International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 10, October-2020

Improving the Accuracy of Temperature Forecasting in Mumbai, Colaba

Sahil Rane
Dhirubhai Ambani International School

Abstract

Background: Temperature predictions are of great importance due to their implications for human activities. Extreme heat can lead to dangerous, even deadly, health consequences, including heat stress and heatstroke. Thus, there is a need to predict temperature accurately so that people can be warned about such conditions so that they take the appropriate precautions.

Methodology: In this paper, we look at temperature data in Mumbai (Colaba) from 2008 to 2020 and attempt to come up with predictive models for the maximum temperature. We carry out feature selection through filter methods first in order to efficiently use a variety of algorithms to develop predictive models. We use various mathematical techniques such as: Multiple Linear Regression (MLR), Simple Exponential Smoothing (SES), Artificial Neural Networks (ANN), and Auto-Regressive Integrated Moving Average (ARIMA) models to predict the temperature. The experimental results are evaluated and compared using the Root Mean Square Error (RMSE).

Results: On experimenting with all four models, it was discovered that the ARIMA model yields the best predictive model having a RMSE of 0.2587777 on testing data by removing some noise and an RMSE of 0.8213 with white noise. This model is also optimal as the residuals of this model are a gaussian white noise (which cannot be predicted). Furthermore, the poor performance of MLR indicates that temperature cannot be accurately modelled through a linear function of the variables considered.

ACKNOWLEDGEMENTS:

I would like to thank Smt. Shubhangi Bhute, Scientist-E (Indian Meterological Department) for guiding me with the project, helping me ideate, and find data for the research paper. Furthermore, I am grateful to Dr. Mahendra Mehta for helping me explore various areas in statistical predictions helping me acquire the knowledge base I needed to work on this paper.

BACKGROUND

This paper attempts to come up with a reliable predictive model for the maximum temperature in Mumbai. Although there are several research papers based on temperature predictive models in general, there were very few papers that aimed to predict temperatures in Mumbai specifically. The models cited in several research papers, for temperature prediction in areas all around the world, were not able to improve on an RMS error of approximately 1 in most cases. Thus, this paper aims to improve the RMS error of these previous models for the dataset of Mumbai (Colaba). In many papers, multiple linear regression and neural networks have been commonly used for temperature predictions with artificial neural networks being the most common. This paper also explores a variety of time series analysis techniques such as exponential smoothing and auto-regressive integrated moving average models.

Often temperatures in Mumbai are underestimated and necessary precautions are not taken while exiting one's house. Due to the adverse effects that this heat can have on the health of individuals, it is necessary that steps be taken at an organisational level to issue warnings to citizens so that they can take the necessary precautions and steps before leaving their house. This idea was inspired by the IMD's Heat Action Plan, which has been implemented in Ahmedabad. While the temperatures in Mumbai may not be as severe, precautions are necessary in order to protect and inform individuals about the adverse effects of heat.

METHODOLOGY:

Feature Selection

Pearson Correlation Test for feature selection:

Variable	Pearson Correlation Coefficient (r)
maxlag1	0.8872607
minlag1	0.2079691
avglag1	0.6158288
DAY	0.02438063
MONTH	-0.03973674
YEAR	0.01284328
totalprecipMM	-0.3852286
windspeedKmph	-0.162895
humidity	-0.432934
visibilityKm	0.2798653
pressureMB	0.2311805
cloudcover	-0.5144175
HeatIndexC	0.595783
DewPointC	-0.009764557
WindChillC	0.7967794

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 10, October-2020

We use the Pearson Correlation Coefficient (r) as a feature selection mechanism. In order to avoid overfitting of the data we will only be using the features that have $|r| > 0.5$

The pearson correlation co-efficient is used to measure correlation between different sets of data to understand how strong the correlation is between two variables. Above we have calculated and tabulated the pearson correlation co-efficients between the maximum temperature of the day with a variety of variables such as total precipitation, pressure, sun hours etc. with a lag of 1 day.

$$r = \frac{\sum_i (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_i (x_i - \overline{x})^2} \times \sqrt{\sum_i (y_i - \overline{y})^2}}$$

By the above criteria we select the variables: maxlag1 (Maximum temperature of the previous day), avglag1 (Average temperature of the previous day), cloudcover (fraction of the sky obscured by clouds when observed from a particular location), HeatIndexC (Index that combines relative humidity and actual temperature), WindChillC (Combination of windspeed and temperature)

$$WindChillC = 13.12 + 0.6215 \times T - 11.37 \times (V^{0.16}) + 0.3965T(V^{0.16})$$

T = Temperature in degree Celsius

V = Wind velocity in kilometres per hour

$$\begin{aligned} \mathrm{HI} = {} & c_1 + c_2 T + c_3 R + c_4 TR + c_5 T^2 + c_6 R^2 + c_7 T^2 R + c_8 TR^2 + c_9 T^2 R^2 + \\ & + c_{10} T^3 + c_{11} R^3 + c_{12} T^3 R + c_{13} TR^3 + c_{14} T^3 R^2 + c_{15} T^2 R^3 + c_{16} T^3 R^3 \end{aligned}$$

$$\begin{aligned} c_1 &= 16.923, & c_2 &= 0.185\,212, & c_3 &= 5.379\,41, & c_4 &= -0.100\,254, \\ c_5 &= 9.416\,95 \times 10^{-3}, & c_6 &= 7.288\,98 \times 10^{-3}, & c_7 &= 3.453\,72 \times 10^{-4}, & c_8 &= -8.149\,71 \times 10^{-4}, \\ c_9 &= 1.021\,02 \times 10^{-5}, & c_{10} &= -3.8646 \times 10^{-5}, & c_{11} &= 2.915\,83 \times 10^{-5}, & c_{12} &= 1.427\,21 \times 10^{-6}, \\ c_{13} &= 1.974\,83 \times 10^{-7}, & c_{14} &= -2.184\,29 \times 10^{-8}, & c_{15} &= 8.432\,96 \times 10^{-10}, & c_{16} &= -4.819\,75 \times 10^{-11}. \end{aligned}$$

T = Temperature in degree Celsius

R = Relative Humidity

HI = HeatIndexC

Multivariate Linear Regression:

First, in order to perform a multiple linear regression on our data it is necessary to carry out feature scaling as the range for each variable differs significantly. If feature scaling was not done then we variables with maximum range will dominate in the training of the regression model. We will bound all our variables in the interval [0,1]. We use min-max normalisation technique on our data.

