Group 7
Sagnik Roy (18CS10063)
Suryansh Kumar (18CS30043)

Machine Learning Assignment 1

4th October 2020

OVERVIEW

Given the data about the Confirmed, Recovered and the number of Deaths related to Covid-19 in the world, we will have to build a decision tree to predict the Rate of increase of Covid-19 cases in the world.

GOALS

1. Split the data into 10 random 80:20 Train_Test splits, and choose the split that gives the best test accuracy for further steps.
2. With the data-split obtained in the previous step, grow the tree to maximum depth and find the depth that gives the greatest Test accuracy.
3. Prune the tree with the best-accuracy depth, obtained in the previous step using a statistical test and prune the tree appropriately to obtain better validation accuracy.

PREREQUISITES

Python Inbuilt Libraries:

The python inbuilt libraries used obtaining, modifying and visualizing data are :

- Pandas
- Numpy
- Matplotlib
- Seaborn

Auxiliary Functions :

Initially the following helper functions were created :

- **train_test_split :** It takes a pandas dataframe and test size as input and returns 2 dataframes by splitting the original dataframe into a train and a test dataframe where size of test dataframe = test_size * size of original dataframe
- **check_purity :** It takes a numpy 2d array data as input and checks whether the data input is pure with respect to the label data (target attribute). Since our target attribute is a continuous real valued attribute, it essentially checks for a uni valued data.
- **classify_data :** It takes a numpy 2d array as input and classifies the data to form the leaf node for our decision tree. It returns the mean of all the label values that it takes as input.
- **get_potential_splits :** Given a data as input it returns all the potential values with respect to all the attributes which can be used to split the data. It returns a dictionary in which the keys are the column numbers of the attributes and the values are a list containing all the points with respect to which the data can be split.
- **split_data :** Given a data, a split_column and a split_value it will split the entered data into 2 parts based on the values of the split_column, with respect to the split_values and return the split data as data_below and data_above..
- **calculate_mse :** Given a data, it calculates the variance of the data with respect to the label values, using the formula variance = $(\Sigma(X_i - X_{mean})^2)/n$
- **calculate_overall_mse :** Returns the weighted mean of the 2 data that it accepts as input.
- **determine_best_split :** Given the data and all the potential points at which my data can be split, the function iterates over all the potential splits and will finally return the split so that the resulting split data has lowest overall mse (That is offers the best decrease in randomness of the data)

Procedure

To construct the tree the function **decision_tree_algorithm** is used. The function first checks if the data entered is pure, that it has only labels having a single value. If the data is pure, we immediately classify the data using **classify_data** function. Also if the tree exceeds the maximum height permissible for the tree the data at the maximum height is immediately classified.

If the data cannot be immediately classified using the above constraints, all the potential splits for the data and the best split which gives maximum reduction in mean squared error for the split data is chosen to split the data into 2 halves and this process continues recursively.

The data structure used for building the tree is a nested dictionary that has the following format: **{question: [yes_answer, no_answer]}**. The question is of the type **attribute_name <= value**, the yes_answer and the no_answer can be either another dictionary or a specific value if the data is being classified at that level.

Classifying Unseen examples and Finding Accuracy

To classify an unseen example, the example along with the decision tree that has been built is passed to the function **classify_example**. The function recursively finds the leaf node to which the example belongs based on the values of its attributes. Finally, it classifies the expected value of this example as the value residing in this leaf node.

Finding Accuracy (Hyperparameter Tuning)

The r_squared parameter is used as a measure of accuracy. It is calculated as:

