UE17CS490B – Capstone Project Phase – 2

SEMESTER - VIII

END SEMESTER ASSESSMENT

Project Title : Probabilistic Models for Predicting Covid-19 Pandemic Spread
Project ID : PW21NVP01
Project Guide : Prof. Nitin V Pujari
Project Team : K Suhas(PES1201700816)
Kethan M V (PES1201701085)
Venkatesh K (PES1201701626)
Akash P S (PES1201701773)

Outline

- Abstract
- Team Roles and Responsibilities.
- Summary of Requirements and Design (Capstone Phase - 1)
- Summary of Methodology / Approach (Capstone Phase - 1)
- Modules and Implementation Details
- Project Demonstration and Walkthrough
- Results and Discussion
- Lessons Learnt
- Conclusion and Future Work

Team Roles and Responsibilities

Name	Role
Akash P S & Kethan M V	Extended Literature Survey
Venkatesh K, Kethan M V, Akash P S	Probabilistic Models design and Development
K Suhas	Identification of relevant and derived metrics
Venkatesh K, Kethan M V, Akash P S	Probabilistic Models testing and deployment
Akash P S, Kethan M V, Venkatesh K and K Suhas	Capstone II Project Report and Paper in IEEE format

Abstract

- The main aim of the project is to analyze the data and develop probabilistic models to predict the future spread of the disease, specifically confirmed cases and validate their efficacy by testing it with actual data.

- Creating a UI to enable easy access to COVID-19 data in the form of graphs and to integrate probabilistic models with the UI to operate and cascade through required operations on the platform seamlessly.

Summary of Requirements and Design

- Literature Survey
  - Literature survey was carried to gain knowledge about the virus and modelling the virus's trend.
- Data
  - Raw Data was sourced from a open source website "covid19india.org".
  - Data Cleaning was done to extract the useful information.
- UI was created to visualize the pandemic spread.

Summary of Methodology

- Data Cleaning
  - Raw data was obtained from covid19india.org
  - Data with negative values was handled
  - Missing state names were filled with the help of district names

- Data Preprocessing
  - Data was segregated quarterly for states in south India
  - Bitmap of Age in window for individual patient
    - >60
    - 40-60
    - 25-40
    - 10-25
    - <10

Summary of Requirements and Design

- Data Visualisation
  - All India visualisation of
    - Confirmed
    - Recovered
    - Deceased
  - State wise Visualisation of southern states
    - Karnataka
    - Maharashtra
    - Tamil Nadu
    - Kerala
    - Andhra Pradesh

Modules and Implementation Details

XGBoost

- Extreme Gradient Boosting is an implementation of gradient boosted decision trees for speed and performance.
- Boosting is an ensemble technique where new models are added to correct the errors made by existing models.
- XGBoost provides a parallel tree boosting that solves many data science problems in a fast and accurate way.

Modules and Implementation Details

XGBoost

- Data is obtained from districts.csv

- Below is a snapshot showing attributes and initial records of the dataset

	A	B	C	D	E	F	G	H
1	Date	State	District	Confirmed	Recovered	Deceased	Other	Tested
2	26-04-2020	Andaman and Nicobar I	Unknown	33	11	0	0	2679
3	26-04-2020	Andhra Pradesh	Anantapur	53	14	4	0	
4	26-04-2020	Andhra Pradesh	Chittoor	73	13	0	0	
5	26-04-2020	Andhra Pradesh	East Godavari	39	12	0	0	
6	26-04-2020	Andhra Pradesh	Guntur	214	29	8	0	
7	26-04-2020	Andhra Pradesh	Krishna	177	29	8	0	
8	26-04-2020	Andhra Pradesh	Kurnool	279	31	9	0	
9	26-04-2020	Andhra Pradesh	Prakasam	56	23	0	0	
10	26-04-2020	Andhra Pradesh	S.P.S. Nellore	72	23	2	0	

Modules and Implementation Details

XGBoost

- Training Data
  - Training data includes the attributes: Date, State, District, Confirmed and Recovered
  - The data is included uptil 25-04-2021 starting from 26-04-2020

Modules and Implementation Details

XGBoost

- Testing Data
  - Testing data includes the attributes: Date, State, District and ForecastId.
  - Testing Data starts from 26-04-2021 and ends on 02-05-2021[7-day window].

