Machine Learning and Time Series Models for Covid19 Predictions

Longpeng Xu

December 20, 2021

Abstract. Predicting the trend of the Covid19 pandemic accurately is an important measure to effectively allocate medical resources, save lives and guarantee public health, and promote social stability and post-pandemic developments. This project proves that machine learning models and ensemble learning (EL) based on the time series model (ARIMA) can closely fit the number of newly confirmed cases, and the idea of 'rolling prediction' applies to the latter. The implementation of the project includes data preprocessing, feature engineering, fitting and prediction, parameter tuning and other processes.

Key terms: machine learning, decision tree, random forest, ARIMA, ensemble learning

1 Introduction

At the end of 2019, a cluster of pneumonia cases of unknown origin appeared in Wuhan, Hubei Province. The causative agent was identified as a new coronavirus (2019-nCoV) on January 7 of the following year, and was later named Covid19 by the World Health Organization. A large amount of evidence shows that the virus is more contagious than SARS-CoV and MERS-CoV (China CDC, 2020). The epidemic was initially controlled in the province, marked by the complete shutdown including all means of transportation of the city, which effectively stopped the spread of the virus at huge economic and societal costs, however.

Hence, accurately predicting the pandemic trend will ensure the whole society within control effectively, safeguarding stability and development in the post-epidemic era. The value of machine learning shed light on data analysis methods for various industries outside computer science (Zhou, 2016), including epidemic prediction. The same is true for time series models, considering the sequential characteristics of epidemic datasets.

This project effectively explores prediction methods that are highly suitable for newly confirmed cases, where machine learning model are used. The best models are decision tree regression ($r^2 = 0.8876$) and random forest regression ($r^2 = 0.9210$); in ARIMA-based ensemble learning, the GBDT model performs best ($r^2 = 0.9246$). The remainder of this report consists of Concepts, Preprocessing and feature engineering, Analysis I: supervised model, Analysis II: ARIMA-based ensemble learning, Conclusions and suggestions.

2 Concepts

2.1 Machine learning

Machine learning is a domain that aids computers with human-like learning capabilities and then learn useful information from large amounts of data. It combines the advantages of human learning (accumulating experience $\rightarrow$ summarizing rules $\rightarrow$ flexible application) and computing power (Zhihu.com, 2016). Machine learning has a wide range of applications, including data mining, natural language processing, computer vision, biometric identification, medical diagnosis, fraud detection, etc. (”Machine Learning”, 2021). Machine learning algorithms include decision trees, random forests, Boosting, support vector machines, naive Bayes, expectation maximization, deep learning and other algorithms. Basically, machine learning divides a dataset into a training set and a test set, and make predictions on the test set by learning the training set.

2.2 Linear regression and regularization

Linear regression is about modeling by linear combination of first-order terms. The unknown parameter θ of the model is solved by data points and ordinary least squares (OLS). For 2D data points $(x_i, y_i),\ i = 1, ..., m$, set the target function as $h_\theta(x) = \theta_0 + \theta_1 x$, and the loss function is

$$J(\theta_0, \theta_1) = \sum_{i=1}^{m} \left(y^{(i)} - h_\theta\left(x^{(i)}\right)\right)^2 = \sum_{i=1}^{m} \left(y^{(i)} - \theta_0 - \theta_1 x^{(i)}\right)^2 \tag{1}$$

Find the partial derivatives to the multi-variable function $J(\theta_0, \theta_1)$, we have

$$\frac{\partial J(\theta_0, \theta_1)}{\partial \theta_1} = 2\left(\theta_1 \sum_{i=1}^{m} x_i^2 - \sum_{i=1}^{m} (y_i - \theta_0) x_i\right) \tag{2}$$

$$\frac{\partial J(\theta_0, \theta_1)}{\partial \theta_0} = 2\left(m\theta_0 - \sum_{i=1}^{m} (y_i - \theta_1 x_i)\right) \tag{3}$$

