Final Project Part-II

alpha

11/15/2020

1 Introduction

The main topic of the final project part II is to formulate reasonable models and algorithms based on the structure of our data and perform cross validation to assess their prediction accuracy. Instead of directly throwing predictors into different models, we found a main challenge in the data structure: Since we have observations from year 2011 to 2018, it's possible that the observation from the same country in different year could be correlated. Thus, we should also consider a data transformation that can eventually help our models to yield better results. After constructing models, we will discuss a model comparison and decide which model has the best performance.

2 Exploratory Data Analysis and Data Processing

2.1 Data Processing

2.1.1 Following the instruction

As the instruction said, the data contains some countries without all information for 8 years, therefore our first step of the data processing is to filter them out and construct a "clean" dataset such that each country has the data for all 8 years. Then, we can split the full data into training data and testing data. We will use the data from 2011 to 2017 to train our models, and the data for 2018 as the testing data.

Also, we create a new predictor GDP per capita by dividing `GDP` by `population`, which is a reasonable way to get rid of the effect of different base of GDP and population.

2.1.2 Taking subtraction to de-correlate the data

We want to point out that, this dataset is a typical **panel data**, which includes both entities (Countries) and time information (Year). For most of the dataset we've seen in class, values of each observation can be seen as independent. However, for this Happiness data, values in one country across years will be kind of correlated, which is so called autocorrelation.

We will talk more about this problem later in **Part 3**. Here what we want to make home is that, due to this problem, we decide to de-correlate the data by performing subtraction between observations of adjacent years in each country, resulting a new dataset containing the increments of each variable in each country. Therefore, when we do prediction, the input predictors are not values of 2019 data but the subtraction value between 2019 and 2018; also, the output response is not the predicted value of happiness in 2019 but the predicted subtraction value between 2019 and 2018.

We will show the advantages of doing this in the model developing part of BMA and Tree models.

2.1.3 Finding possible interactions

At this point, we no longer need to worry about the interpretability of our model, so we can include more predictors which may help improve our prediction without consider too much about their realistic meaning.

To find possible interactions quickly, we decide to first fit a linear model with Country as dummy variable and including all other predictors and their interactions. In code it will be: `lm(Happiness ~ Country.Territory + (GpC + Support + Health + Freedom + Generosity + Corruption + Positive + Negative + Government + Gini.Index)^2, data = data2_diff)`.

And then, we use `step()` function to do stepwise algorithm with AIC and BIC. The remaining predictors are shown in formulas:

AIC:

```
## Happiness ~ GpC + Support + Health + Freedom + Generosity + Corruption +
##     Positive + Negative + Government + Gini.Index + GpC:Freedom +
##     Support:Negative + Support:Government + Support:Gini.Index +
##     Health:Freedom + Health:Generosity + Health:Corruption +
##     Health:Negative + Health:Government + Freedom:Positive +
##     Freedom:Gini.Index + Generosity:Negative + Generosity:Government +
##     Corruption:Positive + Corruption:Government + Positive:Negative +
##     Positive:Gini.Index + Government:Gini.Index
```

BIC:

```
## Happiness ~ Support + Health + Freedom + Corruption + Positive +
##     Negative + Government + Health:Freedom + Health:Corruption +
##     Health:Negative + Corruption:Positive
```

It is obvious that the model selected by AIC contains more predictors, which not only keeps all the original predictors, but also give us many possible interactions. Due to the fact that we want to have wider variable "space" to explore, we choose all the predictors in the model selected by AIC.

We will use them and show the advantages of doing this later in the model developing part of ensemble models.

2.2 Exploratory Data Analysis

In this section, we will focus on the EDA of difference data.

First we can have a look at the overall distribution of the response variable "Happiness".

We can see that the response variable "Happiness" has approximately normally distribution with mean 0.

We can continue to explore the distribution of predictors.

From Figure 2 , we can see that most covariates has distribution near symmetric. One noticeable problem is that the predictor "Health" seems to be distributed in a right-skewed shape. Moreover, from the boxplot, we can see that "Health" has generally a extreme outlier. We can estimate its density to confirm these.

Indeed, from Figure 3 we can observe a slight right-skewed pattern in the density plot above. We can find out the outlier (Table 1).

Since we will be using ensemble methods, which are not going to be significantly affected by the outliers, we can keep the data unchanged. The discovery of possible outliers means that we can expect ensemble method to have better performance than single model.

