Term Project Report

CH 5019 Mathematical Foundations of Data Science

May 10, 2020

Authored and submitted by **Group 12**

ME17B039 Vishwajit Hegde, **ME17B050** Ojas Jain, **ME17B084** Nishant Prabhu, **ME17B134** Bibhabasu Das, **ME17B162** Saarthak Marathe

---

Contents

---

1 Face matching by Singular Value Decomposition

1.1 Overview

In this problem, we were provided several images of faces of different subjects (grayscale). Our objective was to develop a face matching system where all images of a subject are collapsed into one representative image, capturing as much information from each image as possible. In the end, the system should be able to perform classification by judging the L_2 (Frobenius) distance between a test image and the set of representative images.

1.2 Methodology

Each image can be treated as a 2D matrix with each element representing a pixel and its scalar value denoting intensity of white. To extract the most important features for each image, we employ Singular Value Decomposition (SVD) which decomposes any matrix into a summation of single-rank matrices. The eigenvalue associated with each single rank matrix denote its relevance (amount of variance captured by it). Thus, reconstructing the image with single-rank matrices having high eigenvalues can help us capture the most important features of each image. After this has been done for each image of a subject, we aggregate all reconstructed images by averaging. A detailed procedure is described below.

Algorithm 1 SVD for Face Matching

```
    Inputs: $n$ images of each subject, number of components for reconstruction ($k$)
    Outputs: Averaged reconstructed image for each subject
1:  for every subject do
2:      for every image of this subject do
3:          flat_image ← flattened image of shape (1, 4096)
4:          Stack flat_image with stack of other flattened images
5:      end for
6:      $U, S, V$ ← svd(stack)
7:      $U_k, S_k, V_k$ ← first $k$ components of $U, S, V$
8:      reconstructed_stack ← $U_k S_k V_k^T$
9:      representative_image ← $\frac{1}{n}\sum_{i=1}^{n} row_i$ of stack
10: end for
```

Frobenius norm of every test image is computed with every representative image, and the one with the lowest norm is assigned as the class of the test image.

1.3 Process particulars

- For our experiments, $k = 1$ was used. No change in performance was observed for larger values of k, indicating that higher layers more or less represented noise.
- We chose to average the reconstructed stack to obtain the representative image as this would inherently preserve all important features extracted for each example.
- It was necessary to flatten the images and generate a 2D stack as Python implementations of SVD accept only 2 dimensional data as input. Also, we are unsure if 3 dimensional SVD is possible.

1.4 Results

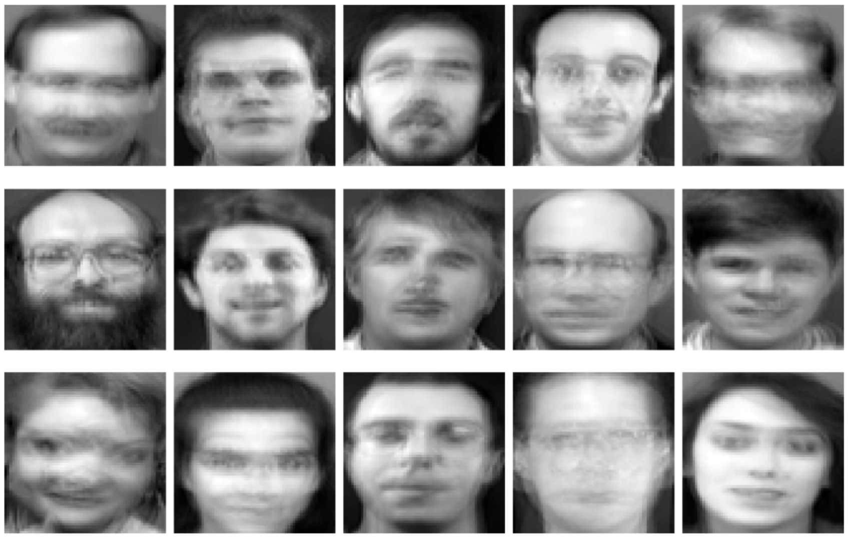


Figure 1: *Collage of representative faces obtained from the algorithm*

The distance based classifier achieved a classification accuracy of **99.33%**. The confusion matrix is shown in **Figure 2**. Only 1 misclassification has occurred.