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

We will use Ordinary least Squares (OLS) regression to get the best linear unbiased estimators.

Lets call the temperature on the i^{th} day some T_i

$$T_1 = \beta_0 + \beta_1 x_{11} + \beta_2 x_{21} + \beta_3 x_{31} + \beta_4 x_{41} + \beta_5 x_{51} + \varepsilon_1$$

$$T_2 = \beta_0 + \beta_1 x_{12} + \beta_2 x_{22} + \beta_3 x_{32} + \beta_4 x_{42} + \beta_5 x_{52} + \varepsilon_2$$

$$T_3 = \beta_0 + \beta_1 x_{13} + \beta_2 x_{23} + \beta_3 x_{33} + \beta_4 x_{43} + \beta_5 x_{53} + \varepsilon_3$$

$$\vdots$$

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 10, October-2020

$$\therefore T_n = \beta_0 + \beta_1 x_{1n} + \beta_2 x_{2n} + \beta_3 x_{3n} + \beta_4 x_{4n} + \beta_5 x_{5n} + \varepsilon_n$$

When we vectorise these equations we get:

$$\vec{T} = \begin{bmatrix} 1 & x_{11} & x_{21} & x_{31} & x_{41} & x_{51} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{1n} & x_{2n} & x_{3n} & x_{4n} & x_{5n} \end{bmatrix} \times \begin{bmatrix} \beta_0 \\ \vdots \\ \beta_5 \end{bmatrix} + \vec{\varepsilon}$$

We can write this as:

$$\vec{T} = X\vec{\beta} + \vec{\varepsilon} \Rightarrow \vec{\varepsilon} = \vec{T} - X\vec{\beta}$$

Since we are using OLS regression we want to minimise $\left|\left|\vec{\varepsilon}\right|\right|^2 = \sum_{i=1}^{n} \varepsilon_i^2$

$$\sum_{i=1}^{n} \varepsilon_i^2 = \varepsilon' \times \varepsilon = \left(\vec{T} - X\vec{\beta}\right)' \times \left(\vec{T} - X\vec{\beta}\right)$$

We use gradient descent with simultaneous update to minimise the error function.

$$\frac{\partial}{\partial \beta_j} \sum_{i=1}^{n} (x_{.i}\beta_. - T_i)^2 = \sum_{i=1}^{n} [(x_{.i}\beta_. - T_i) \times x_{ji}]$$

Algorithm:

Repeat using simultaneous update
{

$$\beta_j := \beta_j - \alpha \times \sum_{i=1}^{n} [(x_{i.}\beta_. - T_i) \times x_{ij}]$$

}

We use the R programming language to run this program and obtain the multivariate linear regression analysis

Code for Multivariate Linear Regression Used:

```
#First we need to read the data from the csv file with headers
data <- read.csv("weather_data_24hr_master1.csv", header = T)
#remove null row
data <- data[-4265,]
#remove columns based on feature selection
data <- data[,c(4,7,9,17,18,20)]
# we need to normalize the data (max-min normalization)
data$maxlag1 <- (data$maxlag1-min(data$maxlag1))/(max(data$maxlag1)-min(data$maxlag1))
data$avglag1 <- (data$avglag1-min(data$avglag1))/(max(data$avglag1)-min(data$avglag1))
data$cloudcover <- (data$cloudcover-min(data$cloudcover))/(max(data$cloudcover)-min(data$cloudcover))
data$HeatIndexC <- (data$HeatIndexC-min(data$HeatIndexC))/(max(data$HeatIndexC)-min(data$HeatIndexC))
data$WindChillC <- (data$WindChillC-min(data$WindChillC))/(max(data$WindChillC)-min(data$WindChillC))
# Data Partition
# Partitioning the Data into testing and training data
# If we repeat the learning, we get the same result
set.seed(222)
ind <- sample(2,nrow(data), replace = T, prob =c(0.7,0.3))
training <- data[ind==1,]
testing <- data[ind==2,]
# Training the Linear regression model
MLR <- lm(maxtempC~., data = training)
summary(MLR)
```

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 10, October-2020

```
# Prediction on training
output <- predict(MLR, training[,-1])
df <- data.frame(output, training[,1])
df
sum((df[,1]-df[,2])^2)/nrow(training)
# Prediction on testing
output <- predict(MLR, testing[,-1])
df <- data.frame(output, testing[,1])
df
sum((df[,1]-df[,2])^2)/nrow(testing)
```

Explanation of code:

We first import the comma-splitted values containing the raw data for our analysis. We then keep only the columns that were deemed statistically significant by the pearson correlation test. We then carry out the max-min normalisation of the data as specified above. We then partition 70% of the data for training our regression model and 30% of the data to test the regression model. We create the multivariate ordinary least squares linear regression model using gradient descent as outlined above. We then calculate the sum of squared error for both the training and the testing data using the regression model found.

Results:

The regression equation obtained by our program is:

$$h_\beta(x) = 24.2541 + 11.1606 \times x_1 + 0.8512 \times x_2 - 0.8588 \times x_3 + 0.4494 \times x_4 + 0.4872 \times x_5$$

Metric of Analysis for results:

We will now be analysing the results produced by the multivariate linear regression model. For analysis we will be finding the square root of mean sum of squared errors (RMS errors) returned by the program. We define the function of our prediction model (hypothesis function) as $h_\beta(x)$. The formula for the RMS error is

$$RMS\ error = \sqrt{\frac{\sum_{i=1}^{n}\left(h_\beta(x) - T_i\right)^2}{n}}$$

For the training data the sum of squared errors was: 1.004442
For the testing data the sum of squared errors was: 1.062815

Since the behaviour of our model is similar for the training and testing data we can conclude that our model is not over or underfitting the dataset.