The minima when setting both Equation 2 and 3 to be zero are the parameter estimations under OLS:

$$\theta_1 = \frac{\sum_{i=1}^{m} y_i (x_i - \bar{x})}{\sum_{i=1}^{m} x_i^2 - \frac{1}{m}\left(\sum_{i=1}^{m} x_i\right)^2} \tag{4}$$

$$\theta_0 = \frac{1}{m} \sum_{i=1}^{m} (y_i - \theta_1 x_i) \tag{5}$$

Similarly, for multiple linear regression on samples $\left(x_1^{(i)}, ..., x_n^{(i)}, y^{(i)}\right),\ i = 1, ..., m$, set the target function as

$$h_\theta(x) = \sum_{i=0}^{n} \theta_i x_i = \theta_0 + \theta_1 x_1 + ... + \theta_n x_n \tag{6}$$

Accordingly, its lost function is defined as

$$J(\theta) = \sum_{j=1}^{m} \left(h_\theta\left(x^{(j)}\right) - y^{(j)}\right)^2 = \sum_{j=1}^{m} \left(\sum_{i=0}^{n} \theta_i x_i^{(j)} - y^{(j)}\right)^2 \tag{7}$$

Setting the partial derivative of Equation 7, $\partial J(\theta) / \partial(\theta_i)$, to be zero leads to a system of linear equations of full rank $n + 1$, which is solvable by linear algebra or matrix approaches (Zhou, 2021; Zhihu user, 2021).

In machine learning, regularization refers to introducing a penalty term to models which have been overfitting. To demonstrate, when more parameters are included in Equation 6, the model is more complex and biased towards its training set. Therefore, it will perform poorly on unseen test sets. L_0 regularization minimizes n, the number of parameters. L_1 regularization aka Lasso regularization minimizes $\sum |\theta_i|$. L_2 regularization aka Ringe regularization minimizes $\sum \theta_i^2$. If regularize a linear regression model, the loss functions includes but are not limited to Lasso and Ringe:

$$J(\theta) = MSE(\theta) + \frac{\alpha}{2}\sum_{i=1}^{m} |\theta_i| \tag{8}$$

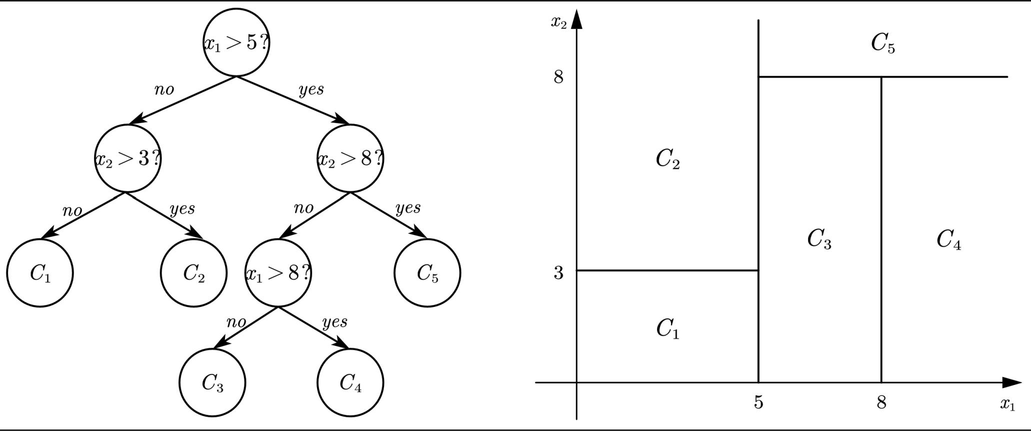
$$J(\theta) = MSE(\theta) + \frac{\alpha}{2}\sum_{i=1}^{m} \theta_i^2 \tag{9}$$

2.3 Decision tree regression and random forest regression

Decision tree regression mainly refers to CART (classification and regression tree) algorithm. The decision tree in Figure 1 partition its feature space (x_1, x_2) into classes $C_1, ..., C_5$, each of which has specific output $y_1, ..., y_5$. Hence, traversing all the features of each training data point allows their categorization into different classes and the corresponding outputs (Zhihu user, 2018). Similar to the solvable parameters in linear regression, the partitioning of the feature space also follows OLS, which is beyond the scope of the report.