ss_res = $\Sigma(X_i - X_{prediction})^2$

ss_tot = $\Sigma(X_i - X_{mean})^2$

r_squared = 1 - ss_res/ss_tot

Where X_i = Actual values of the labels

$X_{prediction}$ = The value of the label as predicted by our tree

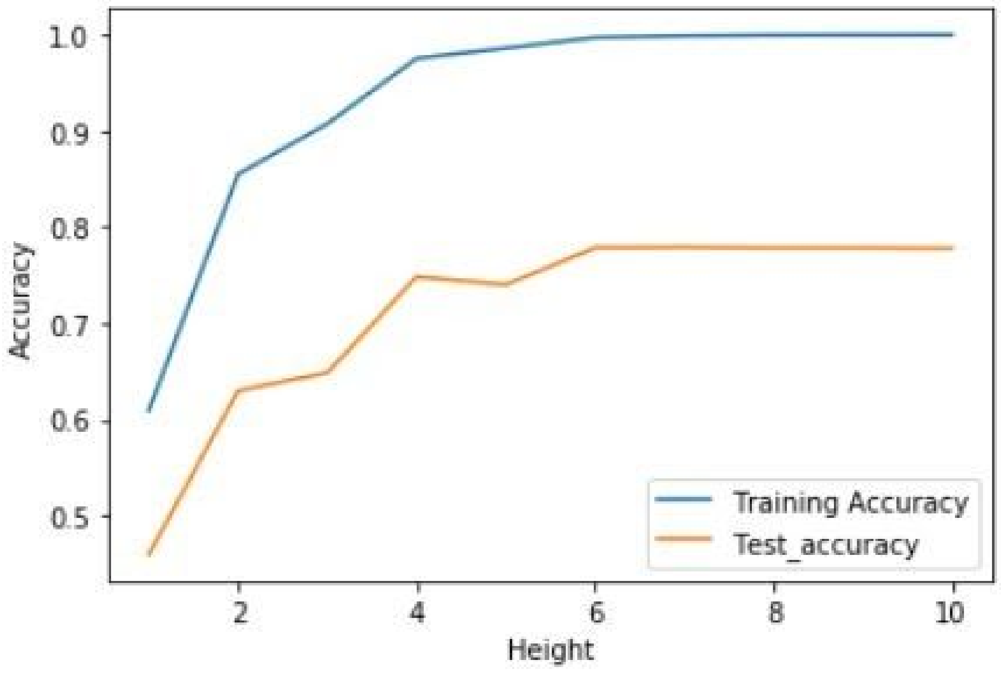
X_{mean} = The mean of the actual values of the labels

Building the Final Tree

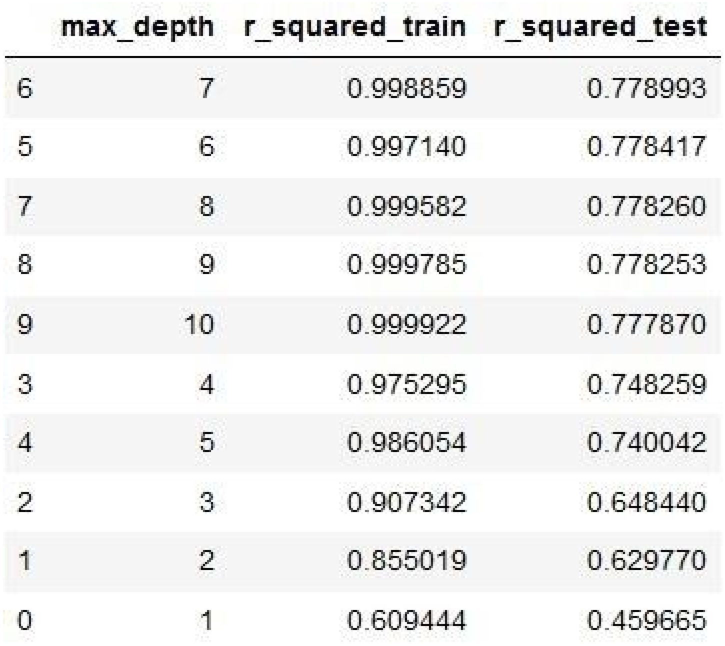
To build the final tree, the **build_tree()** function is used. The function first randomly splits the data into 80:20 (train:test) splits. The train dataset is further split into a training dataset and a validation data set. This gives us finally a 60:20:20 split of training, validation and test data sets. The decision tree is built using the training data set and the accuracy of the tree is computed by classification on the test dataset. The split which gives the best accuracy is then taken forward for the next parts of the assignment.

After obtaining the best split of the train, validation and test dataset, we try to obtain the best maximum depth at which accuracy over the test set is maximum. We create a dictionary in which the key values are **max_depth, r_squared_train (measure of training accuracy), r_squared_test (measure of the test accuracy).** The accuracy versus depth plot helps us to find the depth at which our tree gives maximum accuracy over the test dataset.