Overall, we can conclude that our data with lag 1through a linear relationship does not yield ideal results. This Multiple linear regression was also autoregressive in nature but did not give us the desired RMS error.

Simple Exponential Smoothing

We will also attempt to forecast temperature using time series analysis techniques. We will begin with simple exponential smoothing as a technique because there is no clear trend or seasonality in the data when the time series is plotted. We don't use Holt's Exponential smoothing or Holt-Winter Exponential Smoothing as our data does not have clear increasing/decreasing trends or any visible seasonality based on the plot. In simple exponential smoothing, we say the predicted observation is a weighted average of previous observations. While calculating the weighted averages, the weights decrease exponentially as lag increases, that is, the smallest weights are associated with the oldest observations:

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 10, October-2020

Notation:

$$l_0: start\ value$$

$$y_t: actual\ temperature\ at\ time\ t$$

$$\overline{y_t}: forecasted\ temperature\ at\ time\ t$$

$$0 \leq \alpha \leq 1$$

Weighted average form of exponential smoothing:

$$\overline{y_{t+1}} = \alpha y_t + (1-\alpha)\overline{y_t}$$

Let's try to arrive at the above generalisation by considering smaller examples.

$$\overline{y_1} = l_0$$

$$\overline{y_2} = \alpha y_1 + (1-\alpha)l_0$$

$$\overline{y_3} = \alpha y_2 + (1-\alpha)\overline{y_2}$$

$$\overline{y_4} = \alpha y_3 + (1-\alpha)\overline{y_3}$$

$$\vdots$$

$$\overline{y_t} = \alpha y_{t-1} + (1-\alpha)\overline{y_{t-1}}$$

$$\overline{y_{t+1}} = \alpha y_t + (1-\alpha)\overline{y_t}$$

When we substitute each equation in the next we get:

$$\overline{y_3} = \alpha y_2 + (1-\alpha)(\alpha y_1 + (1-\alpha)l_0)$$

$$= \alpha y_2 + (1-\alpha)(\alpha)y_1 + (1-\alpha)^2 l_0$$

$$\overline{y_4} = \alpha y_3 + (1-\alpha)(\alpha y_2 + (1-\alpha)(\alpha)y_1 + (1-\alpha)^2 l_0)$$

$$= \alpha y_3 + (1-\alpha)(\alpha)y_2 + (1-\alpha)^2(\alpha)y_1 + (1-\alpha)^3 l_0$$

$$\vdots$$

$$\overline{y_{t+1}} = \sum_{i=0}^{t-1} (\alpha(1-\alpha)^i y_{t-i}) + (1-\alpha)^t \cdot l_0$$

This is the weighted average form of the simple exponential smoothing model. We want to find the value of α that minimises the sum of squared errors (SSE), therefore, we use the gradient descent algorithm to minimise the squared error.

$$SSE = \sum_{i=1}^{t} (y_i - \overline{y_i})^2 = \sum_{i=1}^{t} e_i^2$$

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 10, October-2020

We use the R programming language to run gradient descent in order to minimise error.

Code for Simple Exponential Smoothing Used:

```
#first read the comma splitted values containing the dataset
data <- read.csv("weather_data_24hr_master1.csv", header = T)
#filter only maxtempC out which is relevant to our timeseries
data <- data[,4]
#remove NA values
data <- data[-4265]
data
#Let's partition the data into testing and training data.
training <- data[1:2990]
testing <- data[2991:4264]
#Create a timeseries using the maxtempC data
timeseries <- ts(training, frequency = 365, start = c(2008,183))
timeseries
#plot the timeseries
plot.ts(timeseries)
install.packages("TTR")
library("TTR")
#Simple exponential Smoothing
timeseriesforecasts <- HoltWinters(timeseries, beta=FALSE, gamma=FALSE, l.start = 27)
timeseriesforecasts
timeseriesforecasts$fitted
plot(timeseriesforecasts)
#calculate the sum of squared errors for our forecasts
a<-(timeseriesforecasts$SSE)
#calculate the mean of the SSE
MSE<- a/length(training)
#calculate the RMS error
sqrt(MSE)
```

Explanation of code:

We first import the comma-splitted values containing the raw data for our analysis. We then remove all columns except the maxtempC for our timeseries. We partition our dataset into training data and testing data. We train our simple exponential smoothing model on the first 70% of our data. We test this model using the remaining 30% of the data. We then convert the vector into a timeseries. We then plot the timeseries and run simple exponential smoothing on our timeseries. This gives us the minimum value of α. We also find the RMS error of our model on the training data. We then use the same value of α to run our simple exponential smoothing model on the testing data and calculate the RMS error for the same.

Results:

By running gradient descent we get the result that the error is minimised when the learning parameter $\alpha = 0.8400815$

This makes out weighted average exponential smoothing equation to be as follows:

$$\overline{y_{t+1}} = \sum_{i=0}^{t-1} ((0.8400815)(0.1599185)^i y_{t-i}) + (0.1599185)^t \cdot 27$$

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 10, October-2020

Metric of Analysis for results:

We will now be analysing the results produced by the simple exponential smoothing model. One of the main issues that arises in the use of this model is that it does not perform well on sudden fluctuations in the data whereas it performs very well for gradual increases or decreases in our data. The high alpha value indicates high dependence on previous day values of the max temperature. The high alpha also indicates that as lag increases data beyond lag=3 $(0.8400815)(0.1599185)^3 = 0.00343571$ becomes too small to have relevance to our forecasted value. To determine how well our model fits the data we calculate the RMSE for the testing data and the training data. We define our prediction model function for simple exponential smoothing to be: $\overline{y_\alpha}(t)$. The temperature on the t[th] day is defined as $M(t)$.