Modules and Implementation Details

XGBoost

- Model Building
  - Create a dataframe to store output, xout
  - Loop through different districts in each state and fit the data into the model
  - Following model fitting, we predict two values:
    - Confirmed cases
    - Recovered cases
- Sort the xout by ForecastId

Modules and Implementation Details

Code snippet of the model building:

```
for state in states:
    districts = X_xgtrain.loc[X_xgtrain.State == state, :].District.unique()
    #print(country, states)
    # check whether string is nan or not
    for district in districts:
        X_xgtrain_SD = X_xgtrain.loc[(X_xgtrain.State == state) & (X_xgtrain.District == district), ['Date', 'State', 'District'

        y1_xgtrain_SD = X_xgtrain_SD.loc[:, 'Confirmed']
        y2_xgtrain_SD = X_xgtrain_SD.loc[:, 'Recovered']

        X_xgtrain_SD = X_xgtrain_SD.loc[:, ['Date', 'State', 'District']]

        X_xgtrain_SD.State = le.fit_transform(X_xgtrain_SD.State)
        X_xgtrain_SD['District'] = le.fit_transform(X_xgtrain_SD['District'])

        X_xgtest_SD = X_xgtest.loc[(X_xgtest.State == state) & (X_xgtest.District == district), ['ForecastId', 'Date', 'State',

        X_xgtest_SD_Id = X_xgtest_SD.loc[:, 'ForecastId']
        X_xgtest_SD = X_xgtest_SD.loc[:, ['Date', 'State', 'District']]

        X_xgtest_SD.State = le.fit_transform(X_xgtest_SD.State)
        X_xgtest_SD['District'] = le.fit_transform(X_xgtest_SD['District'])

        #models_C[country] = gridSearchCV(model, X_Train_CS, y1_Train_CS, param_grid, 10, 'neg_mean_squared_error')
        #models_F[country] = gridSearchCV(model, X_Train_CS, y2_Train_CS, param_grid, 10, 'neg_mean_squared_error')

        xmodel1 = XGBRegressor(n_estimators=1000)
        xmodel1.fit(X_xgtrain_SD, y1_xgtrain_SD)
        y1_xpred = xmodel1.predict(X_xgtest_SD)

        xmodel2 = XGBRegressor(n_estimators=1000)
        xmodel2.fit(X_xgtrain_SD, y2_xgtrain_SD)
        y2_xpred = xmodel2.predict(X_xgtest_SD)

        xdata = pd.DataFrame({'ForecastId': X_xgtest_SD_Id, 'Confirmed': y1_xpred, 'Recovered': y2_xpred})
        xout = pd.concat([xout, xdata], axis=0)
```