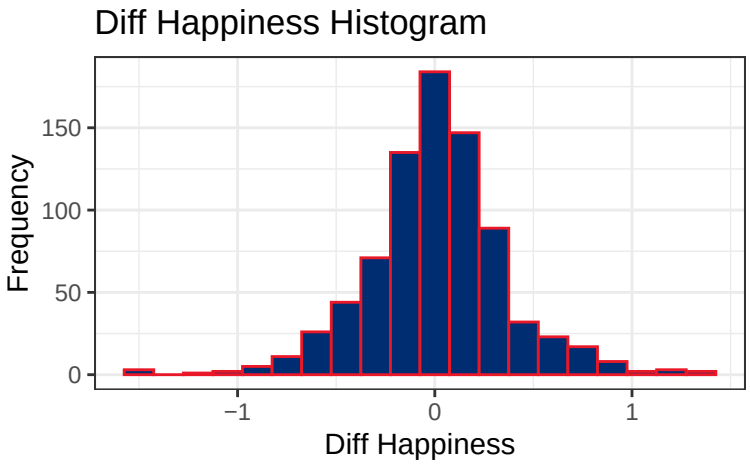


Figure 1: Diff Happiness Histogram

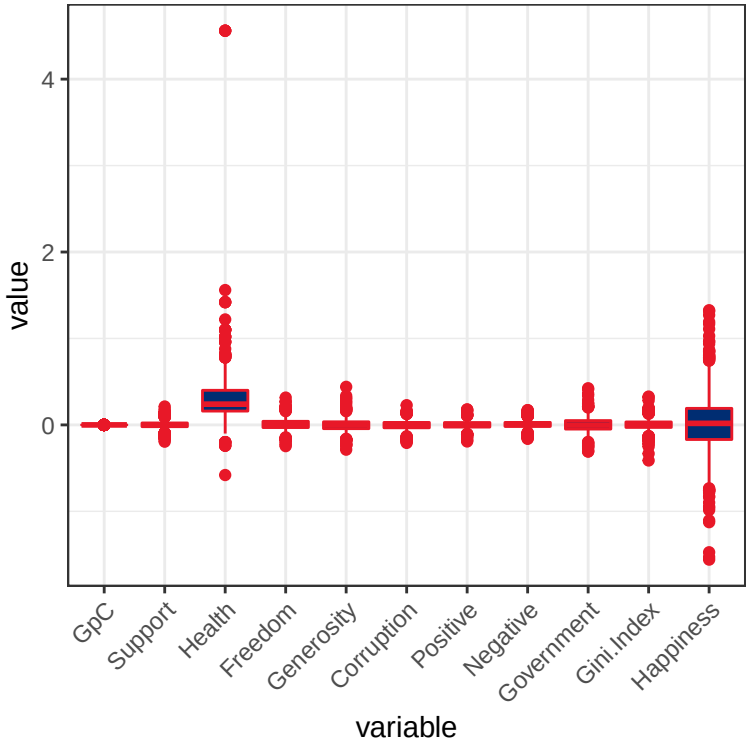


Figure 2: Predictors boxplot

Table 1: Outlier

	Country.Territory	Health	Happiness
309	Haiti	4.56	-0.431

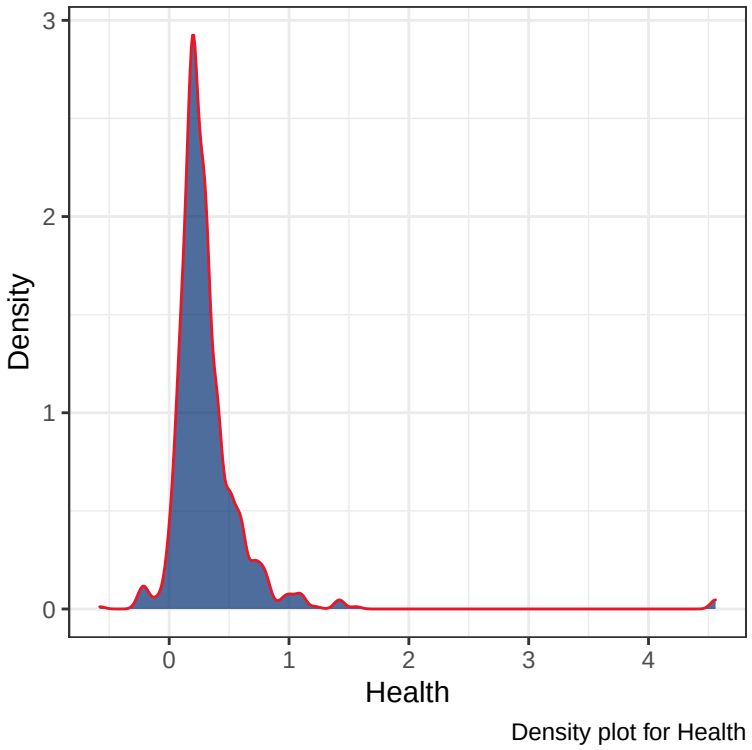


Figure 3: Density plot for Health

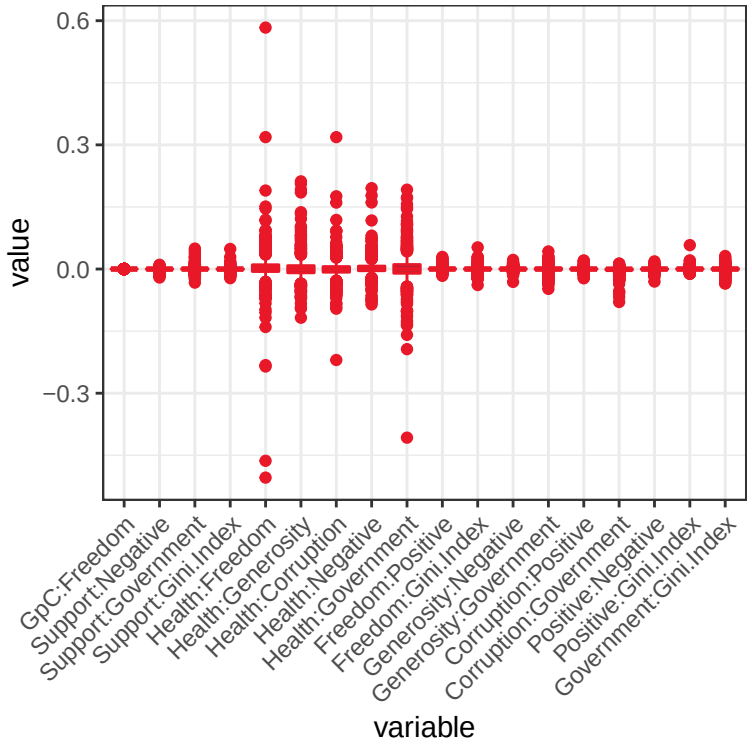


Figure 4: Interaction Boxplot

Besides the original data, we can also have a look at the interactions.

From Figure 4, we can conclude that interaction terms tend to have smaller magnitude than original data. However, a interesting pattern to notice is that the interaction term containing "Health" have exceptionally large variances and extreme outliers both in positive and negative direction. This could be the result of the distribution of "Health" and thus affected the distribution of interaction terms. Since we will use interaction terms in ensemble models, the outliers won't have noticeable effect on the result.

3 Preliminary Model in Part I

As we have said in Part 2, considering the fact that the data might cause autocorrelation issues, we decided to look at the linear model in part 1 and plot its residual to see if we can detect any "sticky" pattern to see if further processing is needed.