The plot obtained is:



In our case, the best test accuracy was reported for the tree at max_height = 7, with the r_squared_train and r_squared_test values against max_depth as:



	max_depth	r_squared_train	r_squared_test
6	7	0.998859	0.778993
5	6	0.997140	0.778417
7	8	0.999582	0.778260
8	9	0.999785	0.778253
9	10	0.999922	0.777870
3	4	0.975295	0.748259
4	5	0.986054	0.740042
2	3	0.907342	0.648440
1	2	0.855019	0.629770
0	1	0.609444	0.459665

So the tree that gives the best test accuracy at max_height = 7. We build this tree which initially gives r_squared_train = 0.998859 and r_squared_test = 0.778993. To avoid overfitting we will now prune the tree of height 7 built earlier based on our validation data set, which is 20% of our original dataset.

Post-Pruning :

After the tree of height 7 has been created, it is now pruned to avoid overfitting. It is done in a bottom up fashion. A node of the tree is taken and it is evaluated that if we delete the subtree rooted at the node, whether it increases the accuracy of our model over the validation dataset. It is performed in the following manner:

prediction = $\Sigma(X_i)/n$ where X_i = values of the train dataset

accuracy_trimmed = r_squared value calculated over the validation dataset using the prediction calculated previously (assuming that the tree is being trimmed at current node).

accuracy_untrimmed = r_squared value calculated over the validation dataset if the tree had not been trimmed at the current node.

If (accuracy_trimmed >= accuracy_untrimmed) we prune the tree at the current node.

So, the same measure that was used to measure the accuracy of the tree is used as the statistical measure to gauge the feasibility of pruning at a particular node.

Using this measure, the tree we generated previously of max_height = 7 is pruned.

Results of the pruning operation:

After the pruning has been performed, we observe that pruning has occurred at 3 nodes and the test accuracy of the final pruned tree measured over our test data set is observed to be:

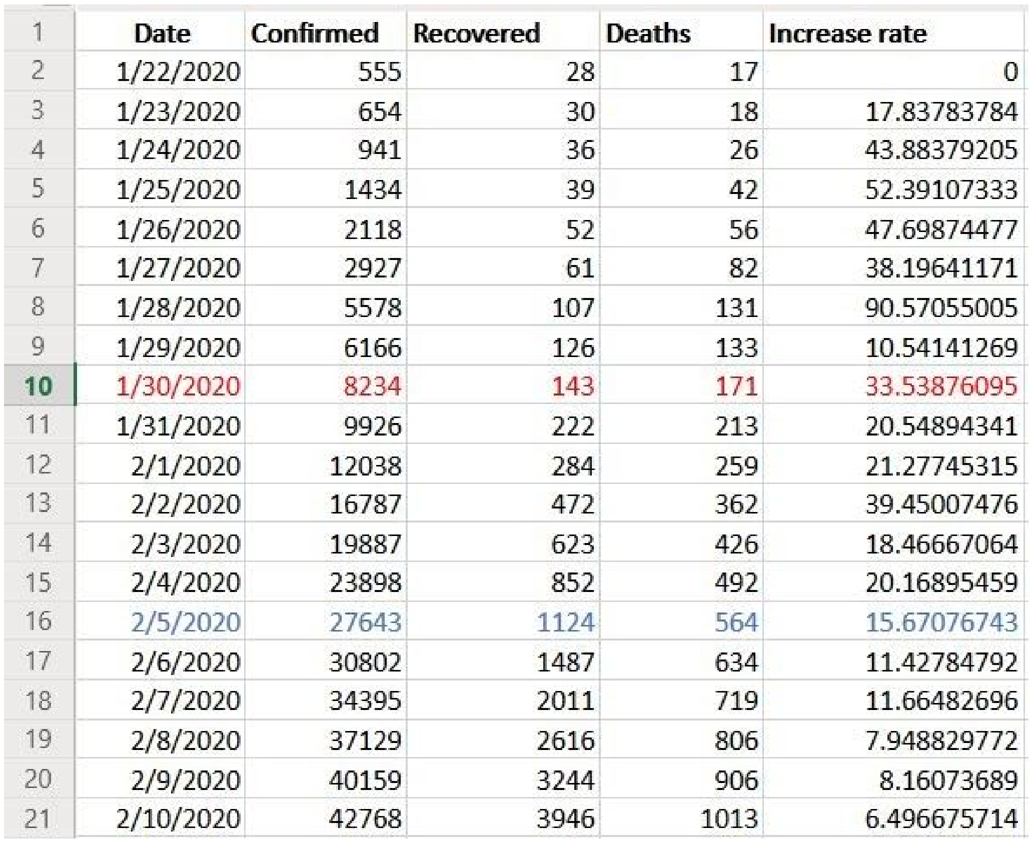
r_squared_test = 0.789235 which is an improvement on the previously observed

r_squared_test = 0.77893 on the original unpruned tree.