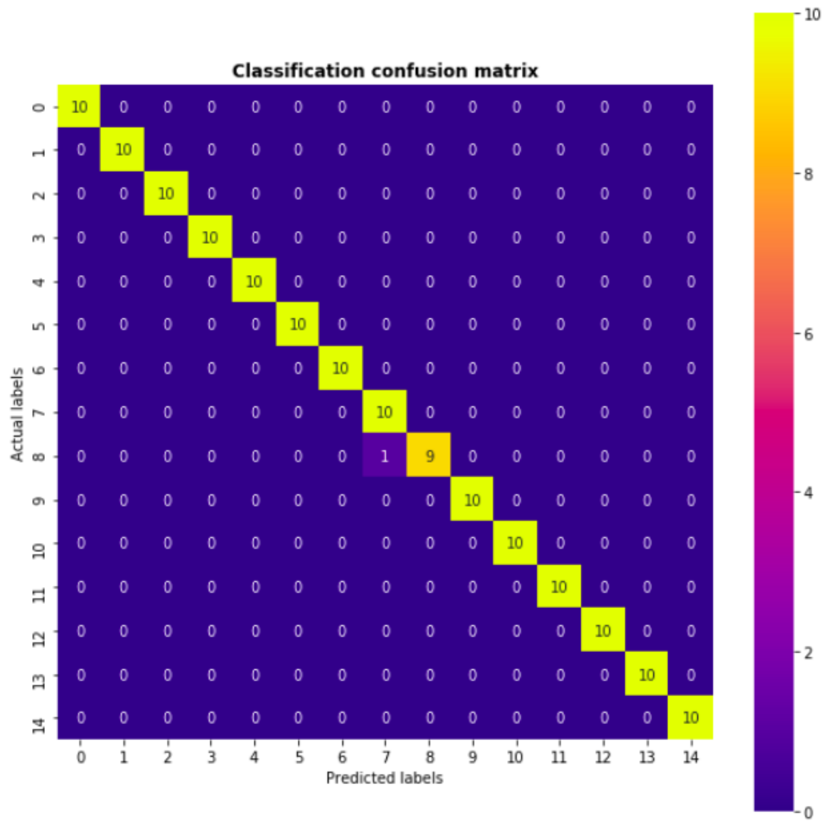


Figure 2: *Classification confusion matrix*

2 Logistic Regression

2.1 Overview

We are given operating conditions of a reactor and whether the reactor passes or fails a test for every configuration. Our objective is to implement a logistic regression model to predict the results of the test, given a configuration. Here, we present the design of the classifier, some data exploration and classification results.

2.2 Model Design

Given an input vector x, the likelihood of that vector belonging to one of two classes is given by:

$$\phi(x) = \frac{1}{1 + e^{-(w^T x + b)}}$$

Through **maximum likelihood estimation** and updates using **gradient ascent**, we aim to estimate w and b which completely define our classifier. If we represent both parameters colloquially by θ and assume that all data points are independent and identically distributed (i.i.d.), then we have ...

$$l(\theta) = \prod_{i=1}^{m} p\left(x^{(i)}|\theta\right)^{y^{(i)}} \cdot \left(1 - p\left(x^{(i)}|\theta\right)\right)^{\left(1-y^{(i)}\right)}$$

We'll maximize the log of this function instead (so log-likelihood now).

$$L(\theta) = \sum_{i=1}^{m} y^{(i)} \log p\left(x^{(i)}|\theta\right) + \left(1 - y^{(i)}\right) \log\left(1 - p\left(x^{(i)}|\theta\right)\right)$$

$$= \sum_{i=1}^{m} y^{(i)} \log \phi\left(x^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - \phi\left(x^{(i)}\right)\right)$$

This form of $L(\theta)$ is often called **binary crossentropy**. To find how L varies with w and b, differentiate it with respect to those.

$$\frac{\partial L}{\partial w} = \sum_{i=1}^{m} \frac{y^{(i)}}{\phi\left(x^{(i)}\right)} \cdot \phi\left(x^{(i)}\right)\left(1 - \phi\left(x^{(i)}\right)\right) \cdot x + \frac{1 - y^{(i)}}{1 - \phi\left(x^{(i)}\right)} \cdot \phi\left(x^{(i)}\right)\left(1 - \phi\left(x^{(i)}\right)\right) \cdot x$$

So our gradient expressions are ...

$$\frac{\partial L}{\partial w} = \sum_{i=1}^{m} \left\{ y^{(i)} \cdot \left(1 - \phi\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \cdot \phi\left(x^{(i)}\right) \right\} \cdot x$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{m} \left\{ y^{(i)} \cdot \left(1 - \phi\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \cdot \phi\left(x^{(i)}\right) \right\}$$

The updates will be performed as follows with the quantities above.

$$w := w + \alpha \frac{\partial L}{\partial w} \quad ; \quad b := b + \alpha \frac{\partial L}{\partial b}$$

More often than not, computations are more stable in minimization problems. While implementing, we have converted this into a gradient descent problem by using negative of the gradients computed above and subtracting the step value from the parameter (instead of adding) in the update equations.

2.3 Class separability

For a classifier (like logistic regression) to produce good results, it is necessary that the classes are decently well separated. We check for that in this section. From the plot shown in **Figure 3**, we can see that coolant flow rate is an important factor in determining the test result, given how different the values of this feature are for each class. For plots of other parameters, please refer to our code.