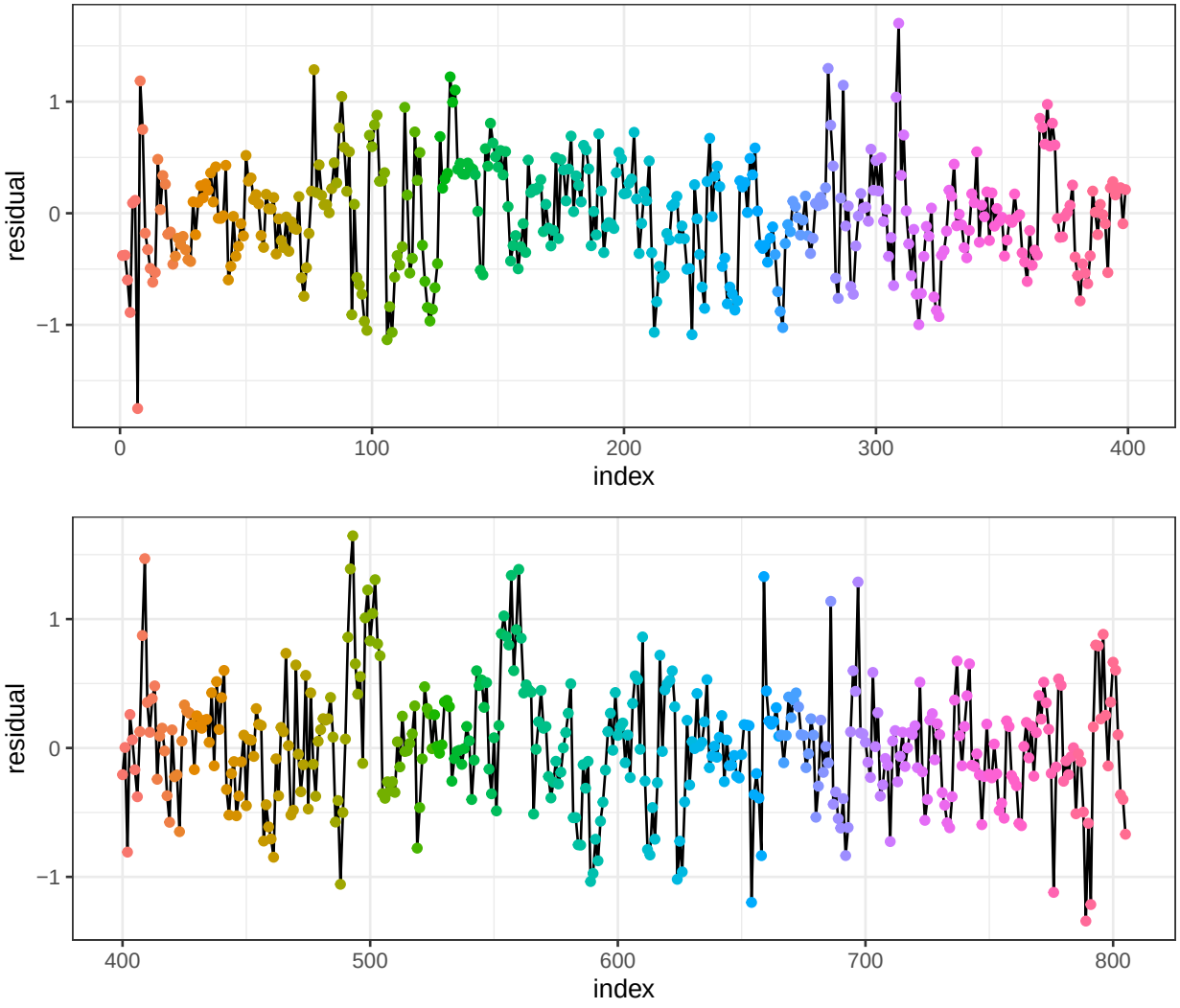


Figure 5: Residual plot

Figure 5 is the residual plot for the linear model we fitted in Part 1. We plotted it in 2 separate graph for better clarity. In Figure 5, each color represents the country of that data point. The first graph is the

residual plot for data points of first 57 countries and the second graph is the residual plot for data points of the remaining 58 countries.

In general, a linear model should have residuals that are normally, independently and identically distributed. However, in Figure 5, we can observe obvious "sticky" patterns. Moreover, the data points with the same color tend to have similar trend and when color changes, the trend also changes. Thus, we can conclude that the residuals are correlated and assumptions for linear model is violated.

Also we can testify this by performing the Durbin-Watson test for autocorrelation of disturbances.

```
##  lag Autocorrelation D-W Statistic p-value
##    1          0.5217        0.9534       0
##  Alternative hypothesis: rho != 0
```

In the Durbin Watson Test, the null hypothesis is autocorrelation equals to 0. In the testing result, we can see that the p-value is close to 0, which means that the null hypothesis is rejected, so we can conclude that the autocorrelation is not 0.

Thus, for further model training and predictions, we decided to construct the difference/increment data and use non-linear models for our prediction, especially tree models and ensemble tree models.

4 Development of The Final Model

Due to the fact that we detect the potential autocorrelation problems, we decide to focus more on tree models and ensemble tree models. Below we are going to use four models as our options: BMA, single tree model, also ensemble models of XGboost and Random Forest.

For BMA and single tree model, our focus is on comparing the difference between using original data and our "difference data", and also the reasoning or meaning behind the result.

For XGboost and Random Forest, our focus is on finding whether we should include all these interaction terms or not, and also how precise our prediction can be.

4.1 Baseline RMSE for final models

Although it's not required to do this, we also evaluate our models according to the test score from Kaggle leaderboard. Due to the fact that "the leaderboard is calculated with approximately 25% of the test data". So we kind of enlarge our "test data" with this move, and thus made lots of submissions.

