UNSW
AUSTRALIA

COMP9417 21T2 Project Report

Kaggle Project: COVID19 Global Forecasting

Group	87
Zhehan Zhang	z5241150
Yuxuan Huang	z5274414
Lu Qi	z5275529
Sifan Yin	z5230358

1. Introduction

The COVID-19 pandemic is the defining global health crisis of our time. The first known case was identified in Wuhan, China, in December 2019. The disease has since spread worldwide, leading to an ongoing pandemic. The task of this challenge for us is to predict daily COVID-19 spread in regions around the world. In this report, all data comes from the project folder. First, the train data set and test data set should be separated from source data. Data analysis will then be conducted, it includes data cleaning and feature selection. We also add some new data to the train set as required by the project. We add some new columns to the train set to make it more accurate. Although the data under the project folder is separated into train set and supplementary data set[1], observations are based on the whole data. After data analysis, we tried different algorithms and adjustment parameters. Then, we compared mean absolute error (MAE), median absolute error (medainAE), r2_score in order to find the best model. Finally, we got the best model and made predictions of data under the project folder and analysed the result.

For data in the project folder, to find the best model, we compared three parameters. Also, we separated data into train confirm case dataset and train fatalities dataset. We try nine basic algorithms on it, linear regression, K-Nearest Neighbor, random forest, XGBoost, Gradient Boosting Regressor, decision tree, Multilayer Perceptron, Ada Boost Regressor and LGBM Regressor. Random forest achieves the best performance among all the models and gets a r2 score of 0.91, medianAE of 0.09, MAE is 8.84.

2. Implementation

First, we get the train and test dataset. After we read the dataset, the task for us is not only to produce accurate forecasts but also to identify factors that appear to impact the transmission rate of COVID-19. There are many cases to influence it, so we decided to add some features to find the possible factors. After merging some features, we got a new dataset. We also need to do some data cleaning for the dataset. We need to drop some useless columns and add some values to fill columns whose values are null.

To train the model and find the best model, we used nine algorithms to do that. We got three indicators to scale those algorithms. To find the best model, we compared the r2 score which is close to 1, MAE and medianAE are close to 0. Finally, we find the best model is random forest. Here are some introductions of some parts of these nine algorithms.

XGboost: XGBoost stands for extreme Gradient Boosting. XGBoost is a gradient boosting library with focus on tree model, which means inside XGBoost, there are 2 distinct parts: 1. The model consisting of trees and 2. Hyperparameters and configurations used for building the model.[2]

LGBM Regressor: Light GBM grows tree vertically while other algorithm grows trees horizontally meaning that Light GBM grows tree leaf-wise while another algorithm grows level-wise. It will choose the leaf with max delta loss to grow. When growing the same leaf, Leaf-wise algorithm can reduce more loss than a level-wise algorithm.[3]

There are also some other algorithms, and many of them are tradition machine learning algorithms. Linear regression is a linear approach to modelling the relationship between a scalar response and one or more explanatory variables.[4] Randomforest is an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time.[5]

3. Experimentation

3.1 Data Analysis & Data Cleaning

When we are not familiar with this data set, we decide to print out the first 5 rows and the last 5 rows of this data set to see what features and data types and the overall characteristics of the data are. Furthermore, through the info and describe methods of the DataFrame, we learn that the original data set has a total of 9 features and 969,640 rows of data.

Firstly, we observe that this data set may have some null values, but we don't know what this magnitude is yet. Through using *isnull()* function, we can calculate the number and proportion of missing values, as well as the column in which they are

located. These details are shown in table 1.

Table 1

	Id	County	Province_State	Country_Region	Population	Weight	Date	Target	TargetValue
missing data	0.0	89600.000000	52360.000000	0.0	0.0	0.0	0.0	0.0	0.0
missing proportion * 100%	0.0	9.240543	5.399942	0.0	0.0	0.0	0.0	0.0	0.0

It can be seen that the magnitude of the missing value is still not small. We need to deal with these missing values to make the data set complete. Our idea is that from *County* to *Province_State* to *Country_Region*, the geographical dimension is gradually increasing. Since there are no missing values in the *County* column, we can use the values of the larger dimension to fill in the values of the smaller dimension. To be more specific, in the first step, we fill in the *Province_State* based on *Country_Region*. Doing so can ensure that the *Province_State* column has no null values. In the second step, we fill *County* based on *Province_State* to ensure that *County* has no empty values. After these two steps, the entire data set should have no missing values.