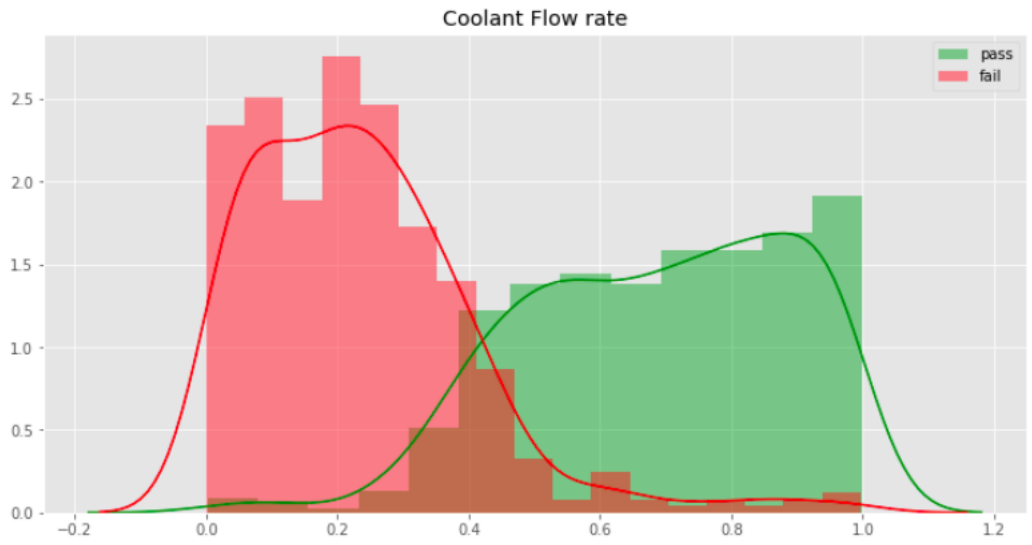
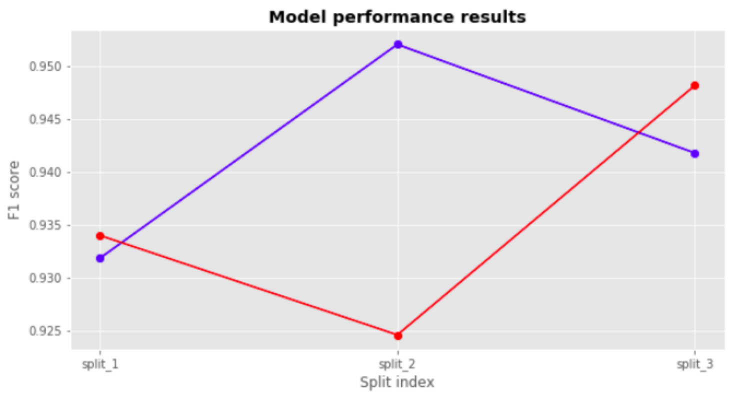


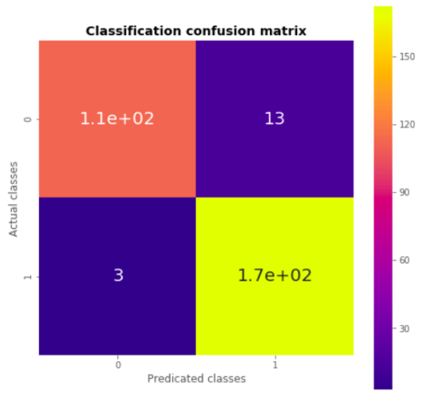
Figure 3: *Class separability by Coolant Flow Rate*

2.4 Results

Our classifier achieved an average F1 score of `0.93559`. Performance of our model over 3-fold cross-validation and confusion matrix over 70-30 split data has been shown in **Figure 4**.



(a) *3-fold cross validation F1 scores*



(b) *Classification confusion matrix*

Figure 4: *Classifier performance analysis*

Pass is encoded as 1 and fail is encoded as 0. Turns out it has trouble classifying ”fail” cases.

3 Analysis of COVID-19 in India

3.1 Overview

In this problem, we were given several datasets pertaining to the spread of SARS-CoV-2 in India. Also, we were asked to answer some questions by appropriately manipulating and visualizing data given in the datasets. For details answers of each question, please refer to our code.

3.2 Which age group is most infected?

A bar plot from `AgeGroupDetails.csv` shows the answer. People in the age group 20-29 are most affected.

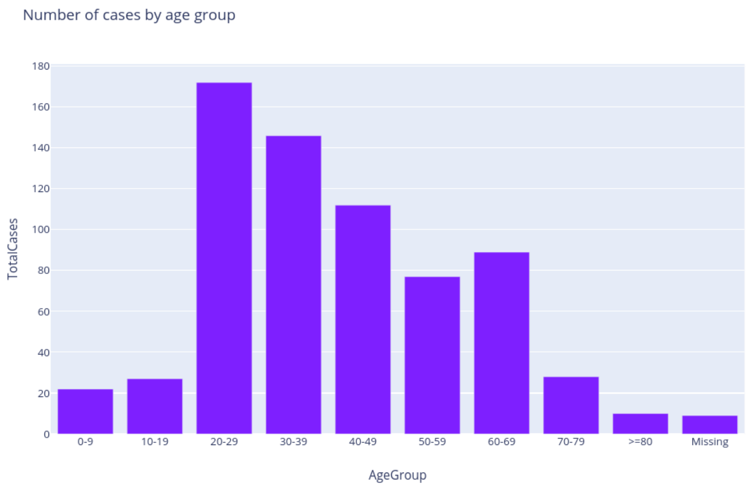


Figure 5: *Spread of COVID-19 by age group*

3.3 Country-wise and state-wise statistics

Figure 6 shows the statistics of confirmed, recovered and death cases on a country level. For state level plots, please refer to our code (too many to show here).