Modules and Implementation Details

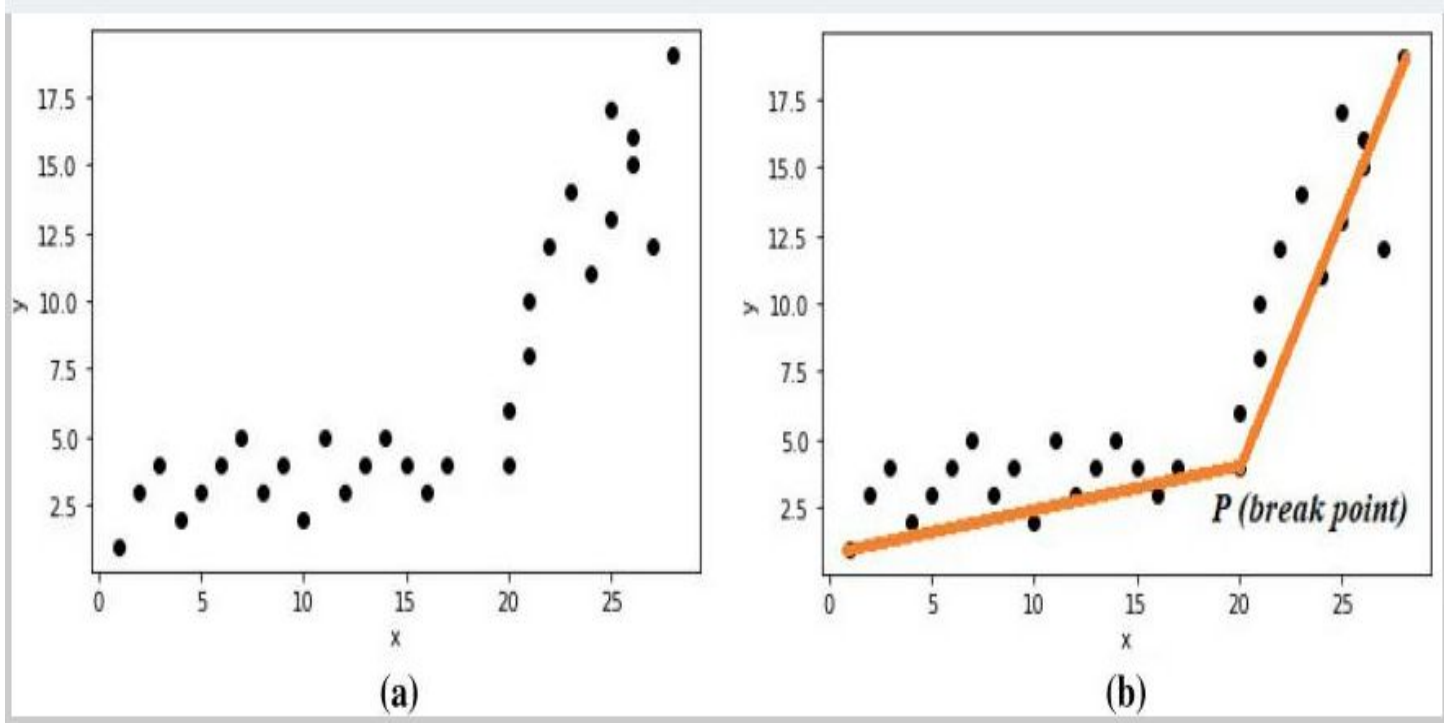
Piecewise Linear Regression

- A piecewise linear function is a function defined on a interval of real numbers.

- Data is non-linear, multiple linear regression lines are needed to fit the model.

Modules and Implementation Details

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x^*_{i2} + \epsilon_i$$



Modules and Implementation Details

- Open source package “pwlf” is used to train and predict the confirmed cases
- Least Square error is the metric used.
- Training data is chosen as,
  - t1 = Q1_2021(Until 25 April)
  - t2 = Q4 + Q1_2021
  - t3 = Q3 + t2
  - t4 = Q2 + t3 (All data until 25 April)
- Testing data
  - 26 April 2021 - 02 May 2021[7 Day window]

Modules and Implementation Details

Pseudo code

```
#Piecewise Linear Regression
import pwlf

#Test data for Quarter 1 of 2021
q1_2021 = [x for x in range(737791, 737882, 1)]

x = np.array(yearly['Date Announced'])
y = np.array(yearly['Confirmed'])


#Initialize piecewise linear fit with x and y data
myPWLF = pwlf.PiecewiseLinFit(x,y)

#Fit the data for three line segments
res = myPWLF.fit(3)

#Predict for the determined points
xHat = np.linspace(min(x), max(x), num = len(x))
yHat = myPWLF.predict(xHat)

