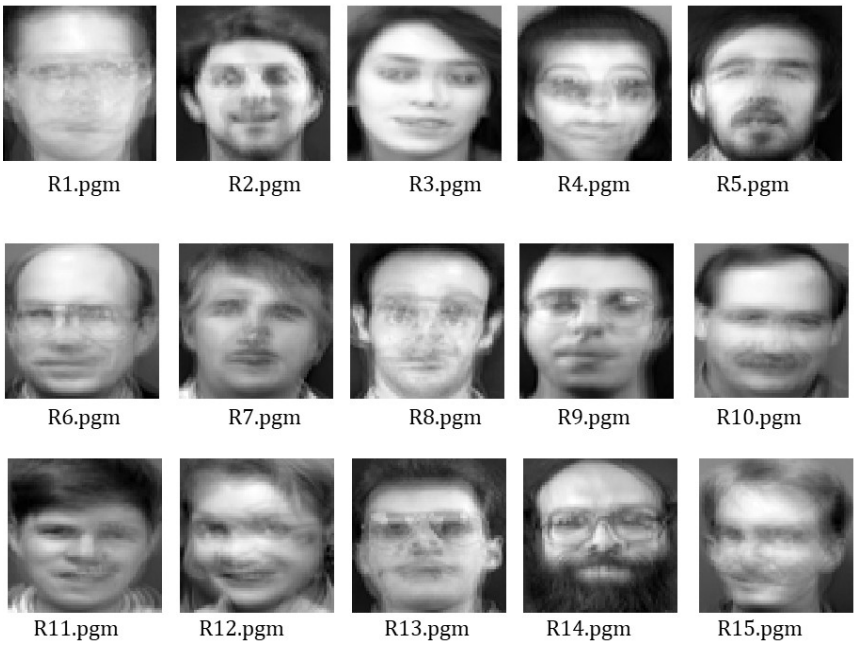

Figure 3: Mean image for each of the 15 subjects

PART 1: CLASSIFYING IMAGES AS ONE OF THE FIFTEEN SUBJECTS

The closeness of two given vectors can be found by the **norm of their difference**. The more closer the vectors, the lesser the norm between them. This idea is used to classify the images.

A matrix is formed, consisting of all the representative images as its columns. The image to be classified is also converted into a vector as explained above. For each of the 15 subjects the norm of the difference between the test image and representative image is found, using the function **norm()** in MATLAB. The representative image closest to the test image will give the least norm of difference. Thus the image can be classified.

The program **TESTING.m** was used to classify each of the 150 images as explained above, and to calculate the number of correct classifications.

RESULTS

For the two approaches followed, the results were as follows:

MEAN IMAGE

141 out of 150 images were classified correctly (Accuracy – 94%)

EIGENFACE

137 out of 150 images were classified correctly (Accuracy – 91.3%)

FURTHER INSIGHTS

Adding more than one scaled eigenvectors along with the mean vector was also considered and tested, but it gave a lesser accuracy, possibly due to the more detailed representative image obtained.

QUESTION 2

This question required us to train a logistic regression model to predict if a reactor would pass or fail based on the given dataset, which contained 1000 variables, each with 5 features and a test variable (pass or fail).

DESCRIBING THE STATISTICS OF THE DATA

Using python's **describe()** function, the numerical descriptive statistics of the data was obtained. There were no missing values.

	Temperature	Pressure	Feed Flow rate	Coolant Flow rate	Inlet reactant concentration
count	1000.00000	1000.000000	1000.000000	1000.000000	1000.000000
mean	546.76643	25.493270	125.029060	2295.797770	0.302692
std	86.85878	14.252407	43.508159	763.680625	0.116062
min	400.31000	1.060000	50.030000	1002.530000	0.100300
25%	469.73500	12.725000	88.587500	1635.682500	0.199075
50%	545.80000	25.375000	124.590000	2268.710000	0.308850
75%	618.87750	37.820000	162.562500	2983.692500	0.401625
max	699.87000	49.890000	199.960000	3595.620000	0.499600

Figure 4: Various central values in the dataset

Using sklearn's **LabelEncoder()**, Pass was encoded as 1 and Fail as 0. Using sklearn's **train_test_split**, the dataset was split into a training set (70%-700 samples) and a test set (30%-300 samples).

From our domain knowledge, the features seem to be independent as these can be independently controlled in a reactor without having much effect on the other reactor variables.

