Predicting Percentage Increase in Covid-19 Cases Worldwide using Regression Decision Trees

Utkarsh Patel
18EC30048

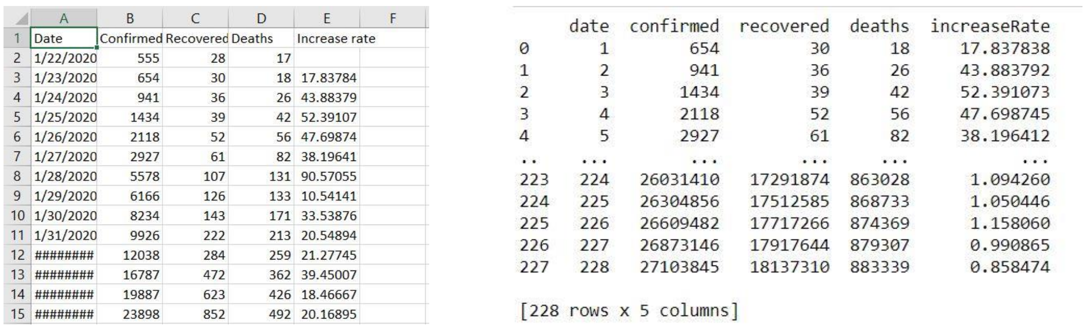
Akhil
18CS10070

Abstract

This report is submitted as part of an assignment of the course CS60050 – Machine Learning, IIT Kharagpur. The dataset contained the time-series percentage increase in COVID-19 cases worldwide. The attributes were date, confirmed cases, recovered cases, number of deaths and percentage increase in number of cases. The target attribute was the percentage increase in cases. It was required to build a regression decision tree for this task. This report briefly explains the procedure we followed to achieve the goal and the results obtained.

1 Generating Containers

We first created a class `Record` to stores the records of the dataset in Python virtual environment. The attributes of the class `Record` are: `date`, `confirmed`, `recovered`, `deaths` and `increaseRate`. While other attributes represented the same thing as in dataset, we modified the interpretation of `date` attribute. Instead of representing a string literal, we converted it to number of days since `EPOCH`, where `EPOCH` is a reference date. We used `datetime` library to do the same. Later, the records (which are now `Record` instances) are fused to form a `pandas.DataFrame` object. This is done because it is quite simple to handle data using such objects.



	A	B	C	D	E	F
1	Date	Confirmed	Recovered	Deaths	Increase rate	
2	1/22/2020	555	28	17		
3	1/23/2020	654	30	18	17.83784	
4	1/24/2020	941	36	26	43.88379	
5	1/25/2020	1434	39	42	52.39107	
6	1/26/2020	2118	52	56	47.69874	
7	1/27/2020	2927	61	82	38.19641	
8	1/28/2020	5578	107	131	90.57055	
9	1/29/2020	6166	126	133	10.54141	
10	1/30/2020	8234	143	171	33.53876	
11	1/31/2020	9926	222	213	20.54894	
12	########	12038	284	259	21.27745	
13	########	16787	472	362	39.45007	
14	########	19887	623	426	18.46667	
15	########	23898	852	492	20.16895	

```
     date  confirmed  recovered  deaths  increaseRate
0       1        654         30      18     17.837838
1       2        941         36      26     43.883792
2       3       1434         39      42     52.391073
3       4       2118         52      56     47.698745
4       5       2927         61      82     38.196412
..    ...        ...        ...     ...           ...
223   224   26031410   17291874  863028      1.094260
224   225   26304856   17512585  868733      1.050446
225   226   26609482   17717266  874369      1.158060
226   227   26873146   17917644  879307      0.990865
227   228   27103845   18137310  883339      0.858474

[228 rows x 5 columns]
```

Fig 1. Dataset before and after generating containers

2 Generating Decision Tree for given `max_depth`

The user is asked to enter a `max_depth`. Taking this as input, we build a decision tree that doesn't exceed this maximum depth. While building the trees of a given maximum depth, the entire dataset is divided into training set, validation set and test set in the ratio 60:20:20. This is done 10 times with independent random splitting. Each tree remembers what was the partition used for its training and testing (and validation, however validation set is not used in this part). For choosing the best attribute for given split, *residual squared* sums of the target concept is employed. The average test accuracy is reported as the test accuracy of these 10 trees, while the best one is chosen for further tasks. To define the accuracy of a decision tree, we used *mean squared error* over the predicted values and actual values.