Secondly, we see that there are many characteristics type data. To study the proportion of available numerical data, we plot a pie chart (see Figure 1 below) to observe the distribution of these data types. As can be seen from the figure, more than half of the data types are non-numerical data in this data set, which will apparently affect the training of the model. Therefore, we consider converting these character type data into numeric type data.

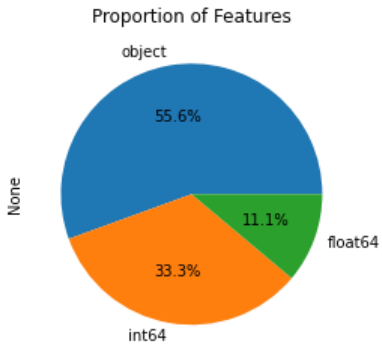

Figure 1

Based on the information given by the *info()* function, there are 5 non-digital types features: *County*, *Province_State*, *Country_Region*, *Date* and *Target*. For the feature of

Date, the data itself contains numbers. We think it can be directly converted into a digital format. The specific method is to remove the hyphens in the character date and unnecessary hours, minutes, and seconds, and convert the remaining year, month, and day into an integer data type. The examples can be seen below (Figure 2):

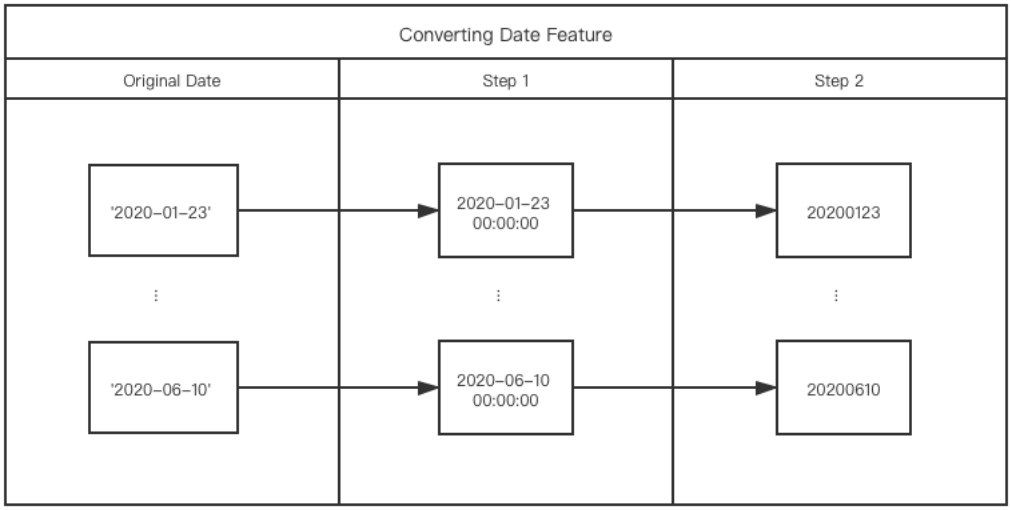

Figure 2

For the other 4 features, our idea here is to classify the same data in each feature dimension and encode these classifications with numbers from 0 to N-1. Considering that the amount of data is relatively large, manual encoding is not realistic, so we use the *LabelEncoder()* function to automatically complete the mapping from discrete character data to numbers.