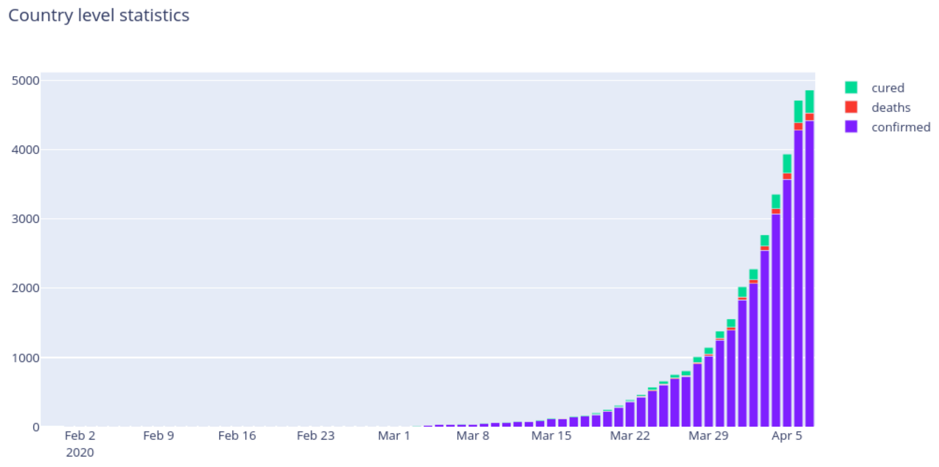


Figure 6: *Spread of COVID-19; Country level statistics*

3.4 Positive cases on state-level and intensity of spread

Figure 7 shows a combined plot of positive cases in all states by date. **Figure 8** shows the variation in intensity of spread (as defined in the problem) by date.