```

Modules and Implementation Details

Pseudo code

```
for dist in dist_d.keys():
    seg = 4
    predicted_max = 10000000
    for train in dist_d[dist]:
        if (train != "best"):
            predicted = districts_model(dist_d[dist][train]['df'], seg,\
                                        quarter2_21.loc[quarter2_21['District'] == dist])
            if(predicted[:n]['Absolute Error'].mean() < predicted_max):
                dist_d[dist]["best"] = predicted[:n].reset_index(drop = True)
                predicted_max = predicted[:n]['Absolute Error'].mean()
    #dist_d[dist]['best'].to_csv("plr_"+dist[:3]+".csv")
```

Modules and Implementation Details

Simple Exponential Smoothing

- Exponential functions assign exponentially decreasing weights over time.

- $F_t = F_{t-1} + (A_{t-1} - F_{t-1})\ \alpha$
  - α - smoothing constant [0,1]
  - F_t - Predicted Value
  - F_{t-1} - Previous Predicted Value
  - A_{t-1} - Previous Actual Value

Modules and Implementation Details

Simple Exponential Smoothing

- Open source package “stats” is used to determine the optimal smoothing constant(α).
- Least Square error is the metric used.
- Training data is chosen as
  - Q1_2021(Until Feb)
  - Q4 - Q1_2021 Only last quarter
  - Q3 & Q4 - Last Two Quarters
  - Q2 to Q4 - Last three Quarters

Modules and Implementation Details

Simple Exponential Smoothing

- Testing Data(window of 7 days) is taken from 26 April 2021 - 02 May 2021
- Values are predicted using the smoothing constants obtained while modeling the training datasets.
- Absolute error and Percentage error are calculated using,
  - Absolute error = |Actual - Predicted|
  - Percentage error = Absolute Error/Actual

Modules and Implementation Details

Simple Exponential Smoothing

Pseudo code

In [55]:
```
def train_val(df,alpha):
    pred = [0]
    for i in range(1,len(df)+1):
        val = alpha*df[i-1] + ((1-alpha)*(pred[i-1]))
        pred.append(int(val))
    return pred
```

In [56]:
```
def predict_val(df_train,alpha,n):
    pred = df_train[len(df_train)-1]
    actual = alpha*pred
    for i in range(0,n-1):
        val = alpha*actual + ((1-alpha)*(pred))
        pred = val
        actual = alpha*pred
        df_train.append(int(val))
    return df_train
```

Modules and Implementation Details

Simple Exponential Smoothing

Pseudo code

```
for dist in dist_d.keys():
    predicted_max = 0
    for train in dist_d[dist]:
        if (train != "best"):
            predicted = predit(dist_d[dist][train]['df'],dist,dist_d[dist][train]['alpha'],n)
            if(predicted[len(dist_d[dist][train]['df']):]['Accuracy'].mean() > predicted_max):
                dist_d[dist]["best"] = predicted[len(dist_d[dist][train]['df']):]
                predicted_max = predicted[len(dist_d[dist][train]['df']):]['Accuracy'].mean()
    dist_d[dist]['best'].to_csv("ses"+dist+".csv")
```