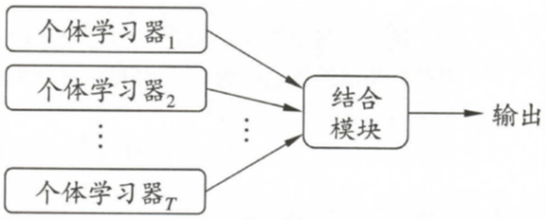
Random forest is a type of ensemble learning. As shown in Figure 2, ensemble learning refers to "first generating a group of individual learners and then combining them using a certain strategy". When the individual learners are decision trees aided with random feature selection, the ensemble model is a random forest. Such random selection allows each node of the base decision tree to extract only one attribute for inspection, which is computationally efficient[1] (Zhou, 2016). Therefore, random forest can be used as a regression model because of the division of feature space by decision trees.

Figure 1 Regression tree (left) and partition of its feature space(right)



[1]This holds when ceteris paribus. e.g., The number of variables pending parameter tuning equals the number of possible values for each variable.

Figure 2 Ensemble learning (Zhou, 2016)



2.4 ARIMA

ARIMA refers to autoregressive integrated moving average. Model $ARIMA(p, d, q)$ is the integration of autoregressive model $AR(p)$, moving average model $MA(q)$ and difference method with order d. The combination of the former two forms autoregressive moving average model $ARMA(p, q)$:

$$X_t = \alpha_1 X_{t-1} + \alpha_2 X_{t-2} + ... + \alpha_p X_{t-p} + \varepsilon_t + \beta_1 \varepsilon_{t-1} + ... + \beta_q \varepsilon_{t-q}$$
$$\xrightarrow{q=0} AR\,(p) \quad \text{or} \quad \xrightarrow{p=0} MA\,(q) \tag{10}$$

where $\{\varepsilon_t\}$ is an i.i.d. white-noise sequence with zero mean, independent of $\{X_t\}$ (Li, 2021). These models can describe time series. For stationary time series, $ARMA(p, q)$ can predict future values based on past values. While for non-stationary time series, $ARIMA(p, d, q)$ is required.

3 Preprocessing and feature engineering

The original dataset consists of 68 rows and 10 columns. The rows range from 31 December 2019 to 7 March 2020, and the columns include the numbers of

- incremental confirmed cases
- incremental recovered cases
- incremental deaths
- corrections (adjusted down, also for the followings)
- corrections to recovered cases
- corrections to death cases
- cumulative confirmed cases given corrections
- cumulative recovered patients
- cumulative death cases
- cumulative confirmed cases

which are integers. The small amount of data and collinearity among features are key concerns for subsequent data preprocessing and model selection. Firstly, exploratory data analysis was performed, where a line chart of each feature was plotted (Figure 3). It was initially found that there is an outlier in the newly confirmed cases, using the 68-95-99.7 rule. The outlier was replace by the mean of the values of its last and its next period. The 10 features to be preprocessed $\mathbb{X}$ and newly confirmed cases Y are the original data required by the ARIMA model.

For machine learning models, $\mathbb{X}$ is identically defined, while Y is defined as the newly confirmed cases of the next period. $\mathbb{X}$ is standardized, and Y are substituted by its logarithm. For feature engineering, I constructed new features based on existing ones: the number of existing cases, the number of net discharged cases, the number of net culumative recovered cases and the number of days since 31 December 2019. Hence, there are 14 features. After splitting the dataset into train and test sets, principle component analysis (PCA) was used for eliminating the collinearity, where dimensionality reduction can be done optionally. As in Figure 4, the cumulative explained variance ratio (CEV) of the top three PCs reaches 0.999.

Figure 3 Line chart of features of the original dataset (excerpt)