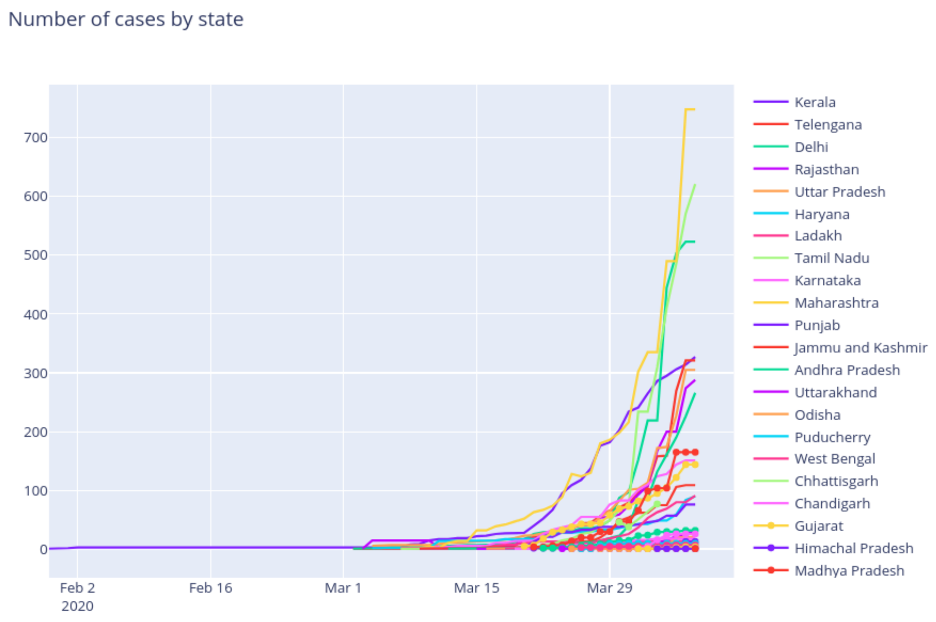


Figure 7: *State-wise positive cases by date*

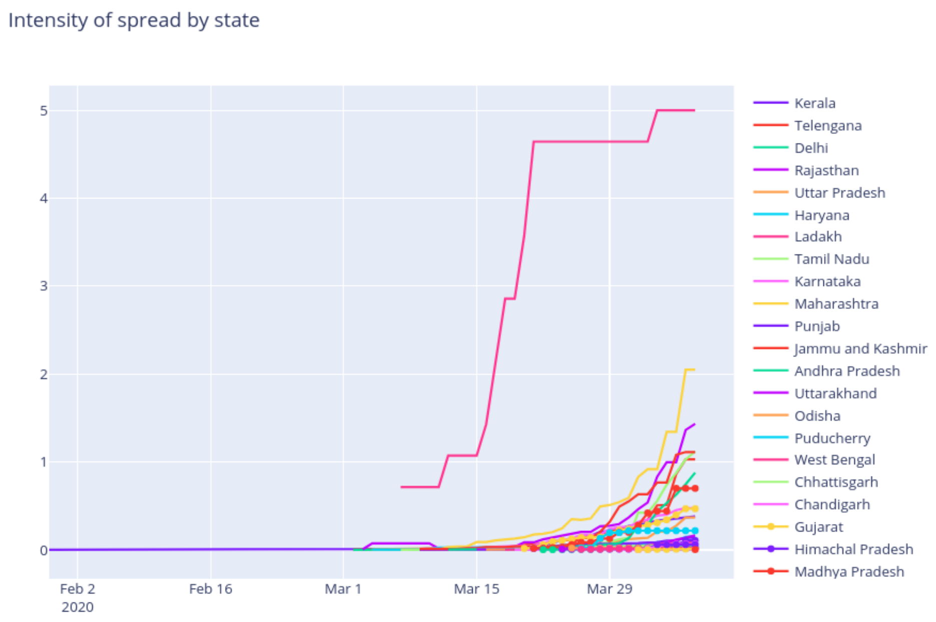


Figure 8: *State-wise intensity of spread by date*

Maharashtra leads in the number of positive cases while Ladakh leads in the intensity of spread, perhaps due to its low population density.

3.5 COVID-19 hotspots in India

Using the log data in `IndividualDetails.csv`, we extracted the number of cases identified in each city and aggregated them together. **Figure 9** shows the active hotspots in descending order of number of cases, as on 10th April, 2020.

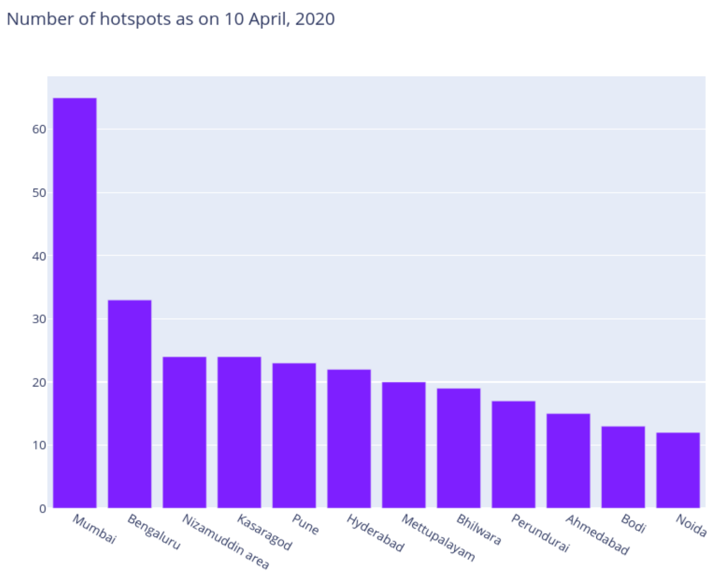


Figure 9: *COVID-19 hotspots in India, as on 10 April, 2020*

3.6 Dynamics of hotspots over 3 weeks

Using the data in `IndividualDetails.csv`, we determined the number of hotspots in each state for each week between these dates and calculated their deltas. **Figure 10** shows the change in number of hotspots between weeks 1 and 2 and then 2 and 3.

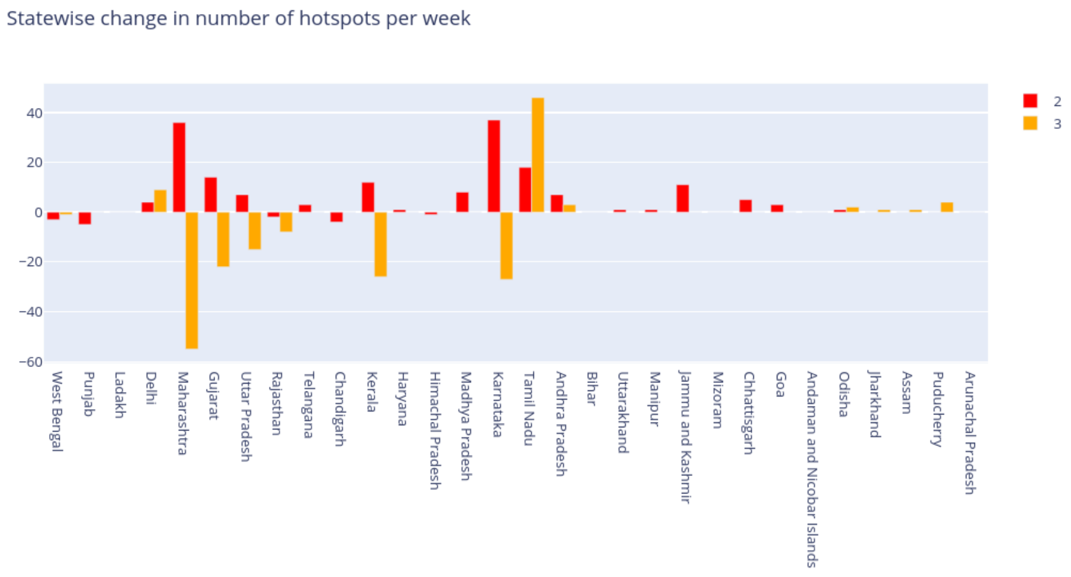


Figure 10: *Change in number of hotspots over 3 weeks*

Karnataka and Punjab have seen the most positive and negative changes respectively in week 2. Tamil Nadu and Maharashtra are the corresponding states for week 3.