	Temperature	Pressure	Feed Flow rate	Coolant Flow rate	Inlet reactant concentration	Test
0	406.86	17.66	121.83	2109.20	0.1033	1
1	693.39	24.66	133.18	3138.96	0.3785	1
2	523.10	23.23	146.55	1058.24	0.4799	0
3	612.86	40.97	94.44	1325.12	0.3147	0
4	500.28	37.44	185.48	2474.51	0.2284	1

Figure 5: Encoded dataset

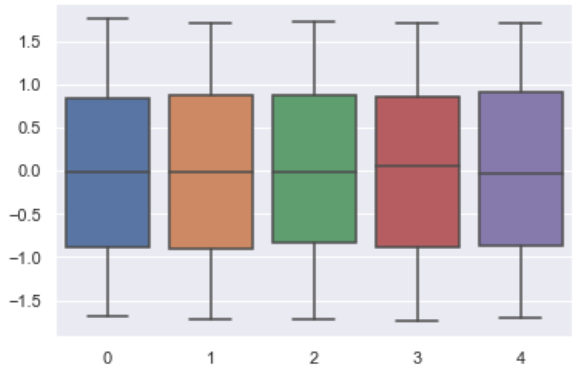

Figure 6: Standardized feature values

The feature values were standardised using **StandardScaler** to get a compact readable boxplot. The x-axis shows the features and y-axis has the corresponding standardised values. From boxplots of individual features, the median, 1st quartile, 3rd quartile, maximum and minimum values can be obtained easily. From the Fig.6, we can see that the whiskers are of almost equal length (and median is in the middle) for all features, which is characteristic of a normal distribution.

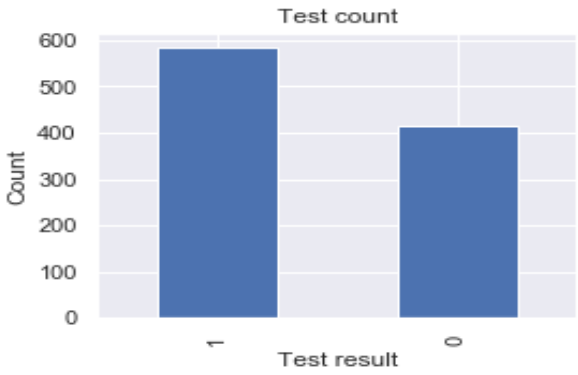

Figure 7: Test counts of pass and fail cases

From the graph in Fig.7 it is clear that the test results **weren't too biased**. Hence there was no need to oversample or undersample the data.

Scatterplots were drawn to visualize the dependence of each of the features on the test case, the results of which are shown in Fig.8 to Fig.12.