The above is the data analysis and preprocessing of the original data set. The newly added data set has a total of 6 columns, 4 of which are numeric data. One of the other two columns is the *Lockdown_Date*, and we deal with it in the same way as the *Date*. Another feature that needs to be processed is the *Lockdown_Type*. Similarly, we use *LabelEncoder()* function to automatically encode. The results of the encoding are shown in the Figure 3:

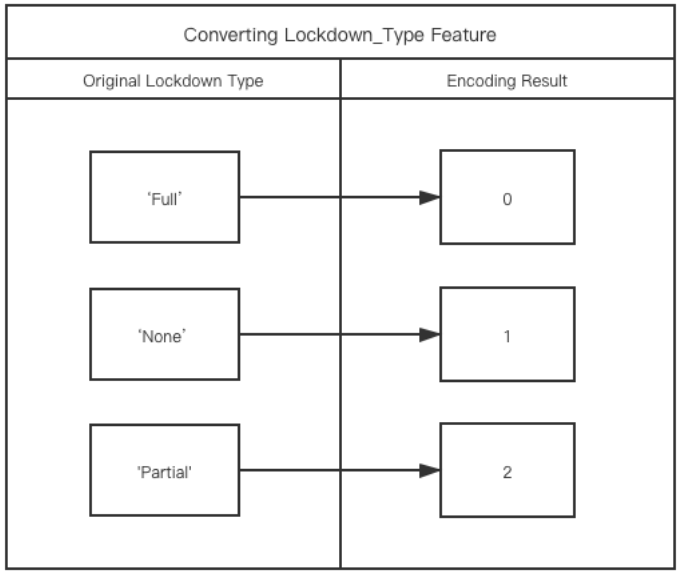

Figure 3

3.2 Feature Engineering

After data analysis and data cleaning, feature engineering should be considered and implemented. First, based on these existing feature dimensions, we have derived some features that we think may affect the rate of virus transmission. For example, based on the dimension of *Date*, we calculated the *Diff_Days*, that is, the date difference between the subsequent confirmation date and the first confirmation date. At the same time, we calculated the new feature *Avg_Population_by_County* based on *County* and *Population*. This is because we feel that narrowing the population to the smallest geographical dimension can help more accurately predict the number of confirmed cases and the number of fatalities. Another thing worth mentioning is that we calculated the number of days of lockdown through the *Lockdown_Date* to generate a new feature *Lockdown_Days*.

Finally, the selection of features for model training and prediction is determined based on factors such as the meaning, data type, and Pearson correlation coefficient (Wikipedia 2021) [6] of the feature itself. The following Table 2 shows some factors of the considerations.

Table 2

Features	Meaningless	Data Type	Strong correlation
Id	index	int	+
County	county under a province/state	string -> int	-
Province_State	province under a country/region	string -> int	+
Country_Region	contry/region	string -> int	-
Population	overall population of contry/region	int	+
Weight	coefficient for the number of confirmed cases or fatalities	float	+
Date	recorded ate	string -> int	-
Target	predict confirmed cases or fatalities	string -> int	+
TargetValue	the number of confirmed cases or fatalities	int	-
Diff_Days	the date difference between the subsequent confirmation date and the first confirmation date	int	-
Count_County	the number of counties under a certain province/state	int	-
Avg_Population_by_County	average population in a certain county	float	+
Tourism	the number of touism	int	-
Latitude	the latitude of a certain country	float	-
Longtitude	the lontitude of a certain country	float	-
Mean_Age	the average age of the population in a country	float	-
Lockdown_Date	the date of lockdown	string -> int	-
Lockdown_Type	none/partial/full lockdown	string -> int	-
Lockdown_Days	the days of lockdown	int	-

The Figure 4 below shows the Pearson correlation coefficient of each feature.

