COVID-19 Time to Hospitalization

Predict the time between symptom onset and hospitalization of COVID-19 confirmed cases

Ao Tang (301297684)

Xiaoliang Zhang (301297782)

Zhengtang Lin (301310788)

Junchao Liu (301299668)

GitHub：https://github.com/chrisliu0424/stat440-Covid-19.git

October 16th, 2020

Abstract

In this project, we tried to predict the duration between new COVID-19 patients between symptom onset and hospitalization. In our data preprocessing, we used one-hot-encoding for all the categorical variables, and tried some special data engineering to expand our training data. Moreover, we fitted our data with different models, and selected the best three models to investigate further. Finally we got a Random Forest model with some hyperparameter optimizations as our best model.

Introduction

Table 1 shown below was the sample data we used for this project. The Explanatory variables were age, sex, city, province, country, V1, confirmed, and symptoms. We had 2 numerical variables(age and confirmed), 6 categorical variables (sex, city, province, country, V1, and symptoms) and most of them were in the form of Unicode.

	age	sex	city	province	country	V1	confirmed	symptoms	outcome	duration
1	68	8a467	ba1b5	89dd9	59dcd	35843	07.02.2020	nausea; nomiting		3
2	25	8a467	44886	fe869	59dcd	d68ec	28.01.2020	pneumonitis		3
3	73	8a467	b4ff4	38fc4	16725	35843	02.02.2020	respiratory symptoms	Recovery	5
4	20	d516d	07f3f	8ac5e	59dcd	35843	02.02.2020		Recovery	0
5	50-59	8a467	38fc4	38fc4	c263d	d68ec	04.02.2020	fever		32
6	77	8a467	38fc4	d7cac	59dcd	b6ab9	29.01.2020			1

Table.1: Preview of the original dataset

Data Pre-processing

Firstly, we converted the column ‘confirmed’ to numeric. For the missing value in these two numerical columns, we filled them in with their means. Dealing with the symptoms column, we did some string manipulation and extracted each individual symptom. Also, we noticed that there were some factor levels in the testing set but did not appear in the training set. We solved the problem by converting all categorical variables into indicator variables and set the levels as the union of levels in the training and testing set. After this preprocessing, we obtained a new training set that was composed of 122 columns and a new test set with 121 columns.

As Figure 1 and Table 1 show below, we could clearly see that the frequencies were quite different for the country column in our dataset. Some had a very low frequency and we did not want to include them as our indicator columns. For example, countries f99a5, 992f7, and 57155 had only one observation, so it had a very high variance and future observations in these countries could be easily affected by these few observations. The solution we found was setting an extra indicator column country.others, to group up all low frequency countries and indicate them as others. We did the same for all the categorical variables(except symptoms). After all these preprocessing, we ended up with 33 columns in training data and 34 columns in test data.