The point here is, we did some simple or naive predictions and submit them to Kaggle to get a baseline test RMSE on Kaggle leaderboard for our final models. The following is what we've done:

1. 2018 Happiness data

If someone without any statistical knowledge tries to predict 2019 Happiness, the easiest thing to do is just guessing with the value of 2018 Happiness. This "philosophy" may have something to do with the martingale. No matter what, the point is, prediction models shouldn't do worse than "doing nothing". The Kaggle score of 2018 Happiness data is 0.39415.

2. ARIMA models

If we don't have other predictors information, then this problem will become a total time series problem. That's why we come up with using ARIMA as an origin of our baseline. We automatically build ARIMA models for each country with the function `auto.arima` in the {forecast} package. The Kaggle score of ARIMA models is 0.52110, worse than 2018 Happiness data. This tells us that predicting without other predictors is not reasonable.

The following are all models that we have attempted, including BMA, Tree models, also ensemble models of XGboost and Random Forest.

4.2 BMA

The first model we constructed is the Bayesian Model Averaging with unit information prior, which means set g to be n. We will train two models, one with the original data and another with the difference data for the comparison. For the model using original data, we added a interaction term between "Country" and "Year" to increase accuracy. The training for both model will try to enumerate over possible models with MCMC method. The model prior is set to be uniform.

Table 2: BMA RMSE

RMSE for original data	RMSE for difference data
0.417	0.277

We can see that the RMSE for the model using difference data is much lower than that of the model using original data. The result here further confirms that our initial consideration about the dependent observations is correct, and our preprocessing significantly improved prediction result. Thus, we selected the BMA with difference data as our BMA model.

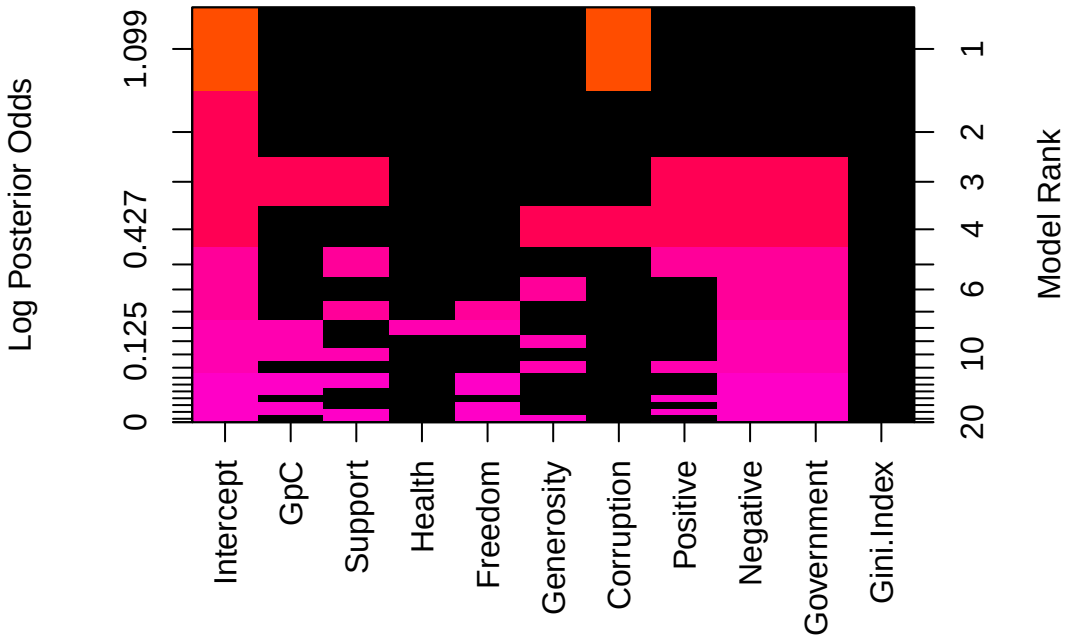


Figure 6: BMA Model Rank

In the Figure 6, we excluded the predictor "Country.Territory" since there are too many levels. Showing all of them will significantly affect the clarity of the graph.

Figure 6 shows that "Gini Index" is not included in top rank models. Based on common knowledge, Gini Index should be a important measurement of the life quality of population, so here we can conclude that its effect is well explained by other variables. On the other hand, we can see that "Government" and "Negative" are included in rank 3 to 20 models. The top rank model has only the intercept and "Corruption", but lower rank models rarely contain variable "Corruption" together with other variables. This means that the effect of variable "Corruption" could be explained by other variables. Overall, we can see that models can vary quite significantly in BMA, and simple models tend to have higher posterior probability. The model averaging provided a relatively small RMSE, but the true model may not be in our model space above.