One kind of statistical test which we usually do is if accuracy after pruning is more than the accuracy before pruning and then prune it. This is a valid statistical test with type I error = 0.5. This same statistical test is used as a measure to decide whether or not we should prune the subtree rooted at a node.

Some Interesting Observations regarding the Dataset

Following is a snapshot of the csv file that we used as our dataset for the model.



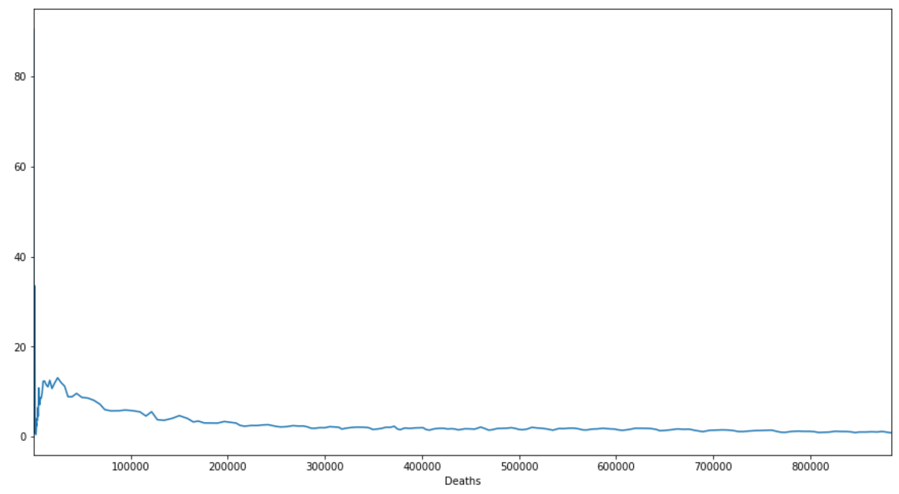
	Date	Confirmed	Recovered	Deaths	Increase rate
1	Date	Confirmed	Recovered	Deaths	Increase rate
2	1/22/2020	555	28	17	0
3	1/23/2020	654	30	18	17.83783784
4	1/24/2020	941	36	26	43.88379205
5	1/25/2020	1434	39	42	52.39107333
6	1/26/2020	2118	52	56	47.69874477
7	1/27/2020	2927	61	82	38.19641171
8	1/28/2020	5578	107	131	90.57055005
9	1/29/2020	6166	126	133	10.54141269
10	1/30/2020	8234	143	171	33.53876095
11	1/31/2020	9926	222	213	20.54894341
12	2/1/2020	12038	284	259	21.27745315
13	2/2/2020	16787	472	362	39.45007476
14	2/3/2020	19887	623	426	18.46667064
15	2/4/2020	23898	852	492	20.16895459
16	2/5/2020	27643	1124	564	15.67076743
17	2/6/2020	30802	1487	634	11.42784792
18	2/7/2020	34395	2011	719	11.66482696
19	2/8/2020	37129	2616	806	7.948829772
20	2/9/2020	40159	3244	906	8.16073689
21	2/10/2020	42768	3946	1013	6.496675714