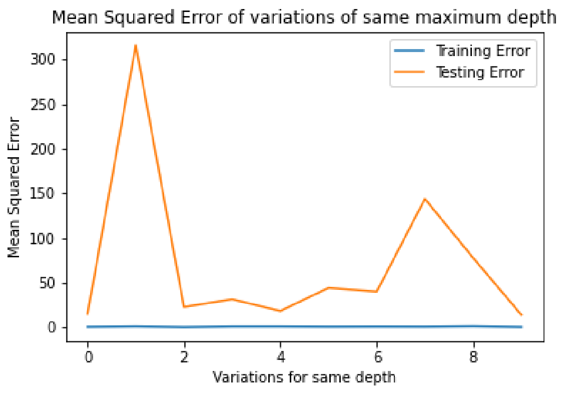


Fig 2. Training and testing error observed over different split of the dataset for same maximum depth (`max_depth` = 6 in this case)

3 Analysing the Model for Different Values of `max_depth`

In this part, the algorithm used in previous part is used for different values of `max_depth`. The range chosen was 1 to 20. The training and test error for each value of `max_depth` is plotted for analysis.

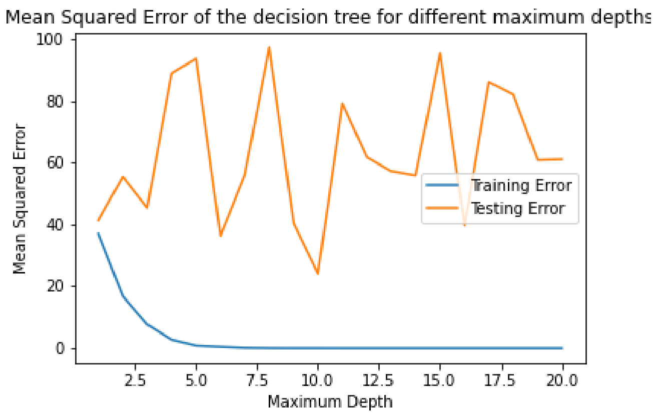


Fig 3. Training and testing error for different values of maximum depth

As it can be observed in the plot, training error decreases monotonically as maximum depth of the tree is increased. However, this is not the case with testing error. Hence, we need to restrict what should be the maximum allowed depth of the decision tree. As per the plot, the optimal maximum depth is 10. Beyond that, the testing error is always greater. (This may change if you run the program again as the splitting is done randomly and independently.) Hence, in our case, best possible depth limit is 10.

4 Pruning the Best Tree over Validation Set

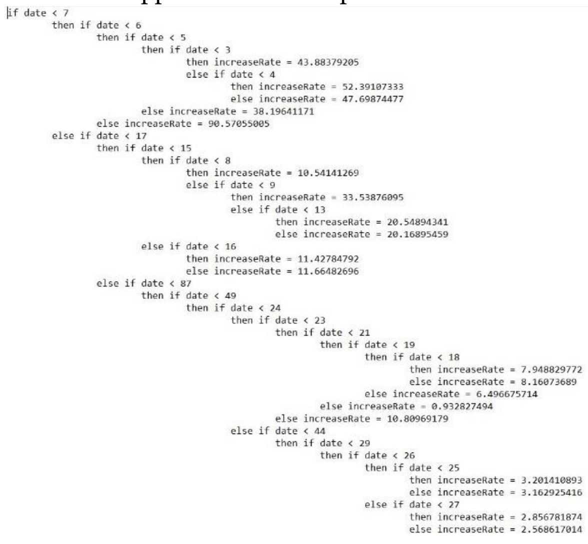
We use the tree which gave the best results (i.e. its depth is 10). It can be further pruned so as to improve its performance. For this case, the validation set is used. Note that it is not necessary that the pruned tree will perform better than the unpruned one over the test set (as pruning is done only on the validation set). For pruning, we are using *reduced-error pruning* algorithm. It is done in top-down manner recursively. For a given node, it checks whether after pruning this node, the error on validation set reduces or not. If it does, the node is pruned by making it a leaf node and its prediction is set to the mean of training instances which were incident on this node. In our case, the unpruned tree performed better on test set as compared to the pruned one. (But the scenario may be different if the program is executed again.)