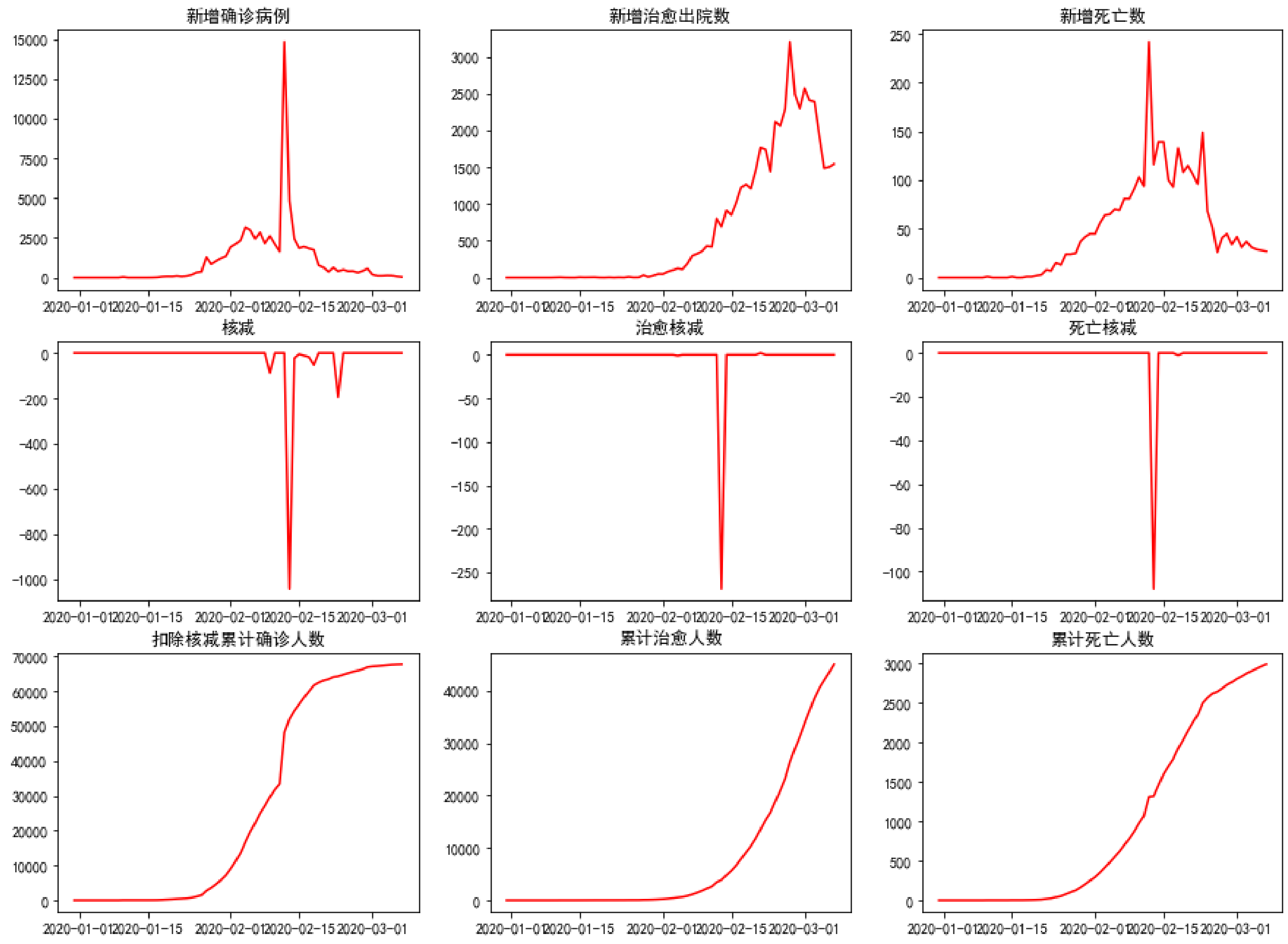
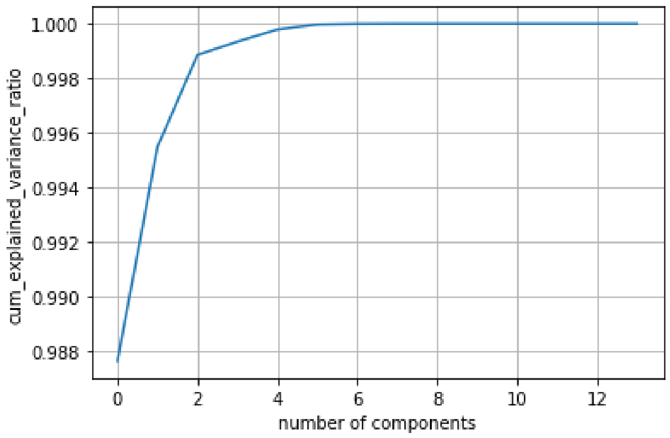


Figure 4 Cumulative explained variance ratio



4 Analysis I: supervised models

The experiments compared three approaches: (1) Data without PCA processing, (2) Data with eliminated collinearity and (3) Data with eliminated collinearity and reduced to 3 dimensions, for the 0.999 CEV ratio. The following models are invovled: linear regression, Lasso regression, decision tree regression and random forest regression. It can be concluded that

- When PCA is not used, decision tree regression is the best model because its MSE is the smallest and R-squared is the largest.
- When PCA is used to remove collinearity, random forest regression is the best model (same reason as before).
- When using PCA to reduce to 3 dimensions, it is difficult to judge the best model.
- It is difficult to compare whether using PCA or using PCA to remove collinearity is better.