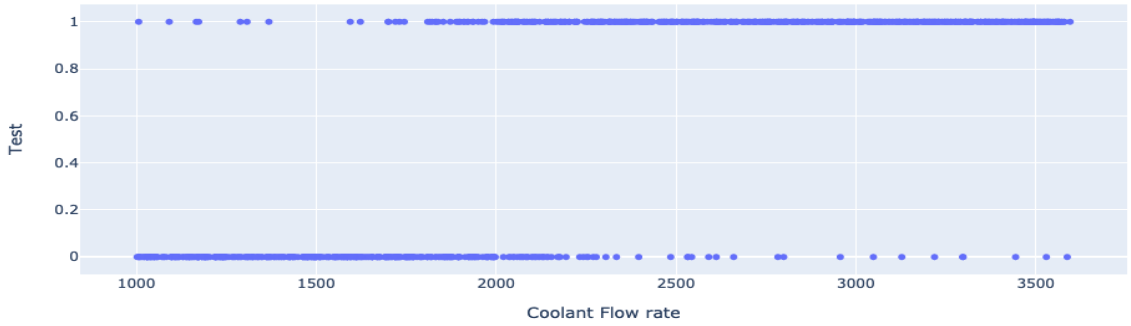

Figure 8: Dependence of test results on coolant flow rate

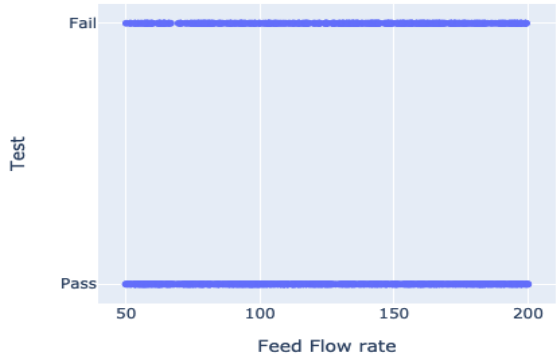

Figure 9: Dependence of test results on feed flow rate

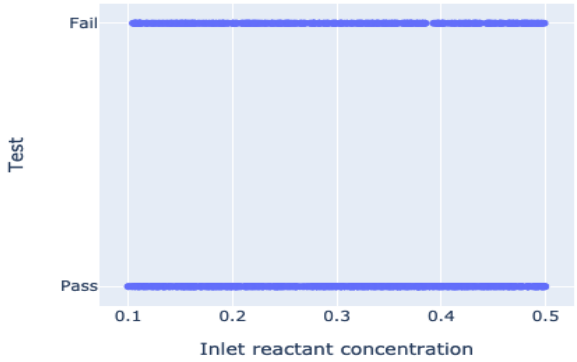

Figure 11: Dependence of test results on reactant concentration

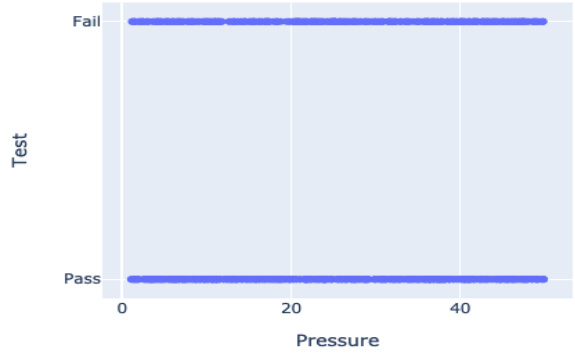

Figure 10: Dependence of test results on pressure

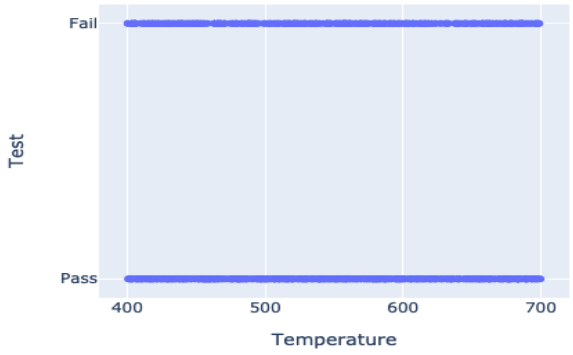

Figure 12: Dependence of test results on temperature

As ahown above, the scatterplot between test and coolant flow rate showed that with increasing coolant flow rate the reactor was more likely to 'PASS'. The scatterplot between the other independent variables and test results showed no such interpretable patterns.

In order to further visualise the data, the top 2 principal components of the features 'Feed flow

rate', 'pressure', 'temperature' and 'inlet reactant concentration' were extracted. A 3D scatterplot with axes PC1 (principal component 1), PC2 (principal component 2) and coolant flow rate was plotted, as shown in Fig.**??** and Fig.**??**. In these plots, yellow dots denote the pass cases and blue dots denote the fail cases.)

From this 3D plot, we can clearly see that information from all features helped in segregation of classes. We can easily visualise a good decision boundary between the Pass and Fail classes. Since all features were assumed to be independent, we initially trained the logistic regression model with all these features, which gave a high accuracy. Thus, we chose to keep all the features and did not want to do feature selection.

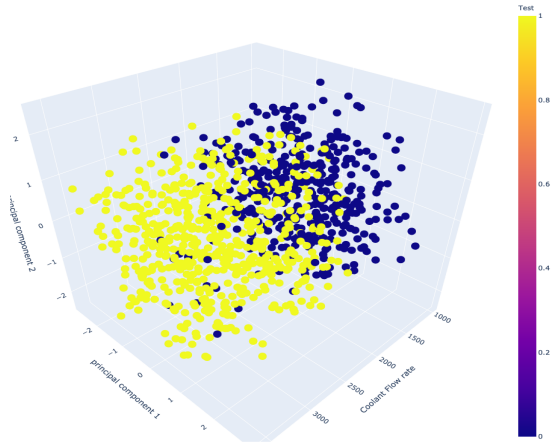

Figure 13: 3D Scatterplot

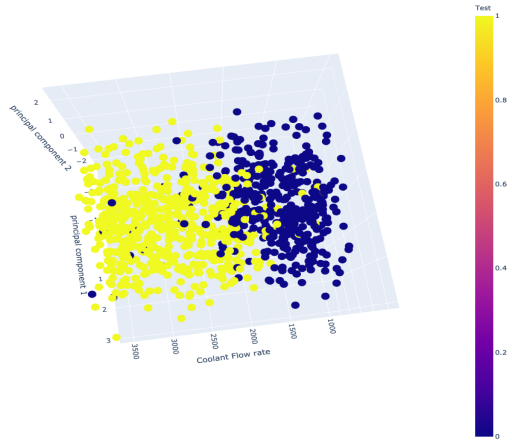

Figure 14: 3D Scatterplot

LOGISTIC REGRESSION

For training the model, the feature values were initially scaled using **StandardScaler**. We thought this could help in updating the parameters with equal importance as some features have huge ranges and may tend to dominate the hypothesis function. Also too many epochs may overfit the model, it was necessary that the loss function converges faster.

A class named **Logistic Regression** was created with

A parameter initialising function:
that initialised learning rate as 0.01, as seen in many logistic regression models and an epochs parameter

Sigmoid function:
that took the hypothesis function as a parameter and sandwiched it between 0 and 1

Fit function:
that took the feature values, test values of the train sample and initial coefficient matrix/theta as

parameters. This is the main function of the training process where loss function is calculated and gradient descent is done to improve the parameter values until the loss function converges within the initialised number of epochs.

pred_prob function:
that took the test sample feature values as parameter and returned the value using coefficient/theta matrix from the trained model.

Predict function:
this took the test values of the testing sample set and compared it with the values from the pred_prob function. The output was the accuracy of the trained model.