3.7 Primary, Secondary and Tertiary Cases

From the remarks provided for each inidividual in `IndividualDetails.csv`, we can find out whether the person has travel history, or has been in contact with people having travel history. Based on this, **Figure 11** shows the top 10 states housing each type of case and their numbers.

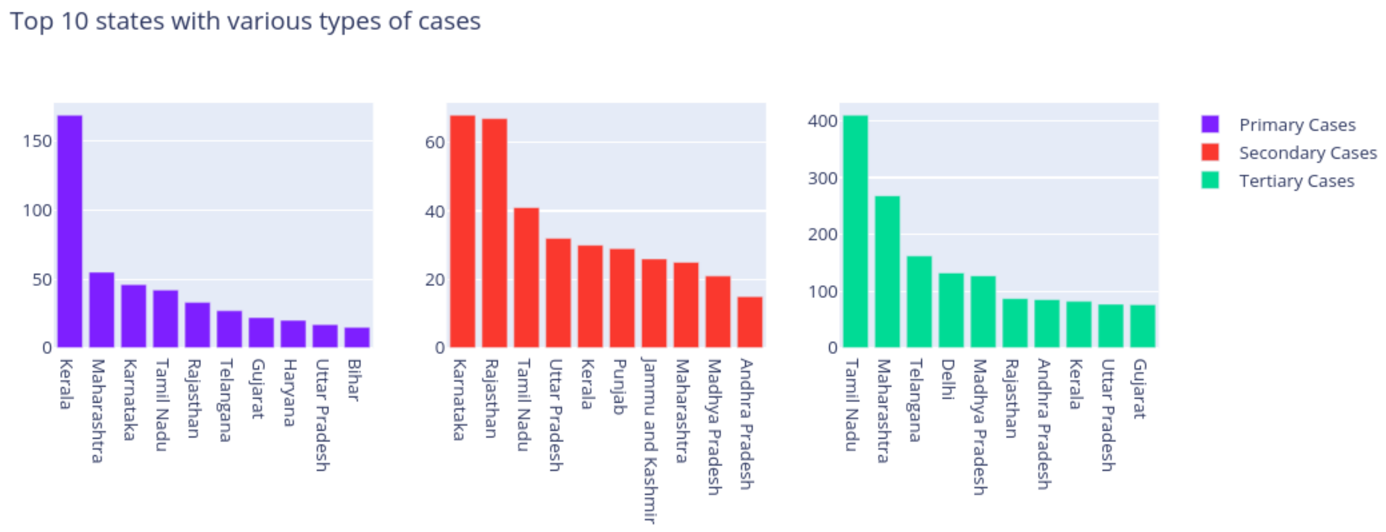


Figure 11: *Primary, secondary and tertiary cases by state*

3.8 Requirement of additional testing facilities

For this question, we used a combination of data from `ICMRDetails.csv` and `covid_19_india.csv`. It was observed from ICMR data that the number of tests performed per days and number of positive cases grows almost linearly after 19th March, 2020. **Thus, we assume that the number of positive cases and number of individuals tested will follow a linear relationship for the entire time period under consideration.**

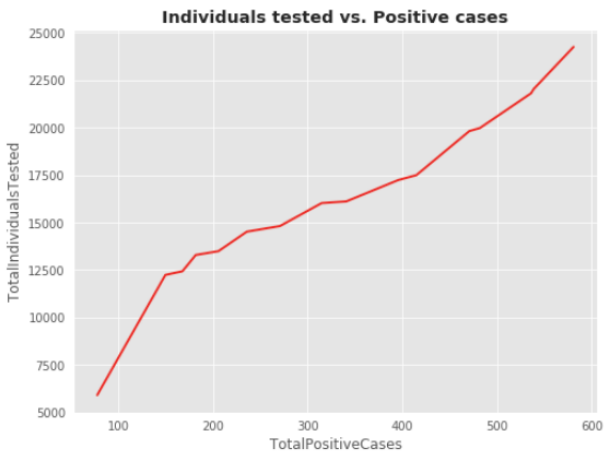


Figure 12: *Linear relationship between number of positive cases and number of individuals tested*

We used actual data for number of positive cases (until 7th April, 2020) from `covid_19_india.csv` and generate hypothetical number of cases until 20th April, 2020 by assuming a 10% rise each day. Then, we perform linear regression to determine the number of tests that will be performed on each day after 19th March 2020, up until 20th April, 2020. Then, by performing a first order difference on the number of individuals tested and dividing by 100, we can find the number of additional labs needed per day. **Figure 12** shows our results graphically.