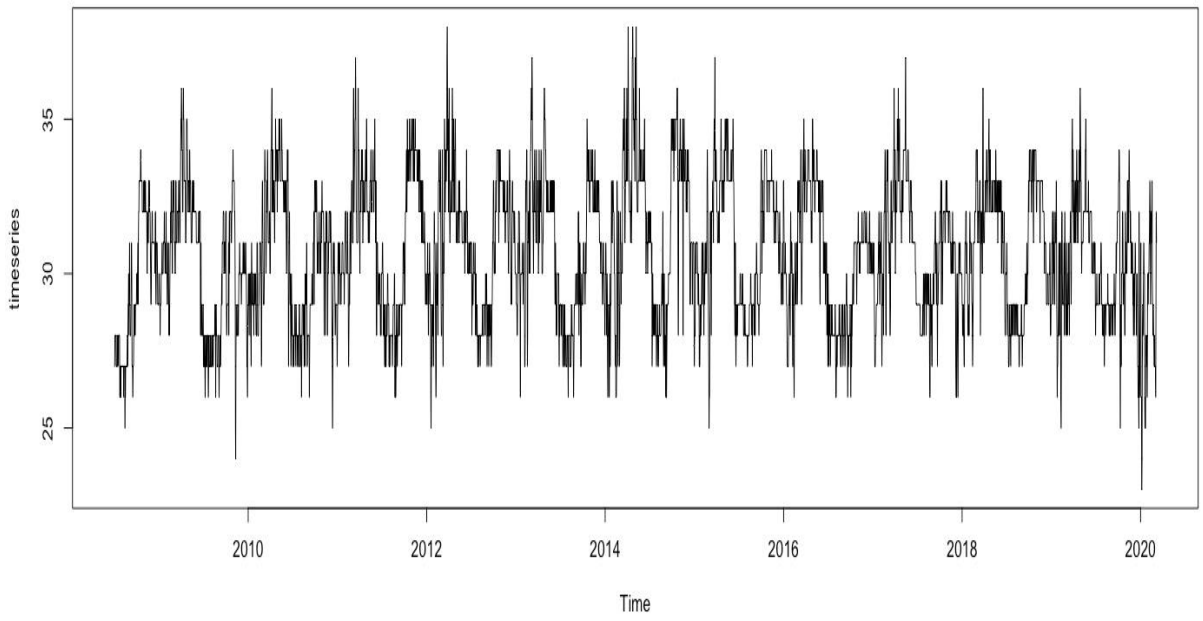
Therefore we get that the RMS error is:

$$RMS\ error = \sqrt{\frac{\sum_{t=1}^{T}\left(\overline{y_\alpha}(t) - M(t)\right)^2}{T}}$$

For the training data the RMS error was: 1.048192
For the testing data the RMS error was: 1.0522970

The model gives very similar values for the RMS training and testing data, thus overfitting of data is not an issue for our model.

Since simple exponential smoothing doesn't help and the data cannot be expressed as a linear relationship we look at artificial neural networks to explore non-linear hypotheses functions for the data.



Time Series plot

Artificial Neural Networks

A neural network is inspired from the model of a human brain consisting of neurons. When neural networks learn they independently find a variety of connections in the data which helps with complicated predictions when we have large datasets with several variables. Each neuron receives the values of the variables (features) from the training set and calculates a weighted average of these values. The result of this calculation is passed through a non-linear activation function.

For the m[th] neuron we supply the vector x of training examples and it calculates the weighted average and returns a z_m value. In this scenario b is a bias constant that is added to the outcome of each neuron.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 10, October-2020

$$z = \sum_{i=1}^{n} (w_i x_i) + b \ (\forall j = 1, 2, 3, \dots, m)$$

The activation function g(z) is applied on each z value to give our forecast value ($\hat{y}_i$) . Thus the output of each neuron is passed to the activation function.

Without an activation function, the neural network would simply return a linear function thus not being able to model complex data with small error.

The way a neural network is trained is by the value of the loss function. We use the sum of squared error as our error function thus during the training our model minimises the error of the neural network. In our learning the values of the weights and the bias parameter are changed in order to minimise the error function. We calculate the partial derivative of the loss function in order to arrive at a minimum value for the error. We use the backpropagation algorithm in order to train our neural network.

For our regression problem the loss function would be Mean Squared Error, which squares the difference between actual (y_i) and predicted value ($\hat{y}_i$).

$$Sum\ of\ squared\ error\ (C) = \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$$

Using chain rule we get the following:

$$\frac{\partial C}{\partial w_i} = \frac{\partial C}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z} \times \frac{\partial z}{\partial w_i}$$

First lets calculate $\frac{\partial C}{\partial \hat{y}}$:

$$\frac{\partial C}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}} \times \left(\sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \right) = 2 \times \sum_{i=1}^{m} (y_i - \hat{y}_i)$$

Now let's calculate the $\frac{\partial \hat{y}}{\partial z}$:

$$\frac{\partial \hat{y}}{\partial z} = \frac{\partial}{\partial z} \times g(z)$$

$$= \frac{\partial}{\partial z} \left(\frac{1}{1 + e^{-z}} \right)$$

$$= \frac{e^{-z}}{(1 + e^{-z})^2}$$

$$= \frac{1}{(1 + e^{-z})} \times \left(1 - \frac{1}{1 + e^{-z}} \right)$$

$$= g(z) \times \left(1 - g(z) \right)$$

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 10, October-2020

Finally let's calculate the $\frac{\partial z}{\partial w_i}$:

$$\frac{\partial z}{\partial w_i} = \frac{\partial}{\partial w_i} \times (z)$$

$$= \frac{\partial}{\partial w_i} \sum_{i=1}^{n} (w_i x_i) + b$$

$$= x_i$$

Therefore, we get:

$$\frac{\partial C}{\partial w_i} = x_i \times g(z) \times \big(1 - g(z)\big) \times 2 \times \sum_{i=1}^{m} (y_i - \hat{y_i})$$

We also calculate $\frac{\partial C}{\partial b}$ which we can get from the above formula as the input for the bias operator is 1.

$$\frac{\partial C}{\partial b} = g(z) \times \big(1 - g(z)\big) \times 2 \times \sum_{i=1}^{m} (y_i - \hat{y_i})$$

After backpropagation is carried out we focus on optimisation which is done through gradient descent. For this we define our learning rate as α.