See Table 1 for the results. The following discussion is based on results without using PCA processing.

4.1 Linear regression and lasso regression

The models are relatively simple. The respective code implements the following: (1) Instantiate a model; (2) Train the model on the train set (both $\mathbb{X}$ and Y for supervised learning); (3) Obtain the prediction of Y for both train and test set; (4) Obtain MSE and R-squared for the train and the test sets, respectively. Linear regression balance its performances on the train and the test sets. In contrast, Lasso regression ($\alpha = 0.001$) performs poorer than linear regression, especially in the test set. The upper subplots of Figure 5 illustrates the fitted versus true values of linear regression and Lasso regression, respectively.

Table 1 The results of the three data approaches (columns) and the four supervised models (rows)

	Without PCA processing				**With eliminated collinearity**			
	MSE_{train}	MSE_{test}	R^2_{train}	R^2_{test}	MSE_{train}	MSE_{test}	R^2_{train}	R^2_{test}
Linear	146043	276746	0.878685	0.737365	170623.0	744323.0	0.857063	0.329477
Lasso	148853	687809	0.876350	0.347262	254754.0	372709.0	0.786583	0.664245
Decision Tree	54348.2	118482	0.954854	0.887559	174575.0	321313.0	0.853753	0.710545
Random Forest	212220	176534	0.823712	0.832467	85292.8	87643.5	0.928547	0.921046

	With eliminated collinearity & reduced to 3D			
	MSE_{train}	MSE_{test}	R^2_{train}	R^2_{test}
Linear	12544500	267976.0	-8.43467	-0.302207
Lasso	12491100	267378.0	-8.39453	-0.299300
Decision Tree	71564.2	125942.0	0.946177	0.387993
Random Forest	163000	56804.5	0.877409	0.723963

4.2 Decision tree regression and random forest regression

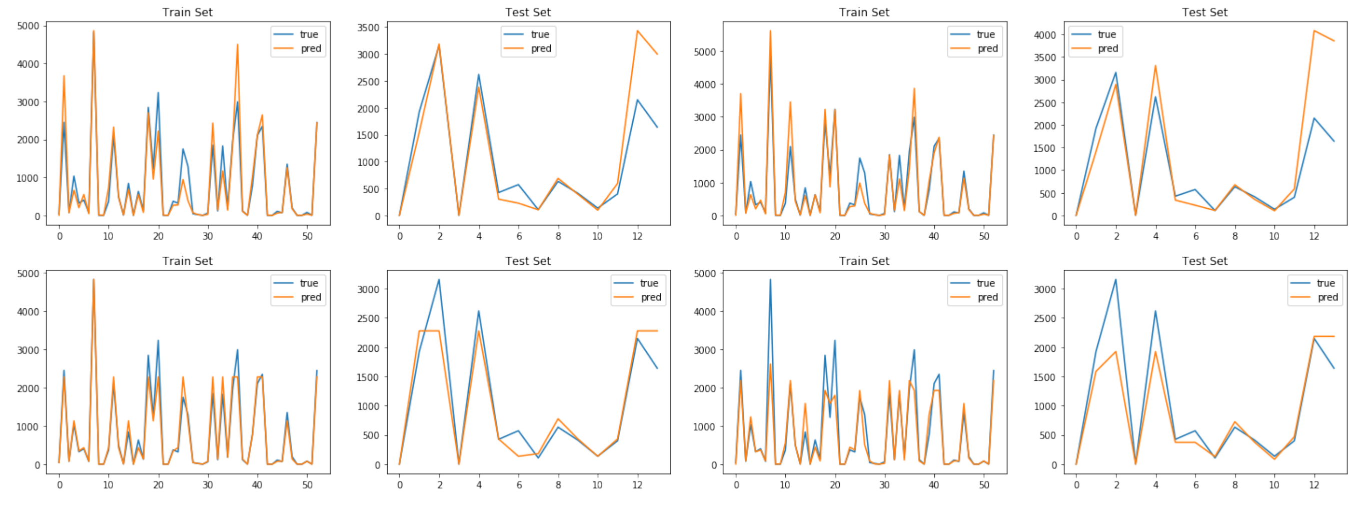
Different from the above two models where the parameters are mostly boolean and do not require tuning, decision tree regression and random forest regression require parameter tuning before being instantiated. In addition, the ranges of parameters are collated in a Python dictionary. See Listing 1.[2]

From Table 1 and the lower subplots of Figure 5, we know that compared to simpler models, decision tree regression and random forest regression have robust performance, especially the remarkable improvement in R-squared.