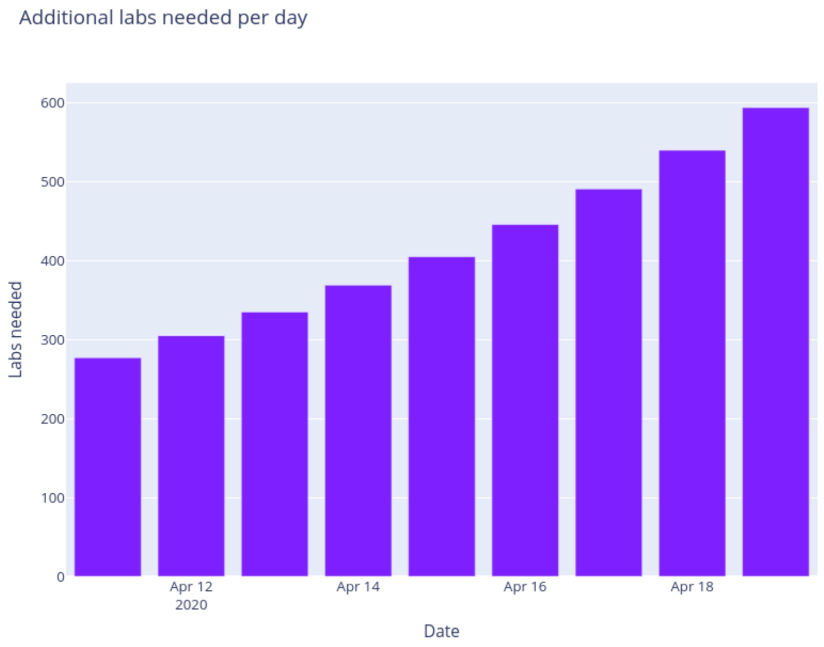


Figure 13: *Number of additional labs needed per day between 11 April and 20 April*

3.9 Flattening the Curve

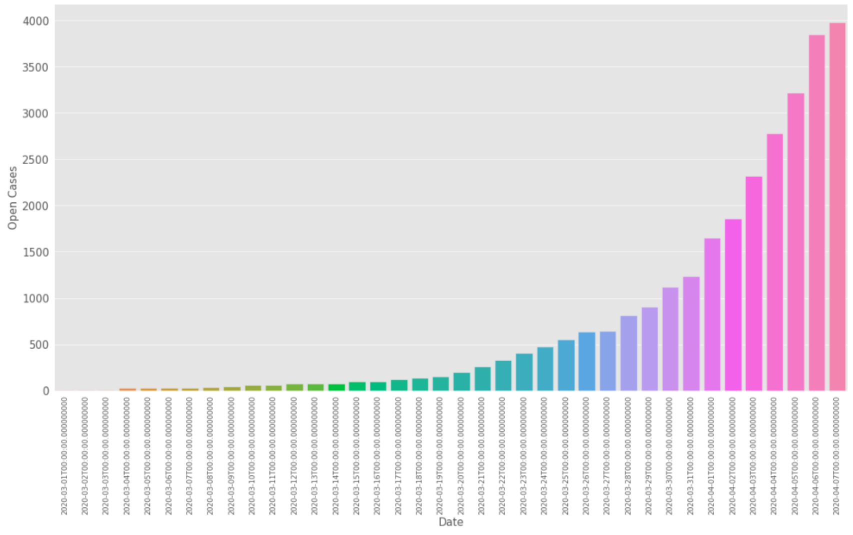


Figure 14: *Number of open cases since 1 March, 2020*

To illustrate the concept of 'flattening the curve', we will fit gaussian curves to this data. There are a lot of possible curves which could fit the data. It depends on the maximum number of cases reached.

This is quantified as the factor by which the maximum cases is more than the current number of cases. To cover plausible scenarios, the factors are taken to be 5, 10 and 20. For each of these scenarios, a number of mean and standard deviation values are tried and whichever combination gives the minimum least squared error is taken. By doing this, it is possible to get an idea of how the curve may flatten with time. Note that this is the total number of open cases at a given time, not the total number of confirmed cases.