It was important that the loss function reaches the global minima. We looped the training with various initial thetas to see which initial theta gives least loss function. We initially started off with high learning rates and different initialisations to get a rough idea of the least possible loss function value (mostly around 0.26968). Then we took low learning rates and looped the training with various initial thetas and large number of epochs expecting it to converge to the global minimum. We got a loss value of 0.26968033865775054 over a large range of initial thetas. This value was the lowest for us based on the parameters we chose, as shown in Fig.15.

```
Initialtheta: -10.0 , Loss: 0.26968033865775054
Initialtheta: -9.0 , Loss: 0.26968033865775054
Initialtheta: -8.0 , Loss: 0.26968033865775054
Initialtheta: -7.0 , Loss: 0.26968033865775054
Initialtheta: -6.0 , Loss: 0.26968033865775054
Initialtheta: -5.0 , Loss: 0.26968033865775054
Initialtheta: -4.0 , Loss: 0.2696803386577507
Initialtheta: -3.0 , Loss: 0.2696803386577507
Initialtheta: -2.0 , Loss: 0.2696803386577507
Initialtheta: -1.0 , Loss: 0.2696803386577507
Initialtheta: 0.0 , Loss: 0.2696803386577507
Initialtheta: 1.0 , Loss: 0.2696803386577507
Initialtheta: 2.0 , Loss: 0.2696803386577507
Initialtheta: 3.0 , Loss: 0.26968033865775054
Initialtheta: 4.0 , Loss: 0.26968033865775054
Initialtheta: 5.0 , Loss: 0.26968033865775054
Initialtheta: 6.0 , Loss: 0.26968033865775054
Initialtheta: 7.0 , Loss: 0.26968033865775054
```

Figure 15: Iterations carried out to find minima

So choosing an initial theta of 5.0, the loss function stabilizes over many epochs to get optimal theta, as shown in Fig.16.

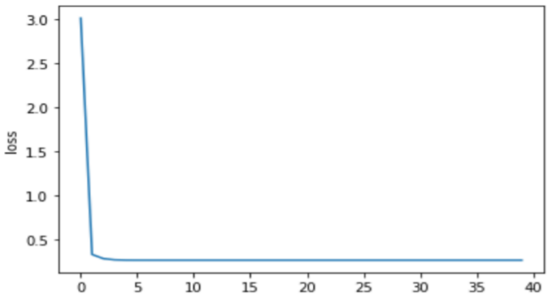

Figure 16: Loss stabilises to get optimum theta

PERFORMANCE ANALYSIS

```
model.theta
array([ 1.12451821e+00, -1.51350179e-01, -4.38569708e-01, -6.50205036e-01,
        3.50525002e+00, -3.30641786e-03])
```

Figure 17: Model Theta

```
Confusion matrix:
 [[112   6]
 [  9 173]]
F1 score: 0.9584487534626039
```

Figure 18: Confusion Matrix and F1

Fig.17 shows the optimised parameters obtained, which were very similar for different initialisations. The first is the intercept and the rest are the feature parameters for initialisation of 5.0. The accuracy was around 0.95.

Fig.18 shows the **confusion matrix** obtained, along with a satisfactory **F1 score** of around 0.9521.

QUESTION 3

Coronaviruses are a large family of viruses which may cause illness in animals or humans. The **Novel Corona Virus**, or **Covid-19**, is seeing an increasing number cases by the day. For all the analysis below, we have used updated data sets from *www.kaggle.com/sudalairajkumar/covid19-in-india*.

1. MOST AFFECTED AGE GROUP

In order to find the most affected age group, we used the dataset **AgeGroupDetails.csv**. This dataset, imported into MATLAB using the **readtable()** function, directly gave us the number of cases in each age group, which was plotted in a bar graph to find the highest number of cases.

In addition to this, for each age group, data from **IndividualDetails.csv** was used to extract the number of death cases and recovered cases in each age group. This was used to calculate the **fatality**, as the number of death cases per number of recovered cases. In this way, we coloured the graphs from white (least fatal) to black (most fatal). This is done using the code **AGE_GROUPS.m**.

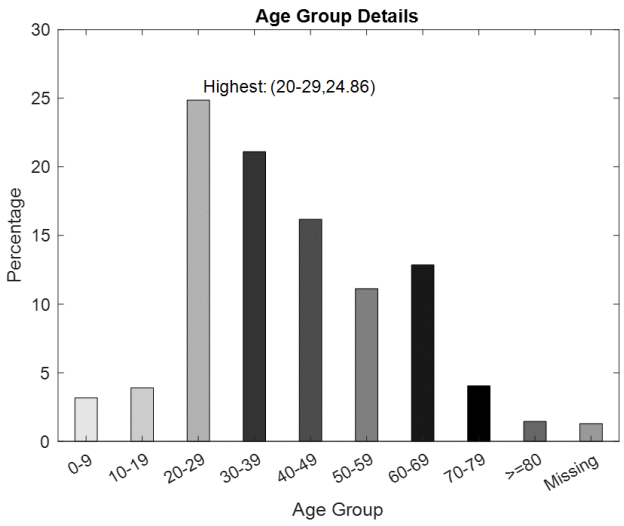

Figure 19: Most affected age group

RESULT

From Fig.19, we can see that the most affected age group is **20-29**, but the virus is most fatal for the age group **70-79**.