Listing 1 Train a decision tree regressor

```
# set up an instantiated model
dtr = DecisionTreeRegressor(random_state = 0)

# ranges of parameters
params = {
    'max_depth':  list( range(3,10,1)),
    'max_features':  list( range(5,16,2)),
    'max_leaf_nodes':  list( range(5,20,2))}

# instantiate an Gridsearch
gs_dtr = GridSearchCV(estimator = dtr, param_grid = params, cv = 5, scoring = make_scorer(r2_score))

# train the model
gs_dtr.fit(x_train,y_train)

# check the optimized parameters
print(gs_dtr.best_params_)

# check the metrics under the parameters
print(gs_dtr.best_score_)
```

Figure 5 Fitted versus true values of linear regression (top left), Lasso regression (top right), decision tree regression (bottom left) and random forest regression (bottom right)



[2]A model is not de facto 'instantiated' until its parameters are specified by the optimized ones given by Gridsearch.

5 Analysis II: ARIMA-based ensemble learning

5.1 ARIMA-based rolling predictions

The idea of rolling prediction is, each time slice a part of Y which increments in length until the whole Y is retrieved. Next, for the stationarity of the time series, operate logarithm, moving average, first-order difference, and first-order difference again[3] on the slice. Then, obtain the initial prediction from ARIMA, and undo the operations on the prediction to retrieve real predictions, one real prediction each time. Therefore, rollingly slice Y and rollingly make predictions is why the function in Listing 2 returns a list of predictions (of length 61). Nevertheless, from Figure 6 we know that the ARIMA does not lead to a highly fitted model, which requires further feature extraction.

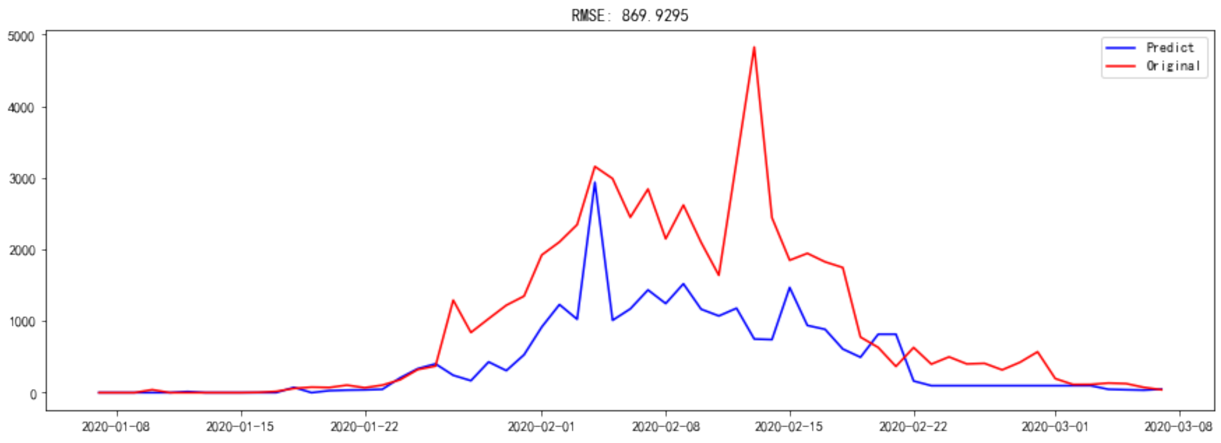
Listing 2 Get predictions from ARIMA

```python
def arima_model_pred(Series_1, step=12, p_max=4, q_max=4):
        # empty list for storing results
        arima_pred = []

        for ij in  range( len(Series_1)-step+1):

                # get data rollingly and their logarithm
                if ij+step <  len(Series_1):
                        dtf = np.log(Series_1[:ij+step-1]+1)
                else:
                        dtf = np.log(Series_1[:]+1)

                # get results of moving average and first-order difference from function 1 (previously defined)
                rol_mean, diff_1 = plot_data(dtf)

                # get parameters from function 2 (previously defined)
                p = arima_order_select_both(diff_1, p_max, q_max)[0]
                q = arima_order_select_both(diff_1, p_max, q_max)[1]

                # get predictions from function 3 (previously defined). notice that modelling can be failed since
                     automatic order setting can be unstable
                # if this is the case, use the predictions of last period again
                try:
                        a_pred = arima_model(diff_1, rol_mean, dtf, ij, order=(p,1,q))
                except:
                        a_pred = a_pred

                # get the initial predictions of the model
                if ij==0:
                        arima_pred.extend(a_pred)
                        print('Initally processing data of 12 days gets predictions of',  len(a_pred), 'days')

                # get the rolling predictions
                else:
                        arima_pred.append(a_pred)
```