Repeat until convergence:

{

$$w_i := w_i - \left(\alpha \times \frac{\partial C}{\partial w_i}\right)$$

$$b := b - \left(\alpha \times \frac{\partial C}{\partial b}\right)$$

}

Code for Artificial Neural Networks Used:

```
#First we need to read the data from the csv file with headers
data <- read.csv("weather_data_24hr_master1.csv", header = T)
#remove null row
data <- data[-4265,]
#remove columns based on feature selection
data <- data[,c(4,7,9,17,18,20)]
# we need to normalize the data (max-min normalization)
data$maxlag1 <- (data$maxlag1-min(data$maxlag1))/(max(data$maxlag1)-min(data$maxlag1))
data$avglag1 <- (data$avglag1-min(data$avglag1))/(max(data$avglag1)-min(data$avglag1))
data$cloudcover <- (data$cloudcover-min(data$cloudcover))/(max(data$cloudcover)-min(data$cloudcover))
data$HeatIndexC <- (data$HeatIndexC-min(data$HeatIndexC))/(max(data$HeatIndexC)-min(data$HeatIndexC))
data$WindChillC <- (data$WindChillC-min(data$WindChillC))/(max(data$WindChillC)-min(data$WindChillC))
#Data partition to divide our data into training anad testing data (70% training 30% testing)
#we set seed in order to be able to repeat the learning
set.seed(222)
ind <- sample(2,nrow(data), replace = T, prob =c(0.7,0.3))
```

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 10, October-2020

```
training <- data[ind==1,]
testing <- data[ind==2,]
#install the neural network packages in R
install.packages("neuralnet")
library(neuralnet)
#We create a neural network n trained on the training data
#This neural network has the error function as the sum of squared error
#It has the activation function as the sigmoid function
#It has 2 neurons
n <- neuralnet(maxtempC~.,
               data = training,
               hidden = 2,
               stepmax = 9999999,
               err.fct ='sse',
               act.fct = 'logistic',
               linear.output = T)
n
#We plot our trained neural network
plot(n)
# We calculate the RMS error for our neural network on the training and testing dataset
output <-compute(n, training)
p1 <- output$net.result
sqrt(sum((training$maxtempC-p1)^2)/nrow(training))
max((training$maxtempC-p1))
output <- compute(n, testing)
p2 <- output$net.result
sqrt(sum((testing$maxtempC-p2)^2)/nrow(testing))
```

Explanation of code:

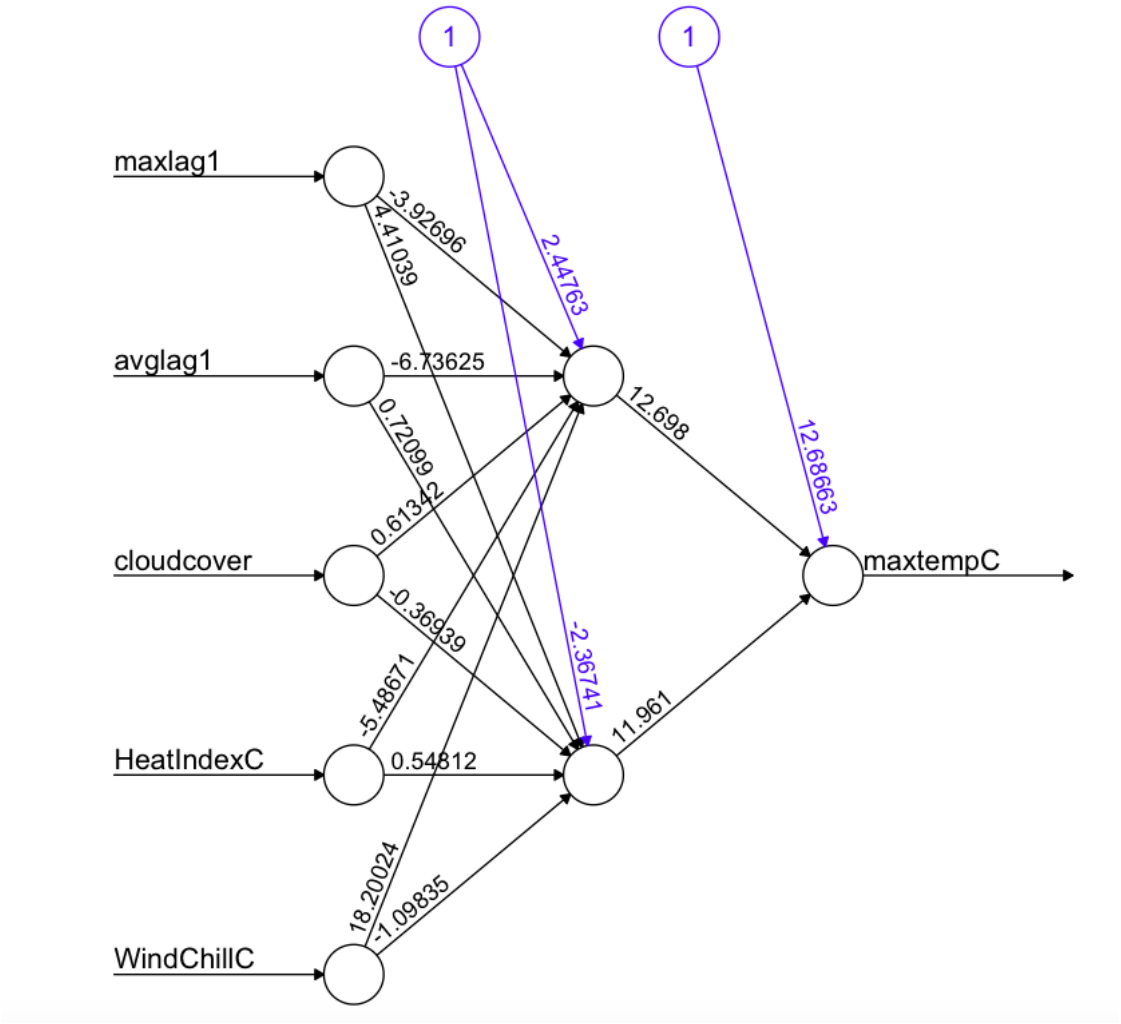
We first import the csv file containing the raw data for our analysis. We then retain all the feature columns from our feature selection and remove all NA values. We first carry out feature scaling of our data through min-max normalisation. We then partition our dataset into training data (70%) and testing data (30%). We train an artificial neural network with 2 neurons to fit our training data. We find the RMS error of our model on the training data. We then run our model on the testing data and calculate the RMS error.