2. CASES PER DAY COUNTRY-WISE AND STATE-WISE

From the dataset **covid_19_india.csv**, the number of cases (confirmed,recovered, and deaths) everyday in each state was counted. The sum of these cases was also taken, as the number of cases in the entire country.

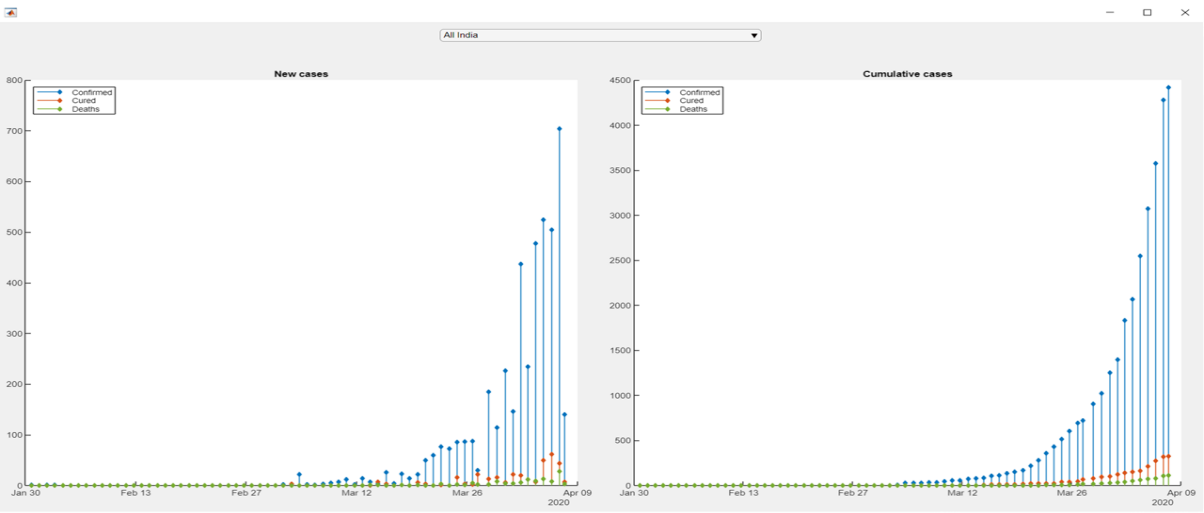

Figure 20: Country wise number of cases

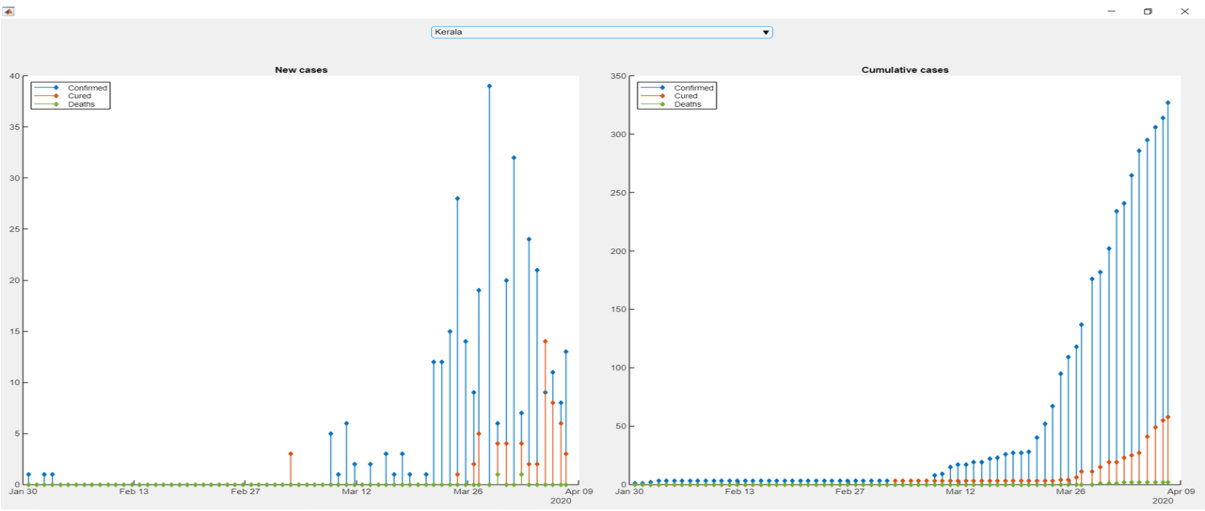

Figure 21: State wise number of cases

Since the dataset contains the **cumulative cases** for each day, a difference operation is carried out to find the **new cases** for each day. These are plotted separately for the entire country and for each state. Using a dropdown menu on top, the state for which we wish to obtain data can be picked. The output of the code **CASES_PER_DAY.m** is shown in Fig.20 for the country and Fig.21 for one random state, Kerala. Similar plots for each state can be obtained by running the code **CASES_PER_DAY.m**.