[3]This is conducted in $ARIMA(p, 1, q)$.

```
                print('The', ij, '-th processing of data is streaming')
    return arima_pred
```

Figure 6 The predictions from ARIMA versus true values



5.2 Data preparation for ensemble models

- Feature extraction: Generate some new features statistically based on the existing ones, and generate features which are one and two periods ahead of the new features. After this step, there are 18 new features in the dataset, including the list of predictions.
- Rollingly retrieve the finalized data: Concatenate the 18 features with $\mathbb{X}$. From the 21st day to last, rollingly slice the 28 features and save the slice to the list `features`, until the second last day is included. Therefore, there are 20 dataframes in `features`. Similarly, do the same to the two-period-lagged Y and save it to `labels`. Hence, there are 20 series of sliced Y in `labels`.
- Data normalization: For each element in `features` and `labels`, conduct train-test split and data normalization repsectively.
- Feature filtering: Use PCA to reduce the dimensions of the train/test sets from `features`. As a result, there are four lists, which are the arguments in the bottom line in Listing 3.

5.3 Parameter tuning and EL predictions

I developed a rolling version of Gridsearch parameter tuning, which has an additional step of saving the optimized set of parameters iteratively for each of the 20 dataframes and series. The first five sets of parameters for GBDT are shown in Table 2.

When the four lists and the parameters are prepared, model training, predicting and comparing are left on the to-do list. Here only the predictions on each test set is required. And the last value of each set of predictions is compared against the the true value. We selected GBDT (gradient boosting decision tree), random forest, and bagging for comparison. The performance of GBDT is the best, with R-squared 0.9246 (see Figure 7). In contrast, random forest and bagging have unsatisfactory performances, with R-squared 0.6573 and -0.2309, respectively. In addition, rolling Gridsearch parameter tuning slightly improves the performance of GBDT, by 1.26%.

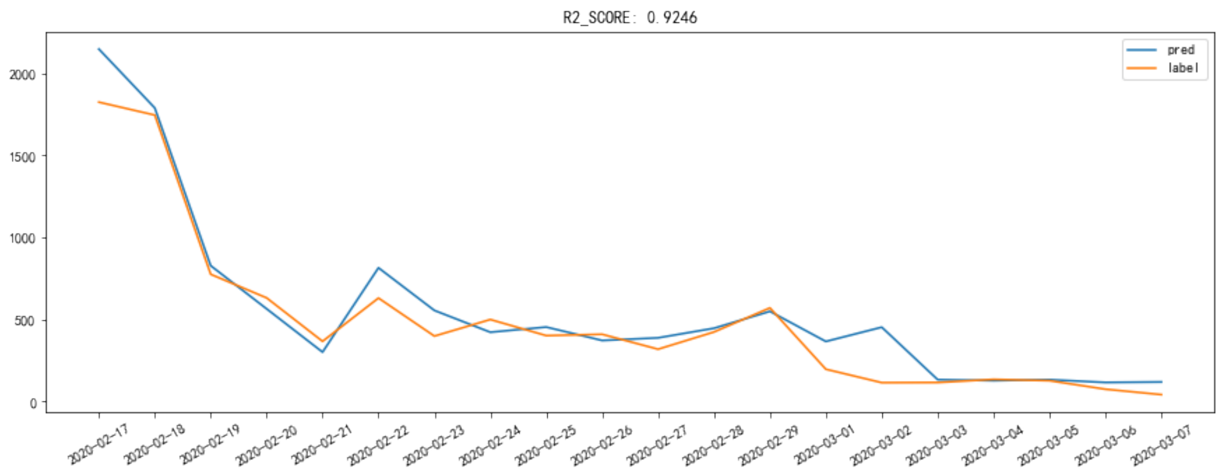
Table 2 Optimized sets of parameters for GBDT (excerpt)