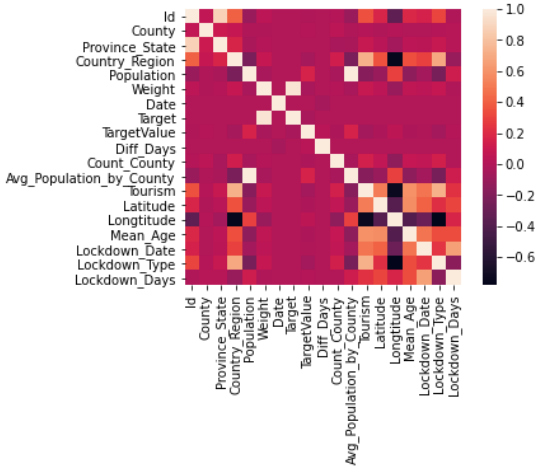

Figure 4

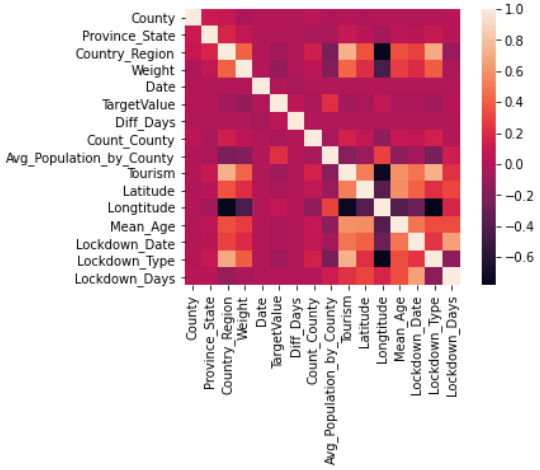

Figure 5

It can be seen that the *Id* feature is meaningless and should not be used for model training and testing. *Population* and *Avg_Population_by_County* has high correlation and *Avg_Population_by_County* is more reasonable, so we drop feature *Population* but keep feature *Avg_Population_by_County.* Similarly, we drop *Target* but keep *Weight*. In addition, *TargetValue* should be removed from features and saved as a label since it is what we need to predict.

Furthermore, we are asked to predict two indicators. One is the number of confirmed cases and the other is the number of fatalities. Therefore, we think it is better to separate these two from the original data set. When predicting fatalities, the features

Latitude and *Longitude* are meaningless and will be discarded. The Figure 5 above shows the Pearson correlation coefficient of final features.

Hence, the final features for confirmed cases are *County*, *Province_State*, *Country_Region*, *Weight, Dat*e, *Diff_Days*, *Count_County*, *Avg_Population_by_County*, *Tourism*, *Latitude*, *Longtitude*, *Mean_Age*, *Lockdown_Date*, *Lockdown_Type*, *Lockdown_Days*. In terms of the fatalities, the final feature has all the features mentioned above in addition to the *Latitude* and *Longitude*.

3.3 Modeling & Hyperparameters Tuning

Firstly, all models are using the default parameters to train. Although some model scores are not bad, we try to adjust some parameters manually. Then we found that if we choose the suitable parameters, the models' performance will be better, therefore, we use different ways to adjust, including GridsearchCV and reading the documents.

For KNN, there are two main hyperparameters that affect the results of the prediction, n_neighbor and weights. At first, we try to use the default parameters to train the KNN model, we got a nice r2 score that is 0.82. Then we use check the KNN hyperparameters [7], we found that if we use different n_neighbors values, the model could have better performance. Therefore, we use the GridsearchCV to help us find the best parameters. Finally, with the help of GridsearchCV function, it is found that n_neighbors is equal to 10, the knn model performs best. In addition, the weights will affect the model performance. Weight function used in prediction, the default values is "uniform" that means all points in each neighborhood are weighted equally. After we try, the "distance" could increase the r2 score, therefore, we choose to weight points by the inverse of their distance.

For random forest regressor, there are many different parameters can be changed, for example, 'n_estimators', 'min_samples_leaf', 'min_samples_split', etc. At first, we want to use GridsearchCV again, however there are too many parameters, and the train data is too large, the running time is too long, adjusting parameters manually. We checked the document about random forest, we increased n_estimators values (from

the default values 100 to 500), however, the it takes too much calculation and the score did not increase significantly, the default value 100 is the most suitable. Furthermore, because we have a large train data, we choose to increase the value “min_samples_leaf” and “min_samples_split”, which can help us to get better performance random forest model.

For the Gradient Boosting Regressor, there are many parameters we can change. We first tried to change the learning rate parameter, and find that if it is increased from 0.1, the score will increase as well. So, the learning rate is increased to 0.5, but if it is over 0.5, the score will decrease. We also tried to change other parameters like n_estimators, alpha, min_weight_fraction_leaf and so on, but the performance is not better.

For MLP Regressor, this model is not good. The score is always below 0. We changed all the parameters in it, but the score is still below 0. So, we think this is not a suit able model. MLP is not worth to compare with other methods.