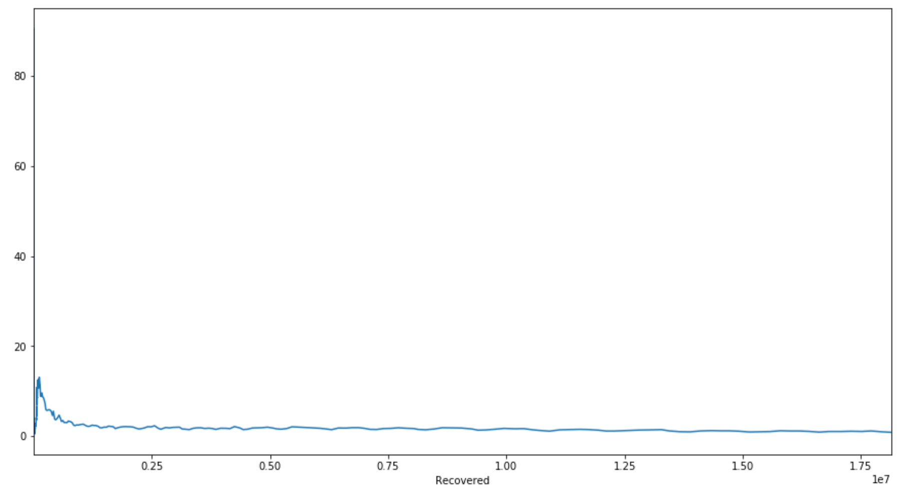
A quick glance over the data gives us the following idea:

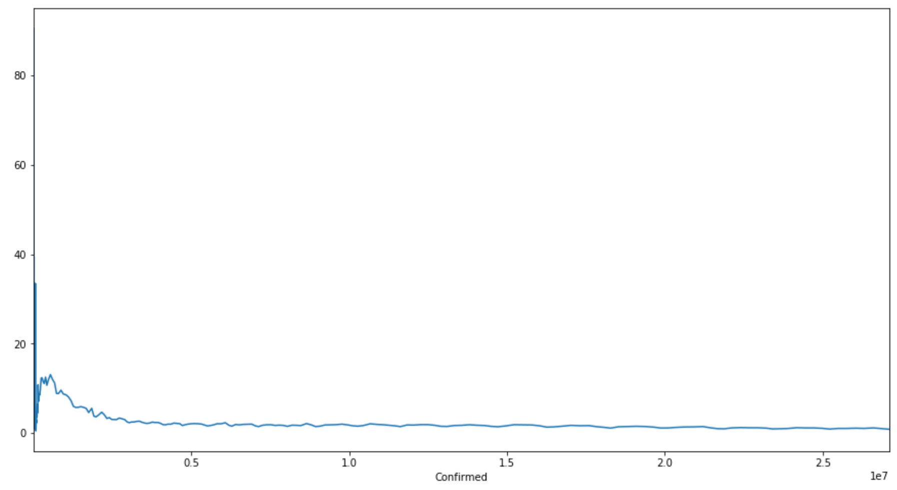
It is clearly visible that the data in all the attribute columns which we use to train our data is monotonically increasing. So the potential split and find best split function will behave similarly for all the 3 attributes. Say, for example, the above snapshot is the data under consideration and the best split occurs for the Confirmed attribute on 30th January 2020 (Highlighted in Red). However, if we split the data with respect to the Recovered or even the Deaths attributes, it would give same mean squared error as for both of them the split of data would be exactly identical, ie, Confirmed<=8234, Recovered<=143 and Deaths<=171 will all give exactly same data splits and hence same mean squared error. The same thing can be said for any other day as well, like 5th February 2020 (Highlighted in Blue). Thus if we want, we can actually build the entire decision tree using only the values of a single attribute as all the attributes will behave similarly.

So, to ensure at least 2 attribute values appear in the decision tree we construct, we have alternated between < and <= in the find best split function in our construction.

The above fact becomes even more clear when we look at the plot of Increase Rate vs the other 3 attributes, which is almost same for all 3 cases **(y-Axis: Increase Rate)**







These plots clearly show that the target attribute (Increase Rate) behaves exactly similar with respect to each of the 3 other attributes. So classification of data obtained using any one r any combination of the three attributes will not show any change in behaviour whatsoever.

Conclusion

The initial tree was built using maximum depth parameter = 6. This was used to obtain the train_test split giving maximum test accuracy. This split was then used to get the best_depth parameter at which the tree gives us best test accuracy.

The best depth was found to be 7. At max_depth = 7, the tree gave the following parameters:

r_squared_train = 0.998859

r_squared_test = 0.778993

After this the obtained tree was pruned to avoid overfitting of the data. For this accuracy the validation set was tested. After pruning operation is over the test accuracy was observed to be:

r_squared_test = 0.789235

which is an improvement on the test accuracy obtained using the unpruned tree.

The final tree which we got is also attached as a pdf file in the assignment submission. It can also be found as a doc file in this link.