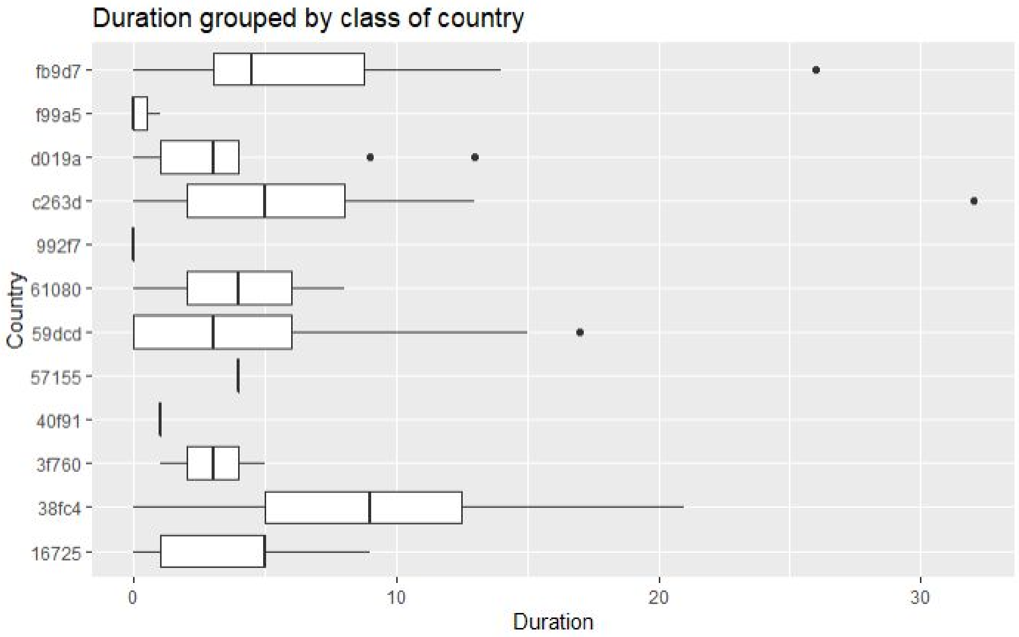

Figure 1: Duration Grouped by Class of Country

16725	38fc4	3f760	40f91	57155	59dcd	61080	992f7	c263d	d019a	f99a5	fb9d7
5	7	2	1	1	134	2	1	24	9	3	30

Table 2: Cross Tabulation for The Counts of Country and Duration

Since we had a small training set, we wanted to do some data augmentation to expand our training size. The procedures as followed:

1. Fit a Linear Regression using lm(duration ~ age + confirmed).
2. Copy the old training data to a new DataFrame new_train.
3. For age and confirmed columns in new_train, age = age +1 and confirmed = confirmed + 3600(1 day).
4. Replace the duration in new_train by the prediction using the model fitted in 1 and new_train data.
5. Combine the train and new_train data.

By doing this data augmentation, we were able to get a new training set with 438 rows. And we simply tested the data with the Random Forest model, the public score on Kaggle improved from 4.39 to 4.33.

Model fitting

After some data inspection and visualization, we suspected that some of our predictors were not very correlated with our response variable. So in this model fitting procedure, we would compare the model performances using two model selection techniques: Stepwise BIC and LASSO. Also with three supervised learning methods: Linear Regression, Ridge Regression, and Random Forest. Moreover, we would use cross-validation to evaluate the model performances based on the Mean Square Predicted Error (MSPE). Noted that LASSO would select two optimal lambdas for the model by its internal Cross-Validation, we used both lambda.min and lambda.1se as two different LASSO models and saw how they performed.