Results:

The neural network plot has been included below with the weights on each “wire” connecting our input to the neurons and our neurons to the output layer.

The value of the bias operator for the hidden layer is 2.44763 and the bias for the output layer is 12.68663.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 10, October-2020



Neural network plot

Metric of Analysis for results:

We will now analyse the results of the neural network model. The use of neural networks allows us to predict the data using a non-linear hypothesis function. The high weights on the WindChillC, maxlag1, and avglag1 indicate that these are the dominant features in our learning. A variety of number of neurons were tried out. 2 neurons were chosen as larger number of neurons seemed to overfit our data and would have an overly complicated hypothesis function. To determine how well our model fits the data we calculate the RMSE for the testing data and the training data. We define our prediction model function for our neural network to be: $\hat{y}_i(x)$. The actual temperature is represented by y_i.

Therefore we get the formula for the RMS error to be:

$$RMS\ error = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i(x) - y_i)^2}{n}}$$

For the training data the RMS error was: 0.9906021
For the testing data the RMS error was: 1.012613

Since the RMS error for the training and testing data using our model is similar we can conclude that our model does not overfit our data. It also performs better than the exponential smoothing model as well as multivariate regression but only marginally. Since expressing the forecast value as a linear/non-linear hypothesis of explanatory variables does not give us the desired results we try to use time series analysis using ARIMA.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 10, October-2020

Auto-regressive Integrated Moving Average (ARIMA)

In order to apply the ARIMA technique we require our timeseries data to be stationary. This means that our data has a constant mean, constant variance (deviation from the mean), and no seasonality. It is necessary to confirm whether our time series is stationary using the Augmented Dickey-Fuller test (ADF test) and the Kwiatkowski–Phillips–Schmidt–Shin (KPSS test).

The ADF test is a unit root test. The mathematics and proofs related to the ADF test will not be described in this paper but can be found in [1]. For our purposes we only need to know the null and alternative hypothesis as defined by the test.

H_0: (null hypothesis) The given data is non-stationary

H_1: (alternative hypothesis) The given data is stationary

On running the ADF test on our time series we get a p-value ≤ 0.01

Since the p-value is less than the significance level of 0.05, we can safely reject the null hypothesis and conclude that our data is stationary. However, for large datasets the adf test can be erroneous rejecting the null hypothesis a vast majority of the times. Hence, we use the KPSS test to confirm that our data is stationary. We use the KPSS test for level stationarity.

For the KPSS test the hypotheses are as follows:

H_0: (null hypothesis) The data is level stationary

H_1: (alternative hypothesis) The given data is level non-stationary

On running the KPSS test we get the p-value ≥ 0.1

Since the p-value is greater than the significance level of 0.05 we cannot reject the null hypothesis which supports our previous conclusion that our data is stationary. Since our data is stationary we can proceed by using auto-regressive integrated moving average processes on our data to come up with an accurate forecasting model for the data.

Since our data is stationary we do not require an ARIMA model and we can directly use an ARMA model (Auto-Regressive Moving Average).

Before we can define an ARMA model we will define white noise.

Definition 4.1: White noise: It is an identically and independently distributed stochastic process $\{X_t\ ,\ t \in \mathbb{Z}\}$ with mean zero such that there is no serial correlation between values of stochastic process in the present and past.

A gaussian white noise is when our stochastic process X_t is:

$$X_t \sim \mathcal{N}(0, \sigma^2)$$

White noise timeseries cannot be predicted as they are a sequence of random numbers. If the series of forecast errors are white noise it suggests that our model cannot be improved.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 10, October-2020