3. INTENSITY OF VIRUS SPREAD

The code **INTENSITY.m** used data from the dataset **covid_19_india.csv** to find the number of cases in each state. Along with this, data from **Population_india_census2011.csv** is used to determine the area of each state. The intensity of the virus spread in each state is thus calculated and plotted, as shown in Fig.22.

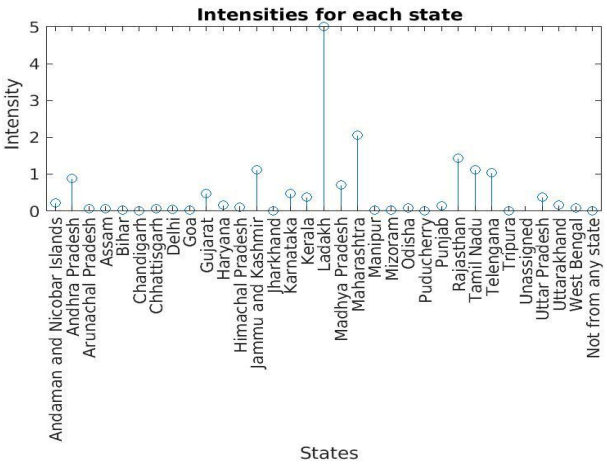

Figure 22: Intensity of spread in states and union territories

4. HOTSPOTS

The code **HOTSPOTS.m** uses data from the dataset **covid_19_india.csv** to find the number of cases in each district as on 10.04.2020. Any district with more than 10 cases is considered a **hotspot**. A list of hotspots along with their state is printed, which is stored in **Output-3.4.txt**. This is shown in Fig.23.

To visualize this, **Google Maps Javascript API** was used, using JSFiddle. The code is also saved in the .zip file as **JSMap.txt**. The output of this is shown in Fig.24.

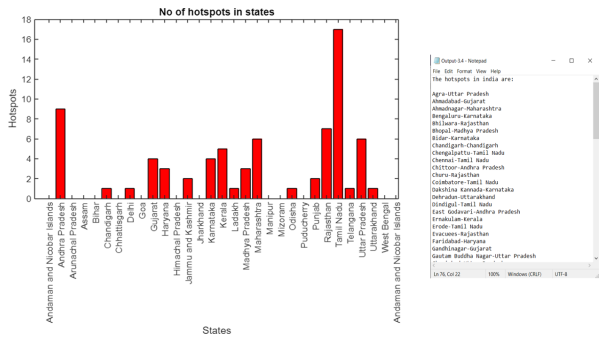

Figure 23: Hotspots

Figure 24: Hotspots visualisation

The map can also be zoomed in to clearly visualise the hotspots, as shown below.

Figure 25: Hotspots in North India

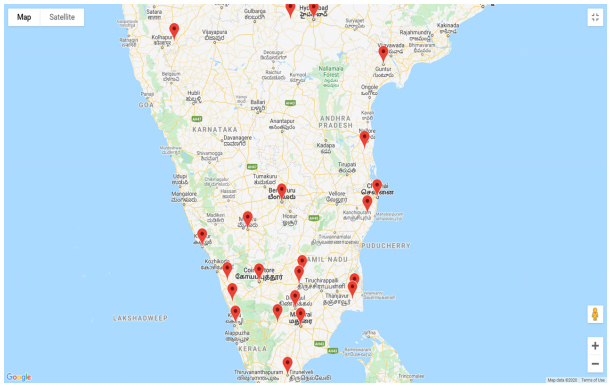

Figure 26: Hotspots in South India

5. CHANGE IN NUMBER OF HOTSPOTS

The function from Question 4 is used to calculate the number of hotspots for four different dates (20/03/2020, 27/03/2020, 03/04/2020, and 10/04/2020), each varying by 1 week. The change in number of hotspots is then calculated for each state, which is plotted separately for each week, as shown in Fig.27. The state with the maximum increase and decrease in hotspots is also printed, using the code **CHANGE_HOTSPOTS.m**.

