A

Project report

On

“Rainfall prediction model”

Submitted by

VASUDEVAN P

ABSTRACT

The main aim of this project is to create a model which can predict the rainfall in Bangalore , this is done through an Open source whether API called Open-Meteo. We collect historical data from the past 3 years (15/02/2020) to (15/02/2023) to achieve our goal.

INTRODUCTION

Overview:

Data are collected from Open-Meteo under Historical Weather API. We select location , timezone and the parameters to complete our dataset. After extacting and loading the data we need to process the data such that data must contain no missing value, no outliers. Appropriate analysis are to be done to understand the data this includes data visualization. Drawing information from the data visualization will play a key part in this project. To conclude our work we create a model which can predict the rainfall in mm (milli meters).

Objectives

1..1. The objective of this project is to develop an model which can predict the rainfall in Bangalore.

1..2. To provide better knowledge of rainfall pattern with respect the parameters that we choose from the API.

Software Tools Use

We developed this project on a PC using Jupyter notebook with python3.9.7, tensorflow version 2.7.0 , dataset was huge and needed much available space.

Limitation of System

Special attention is to be taken when dealing with the data's are generated hourly. In (Rain in mm parameter) statictically 0 has least significance but here 0 means no rainfall and the rainfall are measured in mm.

Assumping a day of random variations which includes light drizzle, heavy rain and no rain in these scenerio our data can be highly skewed. Upon close examination of the data 0's in our dataset is in high numbers, hence we focus on outliers. Datapoints are skewed, care was taken to ensure datapoints are symmentric. When training the data the main problem was convergence though regularization and Feature scaling was experimented none seemed to improve this problem, it was taken care through batchsize and epoch.

Dataset

Data was Collected through Open-meteo following Hourly parameters where chosen :-

temperature_2m:- Air temperature at 2 meters above ground.

relativehumidity_2m:- Relative humidity at 2 meters above ground.

dewpoint_2m :- Dew point temperature at 2 meters above ground.

surface_pressure:- Atmospheric air pressure reduced to mean sea level (msl) or pressure at surface. Typically pressure on mean sea level is used in meteorology. Surface pressure gets lower with increasing elevation.

Rain:- Only liquid precipitation of the preceding hour including local showers and rain from large scale systems.

Cloudcover:- Total cloud cover as an area fraction.

windspeed_10m:- Wind speed at 10 or 100 meters above ground. Wind speed on 10 meters is the standard level.

winddirection_10m:- Wind direction at 10 or 100 meters above ground.

soil_temperature_0_to_7cm:- Average temperature of different soil levels below ground.

soil_moisture_0_to_7cm:- Average soil water content as volumetric mixing ratio at 0-7, 7-28, 28-100 and 100-255 cm depths.

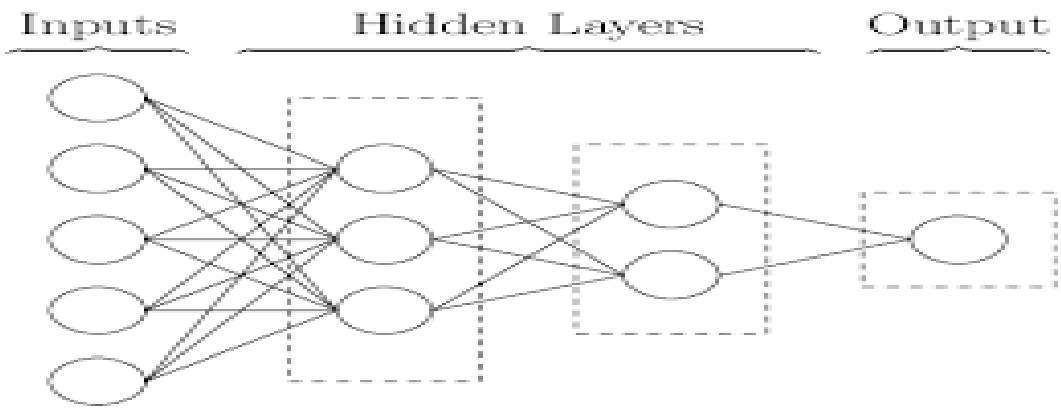
Breaking down the dataset :- As this is a hourly data set on an average across all three 3 years we have around(9,000 hours) which on 3 years equates to (27,000). Our dataset is around the same considering 2020 was a leapyear. Rain (mm) has 20,000 zero value which means no rain so this has to be taken into account because , we have days where it might have rained more than the usual. This type of datapoints can have a adverse effect on training of our model.

Source:-(https://open-meteo.com/en/docs/historical-weather-api)

Architecture

The first 2 layers consists of with 9 neuron as the first input layer(9 because we have neuron), the second layer consists of 18 neuron with ReLu as its activation function, the final layers has 1 neuron which gives us the desired output.

Input layer=>ReLU =>Hidden Layer =>ReLU => Output Layer=>Output(rain in mm)



Dense layer from keras : this layer creates a kernel that is convolved with the layer input to produce a tensor of outputs.