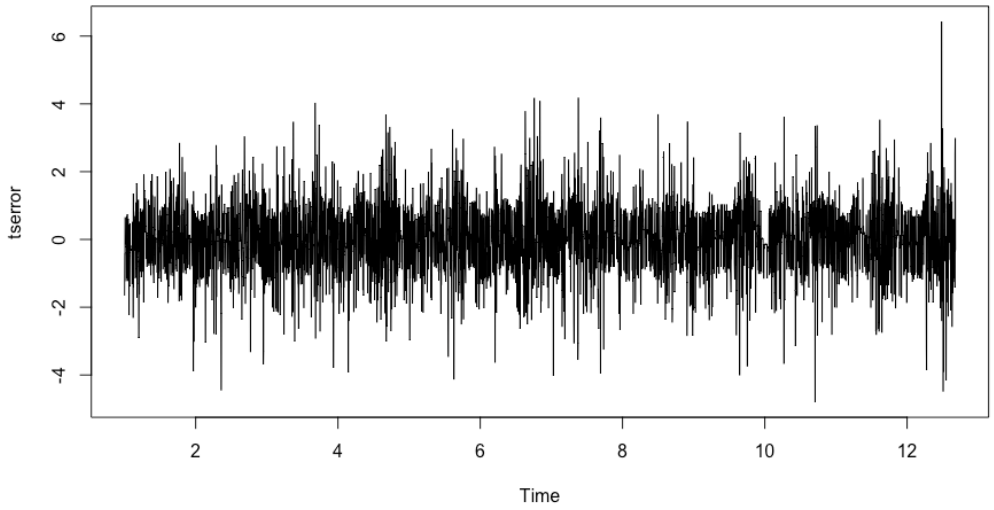


Figure: Plotting the residuals of our ARIMA(2,0,2) model

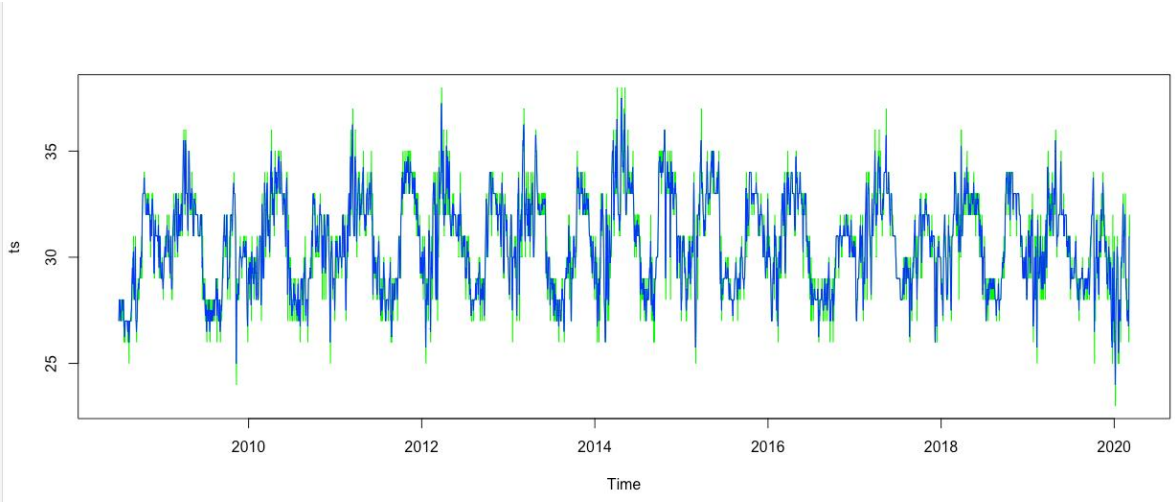
We used the ARIMA model on our timeseries and the residuals of our model were a gaussian white noise with mean 0 and variance 1. However, we want to improve our model despite this white noise. To do this we will attempt to smooth our timeseries in order to reduce the white noise in our timeseries. We can ignore some random fluctuations in our data and thus it is justified to take the moving average of our data. We will use a triangular moving average smoothing technique in order to minimise the error caused by this white noise. For this we will defined our new smooth timeseries to be $\overline{y}$ and our original time series to be y.

Mathematically our smoothed time series of order 2 will be defined as:

$$\overline{y_t} = \frac{y_t + y_{t-1}}{2}$$

To get the triangular moving average $(\overline{\overline{y_t}})$to smooth our time series we apply the moving average of order 2 again.

$$\overline{\overline{y_t}} = \frac{\overline{y_t} + \overline{y_{t-1}}}{2}$$



Blue: modified timeseries with triangular moving average

Green: original time series

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 10, October-2020

An auto-regressive model is one in which we assume that the values of the time series in the future depends on past values of the time series. It is a linear model relating values of the time series to past values of the time series. Let this time series be y_t. Then, for this timeseries the k[th] order auto-regression model (AR(k)) is:

$$y_t = \beta_0 + \beta_1 y_{t-1} + \beta_2 y_{t-2} + \cdots + \beta_k y_{t-k} + \epsilon_t$$

$$y_t = \epsilon_t + \beta_0 + \sum_{i=1}^{k} \beta_i y_{t-i}$$

A moving average model is one in which we assume that the values of the time series in the future depend on the previous residual terms of the time series. It is a linear model relating the value of the time series to past values of the error. Therefore, the k[th] order moving average model (MA(k)) is:

$$y_t = \epsilon_t + \theta_0 + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \theta_3 \epsilon_{t-3} + \cdots + \theta_k \epsilon_{t-k}$$

$$y_t = \epsilon_t + \theta_0 + \sum_{i=1}^{k} \theta_i \epsilon_{t-i}$$

An ARMA model simply combines an auto-regressive model and a moving average model. Thus, an ARMA model is such that a value of the time series can be predicted based on previous values of the time series. Thus an ARMA(p,q) model is:

$$y_t = \epsilon_t + \beta_0 + \beta_1 y_{t-1} + \beta_2 y_{t-2} + \cdots + \beta_k y_{t-p} + \theta_0 + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_k \epsilon_{t-q}$$

$$y_t = \epsilon_t + \beta_0 + \sum_{i=1}^{p} \beta_i y_{t-i} + \theta_0 + \sum_{i=1}^{q} \theta_i \epsilon_{t-i}$$

The algorithm for choosing the best ARMA(p,q) model is not outlined in this paper. However, we use maximum likelihood estimation to arrive at the best ARMA model. We use the R programming language along with the Hyndman-Khandakar algorithm [2] to minimise Akaike's Information Criterion (AIC) using maximum likelihood estimation.

$$AIC = -2\log(L) + 2(p + q + k + 1)$$

$$L: Likelihood\ of\ the\ data$$

$$k = 1\ if\ \beta_0 + \theta_0 \neq 0$$

$$k = 0\ if\ \beta_0 + \theta_0 = 0$$

Code for ARIMA Used:

```
#first read the comma splitted values containing the dataset
data <- read.csv("weather_data_24hr_master1.csv", header = T)
#filter only maxtempC out which is relevant to our timeseries
data <- data[,4]
#remove NA values
data <- data[-4265]
#looking at the first few values of our vector
head(data)
#sequential data partition into training (70%) and testing data (30%)
training <- data[1:2990]
testing <- data[2991:4264]
#convert the data into a time series
ts <- ts(data, frequency = 365, start = c(2008,183))
#smooth the time series using the triangular moving average
```

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 10, October-2020

```
tst2 <- ts(rollmean(rollmean(ts,2),2),frequency = 365, start = c(2008,183))
#plotting the original time series and the smoothed time series
plot(ts, col = 'green')
lines(tst2, col = 'blue')
#carrying out the augmented dickey-fuller test on our time series for stationarity
adf.test(ts)
#carrying out the kpss test to further confirm level stationarity of the data
kpss.test(ts, null = "Level")
#converting the training data into a time series
tstrain <- ts(training, frequency = 365, start = c(2008,183))
#smoothing the training time series using triangular moving average
ts1 <- ts(rollmean(rollmean(tstrain,2),2))
#converting the testing data into a time series
tstest <- ts(testing, frequency = 365, start = c(2016,252))
#smoothing the testing time series using triangular moving average
ts2 <- ts(rollmean(rollmean(tstest,2),2))
#we fit the training data to the arma(2,2) model
fit<-arima(ts1, c(2,0,2))
#finding out the RMS error of the arma(2,2) model
accuracy(fit)
fitted(fit)
#applying our previously derived model on the testing data
refit<-Arima(ts2, model = fit)
#calculating the accuracy of our model on the testing data
accuracy(refit)
#plotting the residuals of our model
plot(residuals(fit))
```