Adaboost is implemented by changing the data distribution. It will group all the weak regression learner which are trained for the same data set together to generate a strong regression learner. AdaBoostRegressor implements the Adaboost.R2 regression algorithm. The parameters of AdaBoostRegressor are base_estimator, n_estimators, learning_rate, loss and random_state. Base_estimator is the weak regression learner, the default is DecisionTreeRegressor, here we set the default value. Parameter n_estimators is the maximum number of iterations of our weak learner. If this value is too small, it is easy to underfit, and if it is too large, it is easy to overfit. Parameter loss has ‘linear’, ‘square’ and ‘exponential’ three options. After testing, setting the default value ‘linear’ would be better. Parameter learning_rate is the weight reduction factor for each weak learner. Finally, it can be found that when random_state=0, n_estimators=50 AdaBoostRegressor perform best. However, the results of AdaBoostRegressor are still too low compared to other algorithms.

As one of the most popular models in Kaggle competitions, XGBRegressor has many parameters can be tuned. Parameter eta is similar to learning rate, by reducing the weight of each step, the robustness of the model can be improved. Parameter gamma

specifies the minimum loss function drop value required for node splitting. The larger the value of this parameter, the more conservative the algorithm. Parameter max_depth is the max depth of tree, which is used to avoid overfitting. Parameter min_child_weight determines the sum of sample weight of the minimum leaf node. Parameter subsample controls the random sampling ratio of each tree. Parameter alpha can make the algorithm faster in the case of very high dimensions. Through testing, XGBRegressor performs well when setting colsample_bylevel=0.9, subsample=0.7, max_depth=5, min_child_weight=1, alpha=20. It is worth mentioning that it is indeed superior to many other algorithms.

For decision trees, there are many parameters that can be adjusted, such as criterion, splitter, max_features, etc. But in our method, only the following main parameters are considered to improve the generalization ability of the decision tree (see Table 3).

Table 3

max_depth	The maximum depth of the tree. When the depth of the tree reaches max_depth, no matter how many features can be branched, the decision tree will stop computing.
min_samples_split	The minimum number of nodes required for splitting. When the number of samples of leaf nodes is less than this parameter, no more branches will be generated
min_samples_leaf	The minimum number of samples required for a branch. If after the branch, the number of feature samples of a new leaf node is less than the hyperparameter, it will be returned and no more pruning.
min_weight_fraction_leaf	The smallest weight coefficient
max_leaf_nodes	The maximum number of leaf nodes, when taking an integer, max_depth is ignored

In the process of adjusting the parameters, we used GridSearchCV to find the appropriate parameters. Finally, we found that when criterion='mse', max_depth=75, min_samples_split=5, min_samples_leaf=2, the decision tree achieves the best performance.

At the same time, we focused on the two parameters min_samples_leaf and min_samples_split. The following figure record the influence of min_samples_leaf and min_samples_split on the decision tree model.