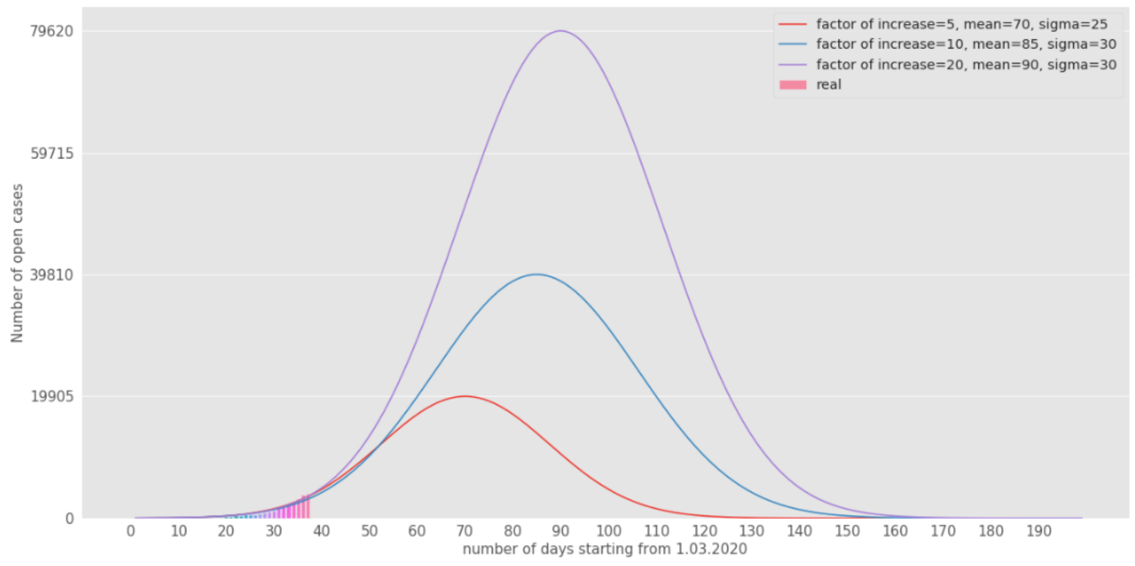


Figure 15: *Fitting gaussians to number of open cases*

We're still quite far away from attaining a flattened curve. It is necessary for us to follow lockdown strictly and maintain social distancing norms.

3.10 Effect of Lockdown

To realize this, we will perform SEIRD simulations for two situations: one in which there's no lockdown and another in which lockdown has been implemented. SEIRD stands for Susceptible-Exposed-Infected-Recovered-Dead, which is a model often used to emulate spread of epidemics in populations. It is governed by the following equations.

$$\frac{dS}{dt} = -\frac{\beta S I}{N}$$

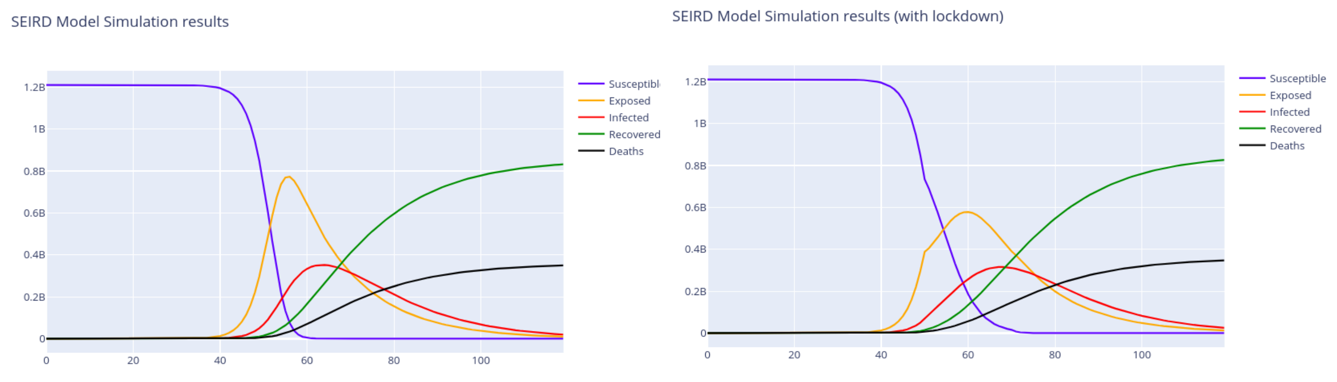
$$\frac{dE}{dt} = \frac{\beta S I}{N} - \sigma E$$

$$\frac{dI}{dt} = \sigma E - \gamma I$$

$$\frac{dR}{dt} = \gamma I$$

$$N = S + E + I + R$$

In these, N is the population size at any time and all greek variables are rate parameters. β affects how many people one person can spread an infection to. σ is the incubation rate: the rate at which latent individuals become infectious. γ is the recovery rate and μ is the mortality rate. **Figure 16** shows the simulation for 150 days without lockdown and **Figure 17** shows the same with lockdown enforced.



(a) *SEIRD simulation without lockdown*

(b) *SEIRD simulation with lockdown*

Figure 16: *SEIRD simulation to observe effect of lockdown*

To enforce lockdown, we reduce the value of β (from 3 to 1). It is worth noting how the number of exposed people (yellow curve) drastically falls down when lockdown is implemented. Also, the rate at which people become susceptible and exposed greatly reduces. Thus, at least in theory, lockdown is effective in reducing the rate at which infections rise, thus helping our medical infrastructure handle the situation more effectively. Please note that hyperparameters of this model were set based on heuristics, and need not accurately represent real life situations.

4 Acknowledgements

- Singular Value Decomposition for image compression, Tim Baumann: **Link**
- Implementing SEIRD model for epidemic simulations, Abhay Shukla on Medium: **Link**