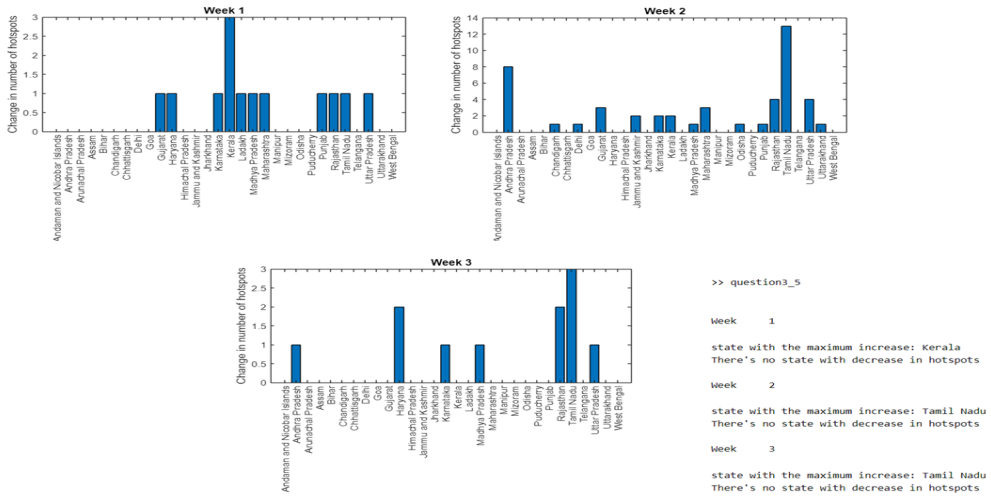

Figure 27: Change in number of hotspots

6. CLASSIFYING PATIENTS INTO PRIMARY, SECONDARY, AND TERTIARY CASES

In the dataset **IndividualDetails.csv**, the notes column had information about how the virus was contracted. The dataset was studied and keywords for each type of case was extracted (only for the top five states). The code **CASE_TYPES.m** was used for this purpose.

Secondary cases had atleast one of the following keywords - Patient numbers (starting with **P1 - P9**), the word **Delhi**, **contact**, **tourist**, **doctor**, **hospital**, **contracted from**, **tenant**, **visited**, **relative**, **local transmission**.

Primary cases had atleast one of the following keywords - **travelled from**, **travelled to**, **international history**, **travel history to**, **travel history of**, **travel history from**, **foreign travel history**. In addition, some of the Primary Cases had a **nationality** that was not Indian, India, or blank.

Any case which did not fall into the above categories is a **Tertiary Case**.

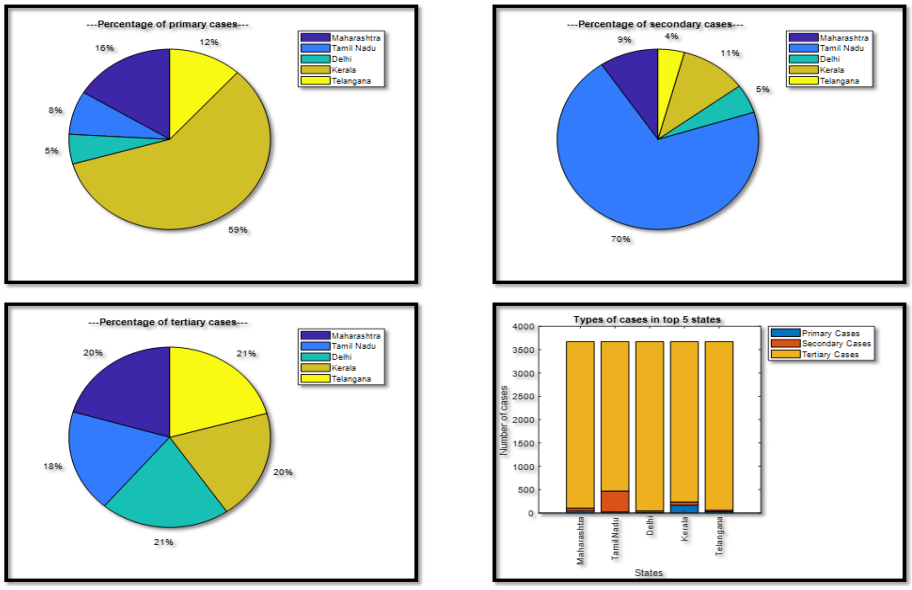

Figure 28: Distribution of cases of each type

7. NUMBER OF ADDITIONAL LABS REQUIRED

To find the number of additional labs required throughout the country, the dataset **ICMRTestingDetails.csv** was used. The number of total tests on 11.04.2020, and the number of positive cases are extracted from that. From that, the number of anticipated positive cases on 20.04.2020 was calculated. Assuming that the fraction of tested cases which are positive increases almost linearly (as is the case with most tests), the number of tests required are calculated using **ADDITIONAL_LABS_INDIA.m**, and subsequently the number of labs required additionally. By this method, the number of additional

labs required will be around **1537**.