Relu: this activation function rectifies all the negative values to zero and establishes a linear function.

IMPLEMENTATION

- Data's are loaded from the desktop using pandas library.
- Data inside the Dataset are closely inspected and to check if everything is in order.
- Data wrangling, Feature Engineering and Exploratory Data are all part of Data preprocessing and are essential.

- Data preprocessing are done on the data which includes treating the missing value and droping the outliers.
- In this scenario we find 88 data points are missing. Our aim objective is to preserve atleast 70% of data with high variance we have in total 26,256 data points of which 88 contributes to less than 1 percent of the data so it is wise to drop the missing values rather than treat it.
- We have taken two approaches in dealing with Outliers:-
    - Z-score.
    - IQR(Inter-quartile range technique).
- Z-score has a threshold value of 3 if datapoint is found to be more than the threshold it is removed from our dataset.
- IQR deals with Quartiles we will drop all the rows which lie within upper and lower range.
- After treating outlier we move onto Data visualization this process helps us to discover the hidden patterns and behaviour in our dataset.
- Heatmap, Histogram, Scatter plots are some of the visual techniques are can use.
- Since we are dealing with number to numeric vs numeric data it is very important to know the correlation between each other .
- We do not what a highly skewed data because it can yield improper results.
- Artifical Neural Network(ANN) is choosen because.
    - We are dealing with relatively high data and parameters though small interms of features but are really essential to this process.
    - Other Regression techniques was also experimented but Artificial Neural Network seemed like a better fit.
- Keras framework are used in this project.
- Model was trained on 25 epochs and 128 Batch size yielding 97% accuracy and 5% in training loss. (Accuracy was kept under 90 % to avoid overfitting.)
- Model was saved in a json object and model weights can be retrieved for future use.
- Model gave us a 97% accuracy after performing 25 epoch.

In [53]:
```
ann.fit(x_train, y_train,
              batch_size=512,
              epochs=50)
```

```
Epoch 1/50
30/30 [==============================] - 1s 1ms/step - loss: 1694.1299 - accuracy: 0.7345
Epoch 2/50
30/30 [==============================] - 0s 2ms/step - loss: 65.4821 - accuracy: 0.3125
Epoch 3/50
30/30 [==============================] - 0s 2ms/step - loss: 6.6498 - accuracy: 0.6292
Epoch 4/50
30/30 [==============================] - 0s 1ms/step - loss: 2.6232 - accuracy: 0.6683
Epoch 5/50
30/30 [==============================] - 0s 1ms/step - loss: 2.0940 - accuracy: 0.7149
Epoch 6/50
30/30 [==============================] - 0s 1ms/step - loss: 1.8783 - accuracy: 0.7289
Epoch 7/50
30/30 [==============================] - 0s 1ms/step - loss: 1.7290 - accuracy: 0.7322
Epoch 8/50
30/30 [==============================] - 0s 2ms/step - loss: 1.6051 - accuracy: 0.7357
Epoch 9/50
30/30 [==============================] - 0s 2ms/step - loss: 1.4970 - accuracy: 0.7394
Epoch 10/50
30/30 [==============================] - 0s 1ms/step - loss: 1.4039 - accuracy: 0.7419
Epoch 11/50
```

```
Epoch 43/50
30/30 [==============================] - 0s 3ms/step - loss: 0.3051 - accuracy: 0.8478
Epoch 44/50
30/30 [==============================] - 0s 3ms/step - loss: 0.2962 - accuracy: 0.8501
Epoch 45/50
30/30 [==============================] - 0s 3ms/step - loss: 0.2875 - accuracy: 0.8546
Epoch 46/50
30/30 [==============================] - 0s 3ms/step - loss: 0.2796 - accuracy: 0.8575
Epoch 47/50
30/30 [==============================] - 0s 3ms/step - loss: 0.2721 - accuracy: 0.8586
Epoch 48/50
30/30 [==============================] - 0s 3ms/step - loss: 0.2650 - accuracy: 0.8608
Epoch 49/50
30/30 [==============================] - 0s 2ms/step - loss: 0.2582 - accuracy: 0.8655
Epoch 50/50
30/30 [==============================] - 0s 2ms/step - loss: 0.2514 - accuracy: 0.8692
```

In [54]:
```
y_pred = ann.predict(x_test)
ann.evaluate(x_test, y_test, verbose=0)
```

Out[54]: [0.24846115708351135, 0.8618595004081726]

Results

After training, results show that accuracy is directly proportional to the number of epochs. Increasing epoch size increases the test accuracy but while the batch size is increased then accuracy decreases.

Number_of_runs=(total_number_of_dataset)/BatchSize. In this project we had choosen BatchSize =512 and total_number_of_dataset=18926.

Number_of_runs=(18926)/(512) (which is more or less 30).

CONCLUSION:

Model we created was encouraging though if we had a high volume of data we could have pushed this project even further.