Table 1. Error of the unpruned and pruned decision tree (*depth* = 10)

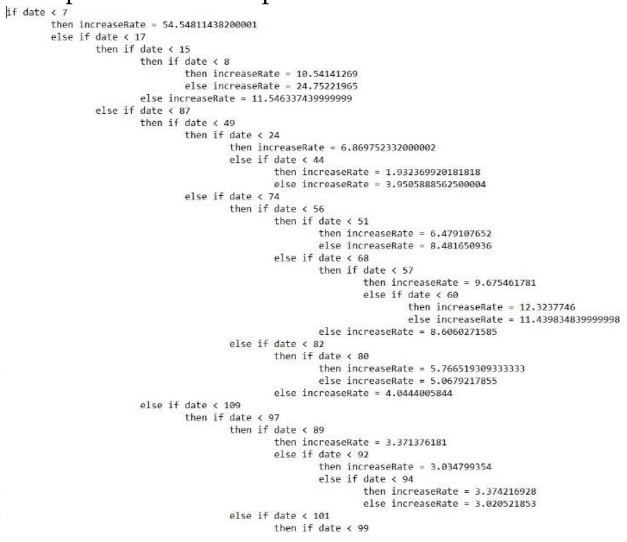
Pruned	**Training Error**	**Validation Error**	**Testing Error**
False	0.006578640592825442 6	31.491522818820886	15.459176608103375
True	14.15782870734966	20.653508126041466	32.293096850435674

5 Printing the Decision Tree

A recursive approach is used to print the decision tree. Some part of the trees is presented below.



```
if date < 7
    then if date < 6
        then if date < 5
            then if date < 3
                then increaseRate = 43.88379205
                else if date < 4
                    then increaseRate = 52.39107333
                    else increaseRate = 47.69874477
            else increaseRate = 38.19641171
        else increaseRate = 90.57055005
    else if date < 17
        then if date < 15
            then if date < 8
                then increaseRate = 10.54141269
                else if date < 9
                    then increaseRate = 33.53876095
                    else if date < 13
                        then increaseRate = 20.54894341
                        else increaseRate = 20.16895459
            else if date < 16
                then increaseRate = 11.42784792
                else increaseRate = 11.66482696
        else if date < 87
            then if date < 49
                then if date < 24
                    then if date < 23
                        then if date < 21
                            then if date < 19
                                then if date < 18
                                    then increaseRate = 7.948829772
                                    else increaseRate = 8.16073689
                                else increaseRate = 6.496675714
                            else increaseRate = 0.932827494
                        else increaseRate = 10.80969179
                    else if date < 44
                        then if date < 29
                            then if date < 26
                                then if date < 25
                                    then increaseRate = 3.201410893
                                    else increaseRate = 3.162925416
                                else if date < 27
                                    then increaseRate = 2.856781874
                                    else increaseRate = 2.568617014
```



```
if date < 7
    then increaseRate = 54.54811438200001
    else if date < 17
        then if date < 15
            then if date < 8
                then increaseRate = 10.54141269
                else increaseRate = 24.75221965
            else increaseRate = 11.546337439999999
        else if date < 87
            then if date < 49
                then if date < 24
                    then increaseRate = 6.869752332000002
                    else if date < 44
                        then increaseRate = 1.932369920181818
                        else increaseRate = 3.9505888562500004
                else if date < 74
                    then if date < 56
                        then if date < 51
                            then increaseRate = 6.479107652
                            else increaseRate = 8.481650936
                        else if date < 68
                            then if date < 57
                                then increaseRate = 9.675461781
                                else if date < 60
                                    then increaseRate = 12.3237746
                                    else increaseRate = 11.439834839999998
                            else increaseRate = 8.6060271585
                    else if date < 82
                        then if date < 80
                            then increaseRate = 5.766519309333333
                            else increaseRate = 5.0679217855
                        else increaseRate = 4.0444005844
            else if date < 109
                then if date < 97
                    then if date < 89
                        then increaseRate = 3.371376181
                        else if date < 92
                            then increaseRate = 3.034799354
                            else if date < 94
                                then increaseRate = 3.374216928
                                else increaseRate = 3.020521853
                    else if date < 101
                        then if date < 99
```

Fig 4. Part of unpruned (left) and pruned (right) decision tree (*depth* = 10)