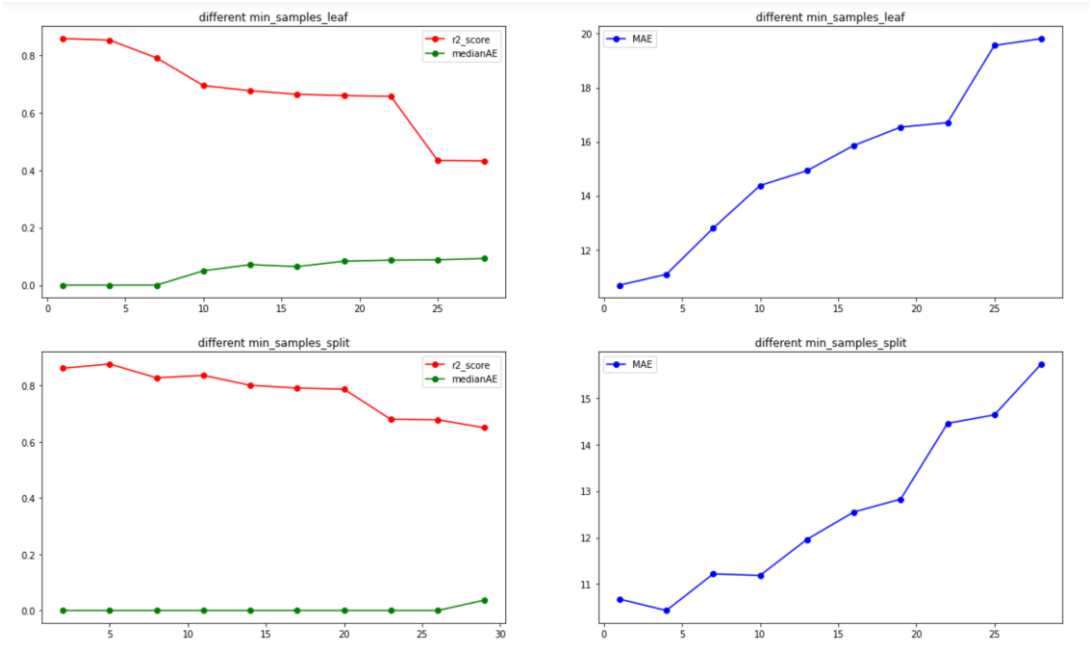

Figure 6

It can be seen from the Figure 6 that as these two parameters become larger, the fitting degree of the decision tree model will decrease, and MAE and medianAE will increase. One of the possible reasons is that the decision tree model is more susceptible to noise.

In the LGBM model, there are the following main parameters: learning_rate: the step length of gradient descent, max_depth: the depth of the maximum tree, n_estimator**s**: the tree of the fitted tree, that is, the number of training rounds.

Similarly, we used GridSearchCV to find the appropriate parameters. A good result was finally reached. Its parameters were n_estimators=1300, max_depth=50, learning_rate=0.01,num_leaves=32.

In this model, we mainly focus on n_estimators, because generally speaking, if n_estimators is too small, the model is prone to under-fitting, but if n_estimators is too large, the model is prone to over-fitting. Generally, we choose a moderate value. The default is 100. The following figure records the impact of changes in n_estimators on the model.

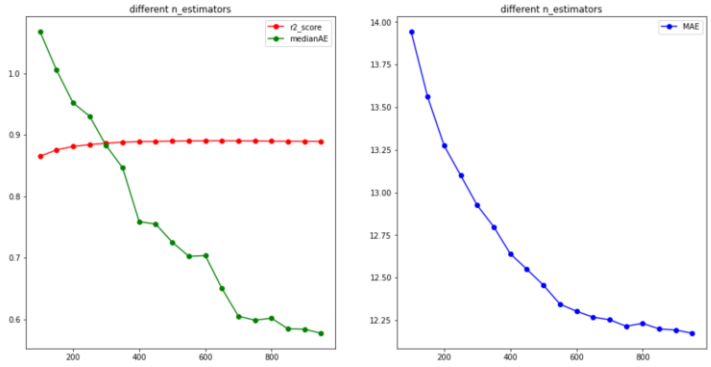

Figure 7

It can be seen from the Figure 7 that as n_estimators increases, the model fits close to 0.9 and remains stable, and the error will decrease as n_estimators increases, so setting a larger value for n_estimators is better, but After more than 1500, the running speed will be very slow.

3.4 Result

For the nine basic models mentioned above, we have done a horizontal comparison, and the evaluation criteria include r2_score, mean absolute error and median absolute error. The results are as follows (Figure 8 & Table 4).