Modules and Implementation Details

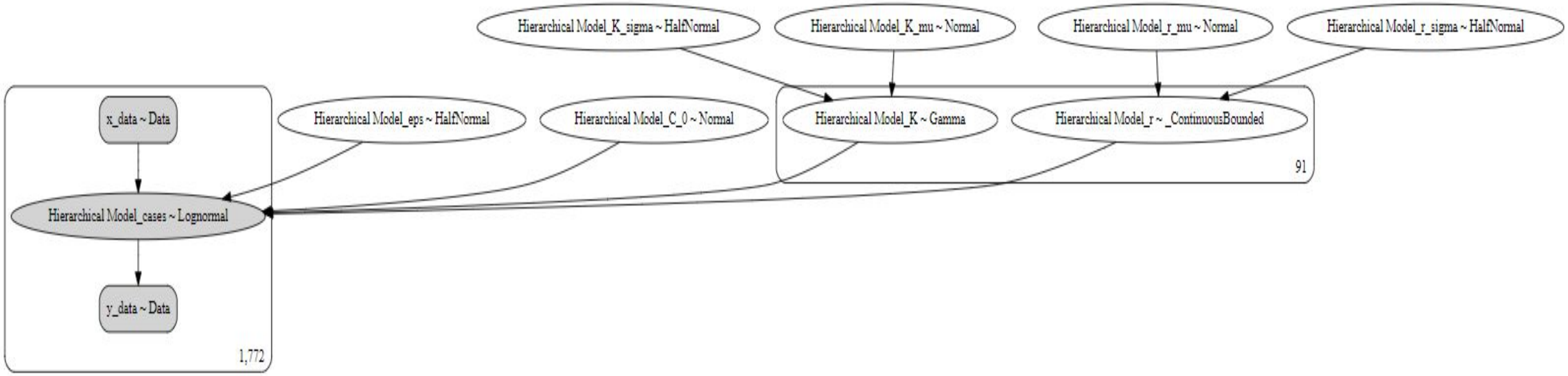
Hierarchical Bayesian Model

- Parameters are nested with one another at different levels.
- Low level parameters called hyperpriors are initially assumed from the data by assigning prior distribution creating hyperparameters.
- Hyperparameters are then assigned to the priors.
- Logistic model is built using these priors creating a 3-stage hierarchy.

Modules and Implementation Details

Hierarchical Bayesian Model

Graphical representation of Hierarchical Bayesian Model



Modules and Implementation Details

Hierarchical Bayesian Model

- Logistic equation:

$$\mu = \frac{\alpha}{1 + exp(-\beta(t - t_0))}$$

where,

Alpha - Maximum no of COVID-19 cases

Beta - Growth rate of coronavirus spread

Modules and Implementation Details

Hierarchical Bayesian Model

- Technologies used:
  - pymc3 - Open source probabilistic programming framework in python.
  - theano - Python library for optimizing and evaluating mathematical expressions.
  - MCMC(Markov chain Monte-carlo) - Features sampling algorithms such as NUTS (No-U-Turn Sampler).

Modules and Implementation Details

Hierarchical Bayesian Model

- Pseudo-code: Logistic Function

```
# Logistic regression function
def logistic(K, r, t, C_0):
    A = (K-C_0)/C_0
    return K / (1 + A * np.exp(-r * t))
```

Modules and Implementation Details

Hierarchical Bayesian Model

- Pseudo-code - Model Definition

```
# Model fitting using pymc3
bayesian_model = pm.Model("Bayesian Model")
with bayesian_model:
    BoundedNormal = pm.Bound(pm.Normal, lower=0.0)

    # Intercept
    C_0 = pm.Normal("C_0", mu=1000, sigma=100)

    # Growth rate
    r_mu = pm.Normal("r_mu", mu=0.4, sigma=0.1)
    r_sigma = pm.HalfNormal("r_sigma", 0.5)
    r = BoundedNormal("r", mu=r_mu, sigma=r_sigma, shape=n_districts)

    # Total number of cases
    K_mu = pm.Normal("K_mu", mu=800000, sigma=800000)
    K_sigma = pm.HalfNormal("K_sigma", 50000)
    K = pm.Gamma("K", mu=K_mu, sigma=K_sigma, shape=n_districts)

    # Likelihood error
    eps = pm.HalfNormal("eps")

    for i, district in enumerate(districts):

        df_district = sorted_data.loc[lambda x: (x.District == district)]

        # Loading train data
        t = pm.Data(district + "x_data", df_district['days_since_100_cases'])
        confirmed_cases = pm.Data(district + "y_data", df_district['Confirmed'])

        # Logistic regression
        growth = logistic(K[i], r[i], t, C_0)

        # Likelihood using Lognormal distribution
        pm.Lognormal(district, mu=np.log(growth), sigma=eps, observed=confirmed_cases)
```