#	learning rate	max depth	n_estimators	score	elasped time
0	0.15	9	60	-0.243688	39.440 s
1	0.05	5	120	-1.22912	39.899 s
2	0.1	3	150	-1.03275	40.970 s
3	0.1	3	150	-0.884883	40.135 s
4	0.15	5	150	-1.14276	40.950 s

Listing 3 The predictions from GBDT versus true values

```
def gbdt_loop(xtr,xte,ytr,yte):
        # empty list for storing predictions
        pred = []

        # iteratively operate on the dataset of datasets (with progress bar by tqdm)
        for i in tqdm( range( len(features))):
                # get relevant data
                X_tr = xtr[i]
                X_te = xte[i]
                y_tr = ytr[i]
                y_te = yte[i]

                # instantiate GBDT with least absolute deviation (LAD) for better robustness
                gbdt = GradientBoostingRegressor(learning_rate = best_params.iloc[i,0],
                        max_depth =  int(best_params.iloc[i,1]), loss='lad',
                        n_estimators =  int(best_params.iloc[i,2]), random_state=0)

                # fit the train set
                gbdt.fit(X_tr,y_tr)

                # predict on the test set
                c = gbdt.predict(X_te)

                # save the predictions of the test set
                pred.append(c[-1])

        return pred

pred = rfr_loop(features_f_tr, features_f_te, labels_tr, labels_te)
```

Figure 7 Predictions from GBDT versus true values



6 Conclusions and suggestions

The impressive fittedness of ARIMA and ensemble learning models derives from their appropriate complexity. Simple models like linear regression in Analysis I can fit the dataset as well, as in an R-squared of 0.7373 in Table 1, but the former ones introduce the extra benefit of cross-validation and more likely to fit a wider range of datasets. Nevertheless, the significance of Analysis I consisting of three data approaches and four models cannot be ignored on the road of exploring.

In addition, the main takeaway from Analysis II: (1) The idea of 'rolling prediction' makes full use of a small-size timer-series dataset at the greatest extent, since it emphasizes the influence by the latest term of data on past data and predictions. (2) Based on the highly predictive results, it can be argued that a large sum of feature engineering does not necessarily mean overfitting in case of time-series data, but trying to extract all hidden patterns of the datasets, which again maximizes the utilization of a small dataset.

Finally, further improvements to this project can be: (1) The comparison in Analysis I is not sufficient enough. A scenario of collinearity elimination and dimensionality reduction with $d > 3$ can be included. (2) Try more ensemble learning models like XGBoost (eXtreme Gradient Boosting), which can lead to more optimized performance than that of GBDT.

7 Bibliography

China CDC. (2020). The China CDC releases an analysis report on the epidemiological characteristics of COVID-19. Chinese government. Retrieved December 17, 2021, from http://kpzg.people.com.cn/n1/2020/0219/c404214-31594469.html

Li, D. (2021, n.d.). Lecture notes on financial time series analysis. School of Mathematical Sciences, Peking University. Retrieved November 18, 2021, from https://www.math.pku.edu.cn/teachers/lidf/course/fts/ftsnotes/html/_ftsnotes/index.html

Machine Learning. (2021, January 14). In Wikipedia. https://en.wikipedia.org/wiki/Machine_learning

Zhou, Z. (2016). Machine Learning. Beijing: Tsinghua University Press.

Zhihu user. (2016, April 25). What is machine learning? Zhihu.com. https://www.zhihu.com/people/Princetechs-Wangchu/answers

Zhihu user. (2018, August 20). Decision tree — regression. Zhihu.com. https://zhuanlan.zhihu.com/p/42505644

Zhihu user. (2021, April 16). Ordinary least squares for linear regression. Zhihu.com. https://zhuanlan.zhihu.com/p/90073632