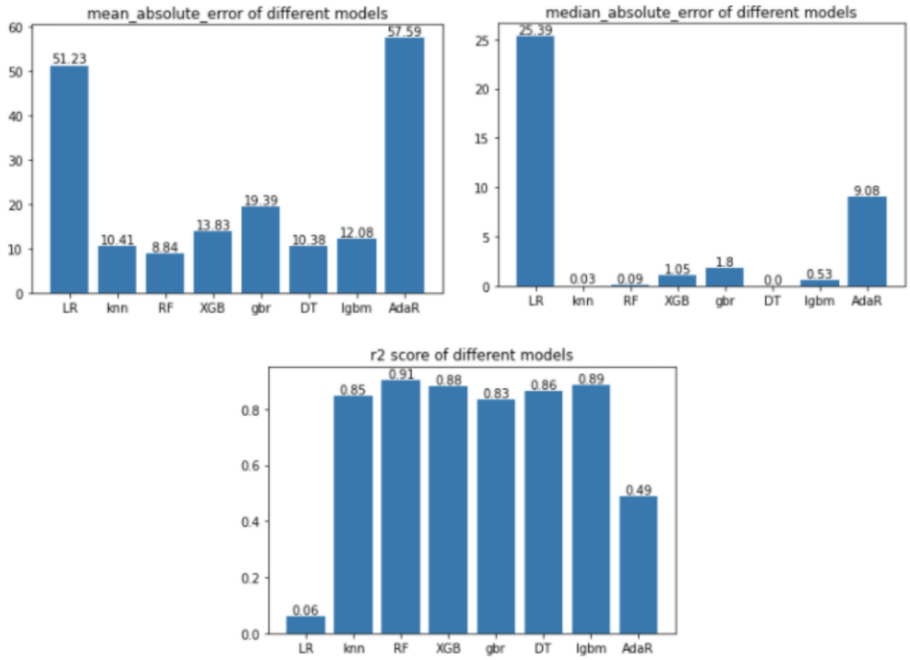

Figure 8

Table 4

	LR	knn	RF	XGB	Gbr	DT	Lgbm	AdaR
MAE	51.23	10.41	8.84	13.83	19.39	10.38	12.08	57.59
medianAE	25.39	0.03	0.09	1.05	1.8	0	0.53	9.08
r2 score	0.06	0.85	0.91	0.88	0.83	0.86	0.89	0.49

Obviously, the LinearRegression model has the worst effect, and its r2 score is just 0.06. This is because LinearRegression is used for linear regression. In this project, the number of infections that need to be predicted and the feature value are not linear. The performance of AdaBoostClassifier is also worse, with 0.49 in r2 score.

On the contrary, we found that RandomForestRegressor is the most suitable model, with the highest degree of fit (0.91), and the MAE (8.84) and medianAE (0.09) are very low. In addition, the performance of the remaining models is good, reaching 0.8, with low errors.

4. Conclusion

During the project, we met with many difficulties. Firstly, the original dataset feature is a few, the models cannot predict well. Then we found a supplement data that enriches the original dataset. In addition, there are several dates in dataset, our group have different opinions on how to deal with the data, finally we decided to calculate the number of days between the two dates as a feature after discussion, and the result is not bad.

Through this project, we have a much better understanding of machine learning. One important is data cleaning and features extraction. Selecting suitable feature and correct processing method can greatly improve the accuracy of the model, especially for large dataset, if the feature extraction is not suitable, it will waste much time. In addition, selecting and comparing different models is important too. Although they are all regression model, the results of different models will be very different. For example, we found MLP is not suitable for our dataset, but random forest regressor perform well.

On the other hand, as for the tunning, it can help us increase the model performance, but it is not as important as features extraction. Sometimes even if we put a lot of effort into tuning the parameters, the results will not improve very much. However,

we do slightly feature extraction, the result could improve significantly. For example, dealing with the time data and null values instead of dropping these data directly.

To conclude, data and features determine the upper limit of machine learning. Models and algorithms only approach this upper limit.

5. Reference

[1] COVID19_Forecasting, accessed 20th July, <https://www.kaggle.com/pelinay/covid19-forecasting?select=Countries_usefulFeatures.csv >

[2] XGBoost Documentation, accessed 21st July, <https://xgboost.readthedocs.io/en/latest/index.html>

[3] What is LightGBM, How to implement it? How to fine tune the parameters?, accessed 26th July, <https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc>

[4] Linear Regression Wiki, accessed 28th July, <https://en.wikipedia.org/wiki/Linear_regression>

[5] Random Forest Wiki, accessed 25th July, <https://en.wikipedia.org/wiki/Random_forest>

[6] Wikipedia 2021, accesses 28th July, <https://en.wikipedia.org/wiki/Pearson_correlation_coefficient>

[7] Knn Documentation, accessed 20th July, <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html>