Modules and Implementation Details

Hierarchical Bayesian Model

- Pseudo-code - Model Fitting

```
# NUTS sampler for drawing samples.
with hierarchical_model:
    trace = pm.sample()
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [Hierarchical Model_eps, Hierarchical Model_K, Hierarchical Model_K_sigma, Hierarchical Model_K_mu, Hierarchical Model_r,
Hierarchical Model_r_sigma, Hierarchical Model_r_mu, Hierarchical Model_C_0]
Sampling 4 chains, 1,513 divergences: 100%|████████████████████████████| 4000/4000 [4:37:34<00:00,  4.16s/draws]
There were 495 divergences after tuning. Increase `target_accept` or reparameterize.
There were 498 divergences after tuning. Increase `target_accept` or reparameterize.
There were 494 divergences after tuning. Increase `target_accept` or reparameterize.
There were 22 divergences after tuning. Increase `target_accept` or reparameterize.
The rhat statistic is larger than 1.4 for some parameters. The sampler did not converge.
The estimated number of effective samples is smaller than 200 for some parameters.
```

Modules and Implementation Details

Hierarchical Bayesian Model

- Pseudo-code - Predicting

```
with bayesian_model:
    for district in districts:

        # Sample posterior predicting
        df_district = sorted_data.loc[lambda x: (x.District == district)]
        x_data = np.arange(0, len(df_district))
        y_data = np.array([np.nan] * len(x_data))
        pm.set_data({district + "x_data": x_data})
        pm.set_data({district + "y_data": y_data})

    ppc_hierarchical = pm.sample_posterior_predictive(trace)
```

```
100%|████████████████████████████████████████████████████████| 1000/1000 [02:19<00:00,  7.15it/s]
```

Project Demonstration

DEMO

Walkthrough

Results and Discussion

- All models report more than 90% Accuracy for few districts
- Piecewise Linear Regression Model has the highest average accuracy among all the four probabilistic models
- Accuracy of Simple Exponential model decreases drastically over time.

Results and Discussion

- Hierarchical Bayesian model follows a lognormal curve so the model wouldn't be able to predict the results accurately as we have seen in the case of Delhi and Mumbai due to the steep rise in the confirmed cases.