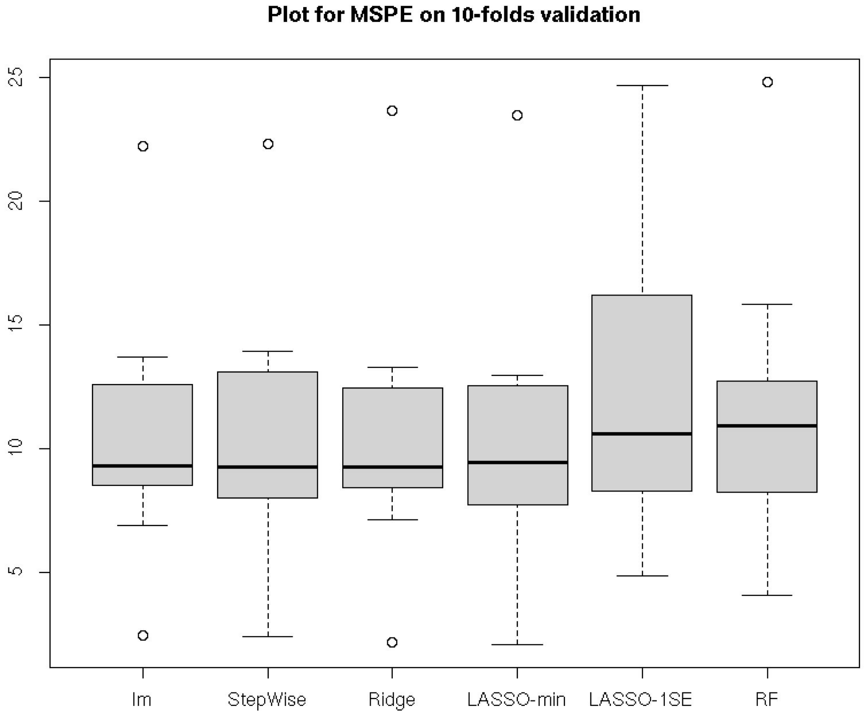

Figure 2: MSPE for CV

The MSPE for each model was shown in Figure 2, the performances were very similar for all models, but we also noticed that the variability for these MSPEs were quite large. That meant the value of MSPEs varied a lot from fold to fold. After taking a deeper look into the LASSO-1SE model, we found that in every fold LASSO-1SE had eliminated all variables and kept only the intercept. So the model was basically using the mean duration of the training set in that fold to be the predicted values for the validation set. Comparing the performance of LASSO-1SE to other models, we could say that the correlation between predictors and responses were little in the provided dataset. In addition, the best model chosen by Cross-Validation did not have the lowest (or second lowest) MSE in Kaggle's Public Leaderboard when comparing to other models using the same data most of the time. So we could conclude that Cross-Validation is not remarkably useful in selecting the best model, because of the limited sample size and the weak correlation between our predictors and response. We would use Kaggle's Public Leaderboard as an unbiased estimate for the Private Leaderboard.

Model evaluating

After Fitting all 6 models with different sets of parameters, we chose 3 of them that seemed to have a lower MSPE to do hyperparameter tuning and ensembling. The three methods we chose were Random Forest (RF), Ridge Regression, Partial Least Square Regression (PLS). The score of the public leaderboard for these three models were 4.30689, 4.43352, and 4.52113, respectively.

Test	mtry	ntree	Score(public)
1	20	700	4.35295
2	10	400	4.29110

3	10	300	4.28436
4	10	200	4.28880
5	9	300	4.28705
6	11	300	4.29627

Table 3: Tuning parameters and scores for RF

First, we conducted a set hyperparameters tuning with RF, as in Table 3, our best score for the competition appeared in the third row, which we scored 4.28436 with mtry = 10 and ntree=300. We used all the predictors in the processed data, because we learnt that RF was not too afraid of overfit, and it could handle models with many predictors relatively well.

Test	W_{RF}	W_{Ridge}	W_{PLS}	Score(Public)
1	0.4	0.3	0.4	4.36047
2	0.5	0.2	0.3	4.33466
3	0.6	0.2	0.2	4.32188
4	0.6	0.3	0.1	4.32081
5	0.7	0.3	0	4.31067
6	0.7	0.2	0.1	4.31177
7	0.8	0.2	0	4.30435

Table 4: MSPE for the ensemble method

We would now use our best RF model with ridge regression and PLS to build an ensemble model, hoping that it would have a better prediction for our response. We tried the bagging ensemble methods with Cross-Validation (CV), but almost all of the time, one of the three estimators was negative. Due to this reason and the instability of CV in our dataset, we would tune the ensemble parameters manually.

We had shown part of the ensemble results in Table 4, after many testings, the ensemble models could not perform as good as our best RF, so we would use the best RF as our final model.

Conclusion

In this competition, since the correlation between predictors and response were very weak, and with limited sample size. We considered data preprocessing plays a more important role than finding the best model. So we spent most of our time cleaning the data and trying to expand the dataset. Although we used Kaggle's public leaderboard as an unbiased estimate for the private score, we could not totally rely on the public score. Because with a dataset as we had in this competition, the correlation was weak and we might potentially get a slightly better or worse score just because of luck, but getting a better Public Score will have more chance to result in a better private score generally. With the preprocessing we did and the best RF we got, we were able to get a 10.4% improvement from our baseline model(from 4.78220 to 4.28436).