Additionally, assuming that because of the travel restrictions, people cannot travel across states to get tested. Thus an additional dataset **IMCRTestingLabs.csv** was downloaded from *https://www.kaggle.com/sudalairajkumar/covid19-in-india*, which contains information about each Testing Lab. This information, along with the information in **covid_19_india.csv** is used to find out how many additional testing centers are required in each state, using the code **ADDITIONAL_LABS_STATES.csv**. The output is shown in Fig.29.

```
COMMAND WINDOW
      1537

>>

  ==Labs needed in each state====
    "Andaman and Nicobar Islands"    "3"
    "Andhra Pradesh"                 "54"
    "Arunachal Pradesh"              "0"
    "Assam"                          "0"
    "Bihar"                          "5"
    "Chandigarh"                     "4"
    "Chhattisgarh"                   "1"
    "Delhi"                          "122"
    "Goa"                            "2"
    "Gujarat"                        "34"
    "Haryana"                        "25"
    "Himachal Pradesh"               "1"
    "Jammu and Kashmir"              "30"
    "Jharkhand"                      "0"
    "Karnataka"                      "29"
    "Kerala"                         "114"
    "Ladakh"                         "8"
    "Madhya Pradesh"                 "40"
    "Maharashtra"                    "191"
    "Manipur"                        "0"
    "Mizoram"                        "0"
    "Odisha"                         "0"
    "Puducherry"                     "1"
    "Punjab"                         "24"
    "Rajasthan"                      "72"
    "Tamil Nadu"                     "117"
    "Telengana"                      "0"
    "Tripura"                        "0"
    "Unassigned"                     "0"
    "Uttar Pradesh"                  "69"
    "Uttarakhand"                    "5"
    "West Bengal"                    "12"
```

Figure 29: Output of question 3.7

8. FLATTENING THE CURVE

The number of cases were plotted with the days since the the first case, using **CURVE_FITTING.m**. As can be seen in Fig.30, the curve goes from an initial exponential fit to almost linear/quadratic towards the end. This will further move on to become constant. This is the popular notion of **flattening the curve**.

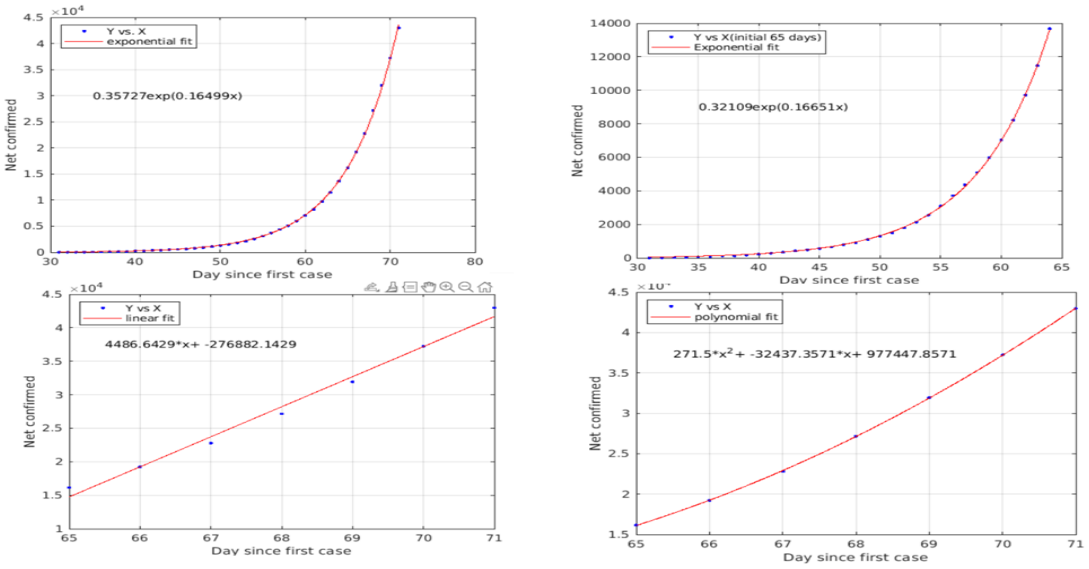

Figure 30: Curves fit to different parts of the plot

9. EFFECTIVENESS OF THE LOCKDOWN

Without the lockdown, the trend would have continued to be exponential as in the inital days. But the effects of the 21 day lockdown were seen in the last few days relevant to the data, as the curve gradually progressed to become quadratic. An **exponential growth** would have caused the number of cases to rise to **43707**, but the actual number of cases was **42980**. This difference (**727** cases) was most likely caused by the lockdown, seeing how the spread of the virus is otherwise hard to contain.

From the curves in Question 3.8, we also concluded that if the growth was perfectly **quadratic** in the last one week, the number of cases would have been **43027**, making a difference of just **680** cases, which means the growth is less than a quadratic extrapolation.

On the other hand, if the lockdown had been more effective, the curve could have become **linear** in the last week, leading to 41670 cases, with a difference of more than **2000** cases.

References:

Question 1

-https://www.cs.cmu.edu/~pmuthuku/mlsp_page/assignments/assignment2_hints.html

-http://jmcspot.com/Eigenface/

Question 2

-Building a Logistic Regression in Python - Animesh Agarwal

Question 3

-www.kaggle.com/sudalairajkumar/covid19-in-india