Explanation of code:

We first import the comma splitted values and filter only the relevant maxtempC data from the dataset. We then create multiple time series: one containing the entire dataset, one containing the training data, and one containing the testing data. We then use triangular moving averages to smooth these time series. We also run the ADF test and KPSS test on the data to test for stationarity. We then fit the ARMA(2,2) model to our data and calculate the RMS error on the training and testing data. We plot our residuals to observe whether it is a white noise time series to ensure optimality of our time series.

Results:

According to the ADF and KPSS test we get that our time series is stationary. We get that an ARMA(2,2) model that best fits our data.

```
Coefficients:
         ar1      ar2      ma1      ma2  intercept
      1.5080  -0.5176  -0.7015  -0.1054    30.6690
s.e.  0.0469   0.0452   0.0485   0.0260     0.3655

sigma^2 estimated as 1.011:  log likelihood = -4259.82,  aic = 8531.63
```

Co-efficients of ARMA model

Metric of Analysis for results:

We will now analyse the results of the ARMA(2,2) model. To determine how well our model fits the data we calculate the RMS error for the testing data and the training data. We define our prediction model function for our ARMA model to be: $\hat{y}_i$. The actual temperature is represented by y_i.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 10, October-2020

$$RMS\ error = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}}$$

The plot of our residuals is also a white noise time series. Thus, we can conclude that our ARMA model is optimal as white noise cannot be predicted.

For the training data the RMS error (after removing white noise) was: 0.2593737
For the testing data the RMS error (after removing white noise) was: 0.2587777

For the training data the RMS error (without removing white noise) was: 0.8115352
For the testing data the RMS error (without removing white noise) was: 0.8213673

Since the RMS error for the training and testing data using our model is similar we can conclude that our model does not overfit our data and is a reliable predictor even on unseen testing data. It also performs significantly better than the exponential smoothing model, multivariate regression model as well as the neural network model.

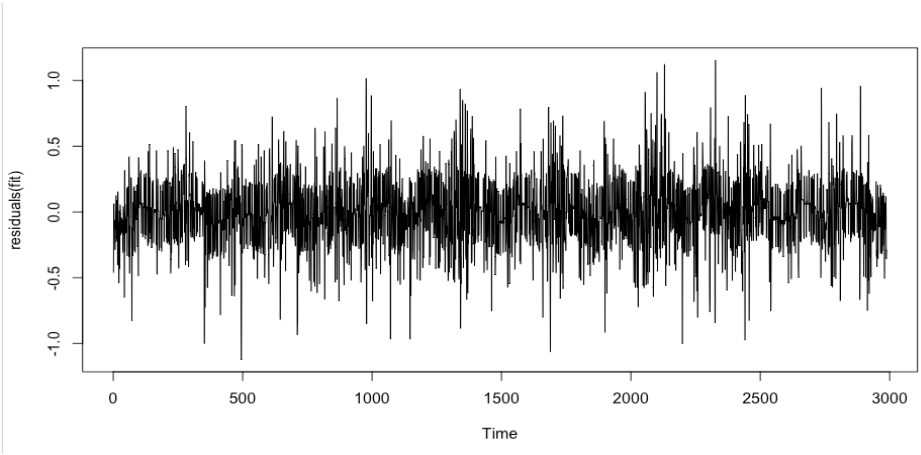


Figure: plot of residuals of the ARMA(2,2) model

Results:

The results of the research are that the ARMA(2,2) model works best for prediction of data in Mumbai (Colaba). After the removal of noise this model gives an RMS error of 0.2587777, which is much lower than the previous values. To outline the reliability of the model the actual temperature recorded and model predictions were compared for both maximum and minimum temperature in Colaba.

To test the model the predictions for 15 days were carried out everyday.

Maximum Temperature:
Accurate: 9
Usable: 4
Incorrect: 2

Minimum Temperature:
Accurate: 13
Usable: 2
Incorrect: 0

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 10, October-2020

Date	Actual max temp	Actual min temp	model prediction max	error	scale	model prediction min	error	scale
17/07/20	28.4	25.5	29	1	accurate	25.2	0	accurate
18/07/20	27.8	25.3	28.8	1	accurate	25.5	0	accurate
19/07/20	30.2	23.4	29	1	accurate	25.3	2	usable
20/07/20	32.0	25.5	30.2	2	usable	23.8	2	usable
21/07/20	31.8	27	31.5	0	accurate	25.5	1	accurate
22/07/20	31.5	27	31.2	0	accurate	26.6	0	accurate
23/07/20	32.2	27.2	31.1	1	accurate	26.6	1	accurate
24/07/20	31.8	25.5	31.8	0	accurate	26.8	1	accurate
25/07/20	28.4	26	31.4	3	incorrect	25.5	0	accurate
26/07/20	30.8	25.5	28.8	2	usable	26	0	accurate
27/07/20	31	25.5	31	0	accurate	25.6	0	accurate
28/07/20	28	25.5	30.9	3	incorrect	25.6	0	accurate
29/07/20	26.8	25	28.4	2	usable	25.5	0	accurate
30/07/20	30	25	27.7	2	usable	25.4	0	accurate
31/07/20	31	25.5	30.2	1	accurate	25.1	0	accurate

REFERENCES:

[1] http://debis.deu.edu.tr/userweb/onder.hanedar/dosyalar/1979.pdf
[2] https://otexts.com/fpp2/arima-r.html
[3] http://www.imdmumbai.gov.in
[4] http://www.imdmumbai.gov.in/scripts/search.asp