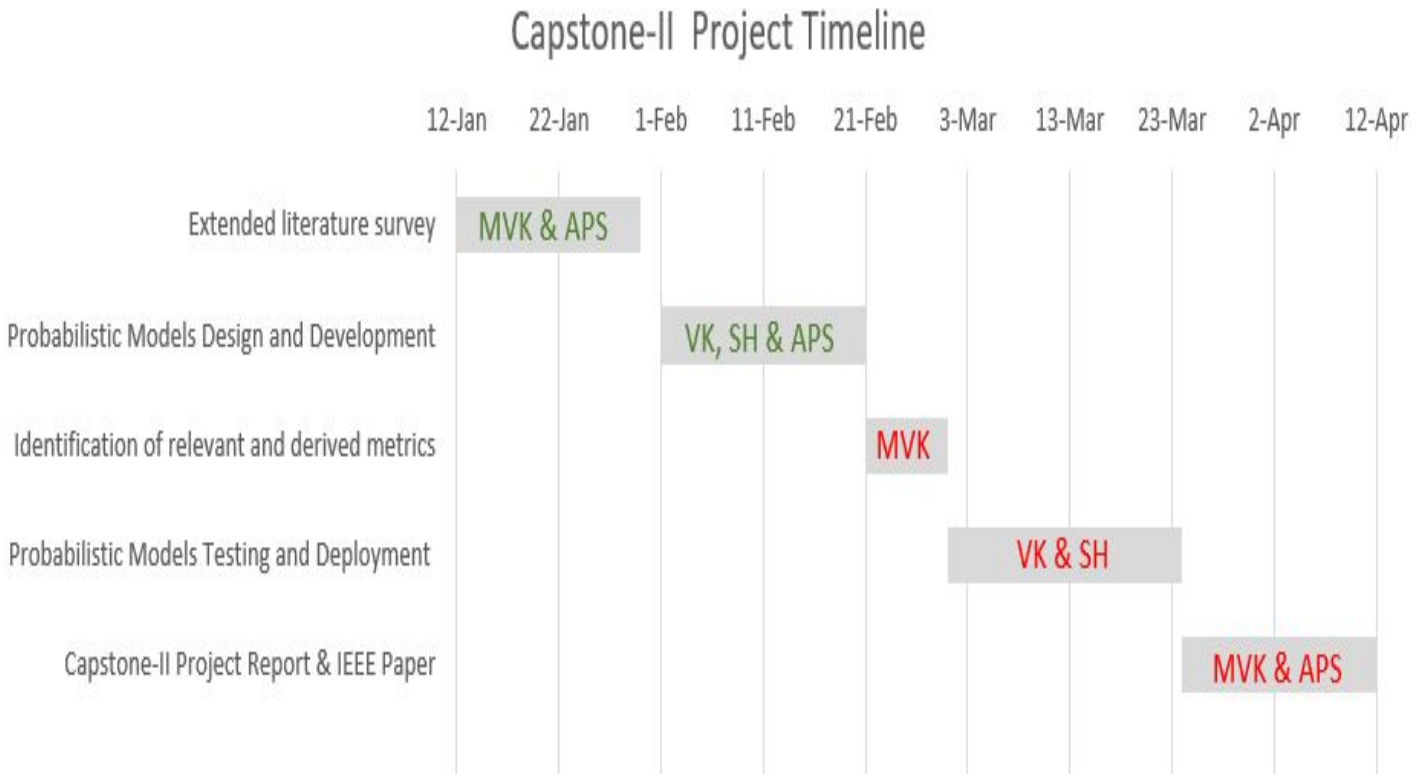
- Hierarchical Bayesian models tend to underestimate the number of confirmed cases which is true as the model fitted assumes that the cases follow a lognormal curve.

Results and Discussion

- XGBoost provides high accuracy for the first 3 days of prediction. The accuracy reduces as we move towards future days.

- The drawback of this model is that it does not predict according to the trend hence will not give satisfactory results for dates more than a week. For instance the percentage of incorrect prediction for confirmed cases is almost 20% for test data after 7 days.

Schedule


Capstone-II Project Timeline
12-Jan
22-Jan
1-Feb
11-Feb
21-Feb
3-Mar
13-Mar
23-Mar
2-Apr
12-Apr
Extended literature survey
MVK & APS
Probabilistic Models Design and Development
VK, SH & APS
Identification of relevant and derived metrics
MVK
Probabilistic Models Testing and Deployment
VK & SH
Capstone-II Project Report & IEEE Paper
MVK & APS

Schedule

- Probabilistics testing and deployment was delayed by a week
    - Segregating training dataset for five cities and four quarters was decided in a later stage.
    - Simple Moving average was Developed and Deployed.
    - Designed algorithm for predicting more than one value for simple exponential smoothing.

Lessons Learnt

- Predicting more than one value for simple exponential smoothing model.
- Converting regression models into probabilistic models by introducing confidence interval.
- Cannot fit all the district data using a single curve in Hierarchical Bayesian model.

Conclusion and Future work

- Reducing the confidence interval window for better precision.
- Finding the optimal number of segments for piecewise linear regression.
- Checking the accuracy for Hierarchical Bayesian model using different curves like exponential curve.

Thank You