4.3 Single Tree Model

4.3.1 Comparison of Original Data and Difference Data

After BMA, we decided to continue with tree model. Our first step is to fit a single tree and compare the performance of original data and difference data. We will use function `tree()` from package `tree` and compute the RMSE.

Table 3: Tree RMSE

RMSE for original data	RMSE for difference data
0.6833	0.312

After fitting the models, we can visualize the trees.

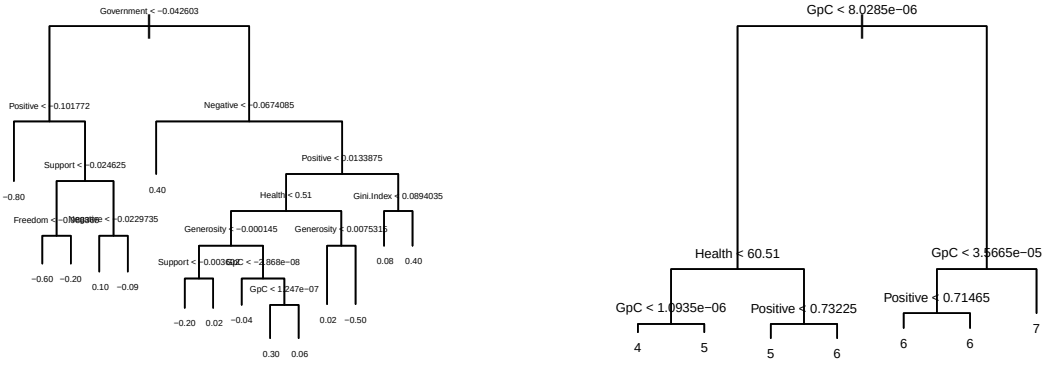


Figure 7: Regression Tree for Difference Data

From Figure 7, we can see that the RMSE and structure of trees constructed using original data and difference data are completely different. Overall, the tree model using difference data has significantly smaller RMSE and larger tree depth. The obvious disparity between the trees could be the result of the instability of tree

models. Overall, the RMSE is not ideal, and the BMA model resulted a smaller RMSE than the single tree model. Thus, we would like to proceed to perform cost-complexty pruning to improve model performance.

4.3.2 Cost-Complexity Pruning

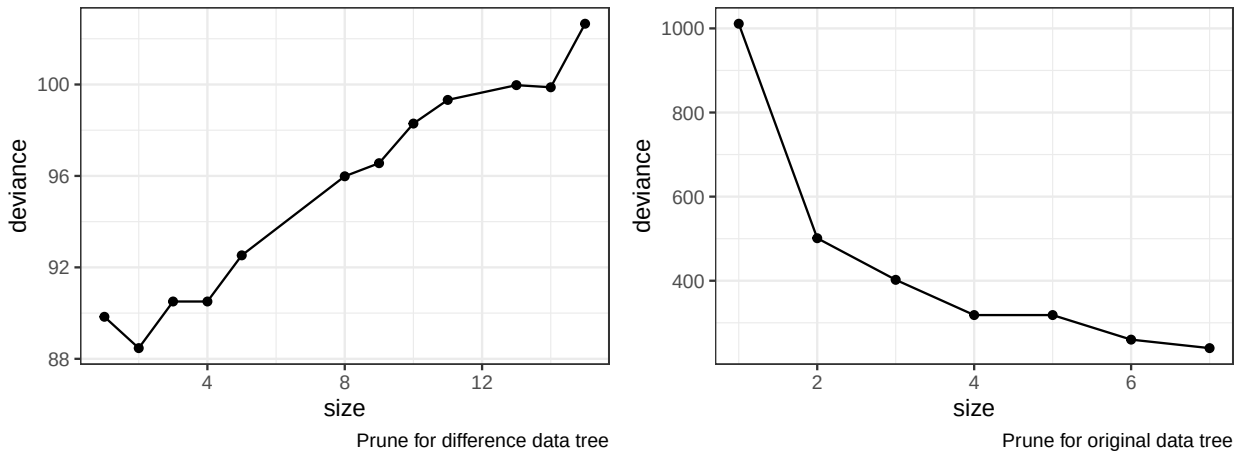


Figure 8: Pruning Cross-Validation

The cross validation (Figure 8) shows that the deviance is minimized when $Size = 2$ for the tree model using difference data, while $Size = 7$ gives the best result for tree model using original data. We can continue to refit the tree models and compute their out of sample RMSE.

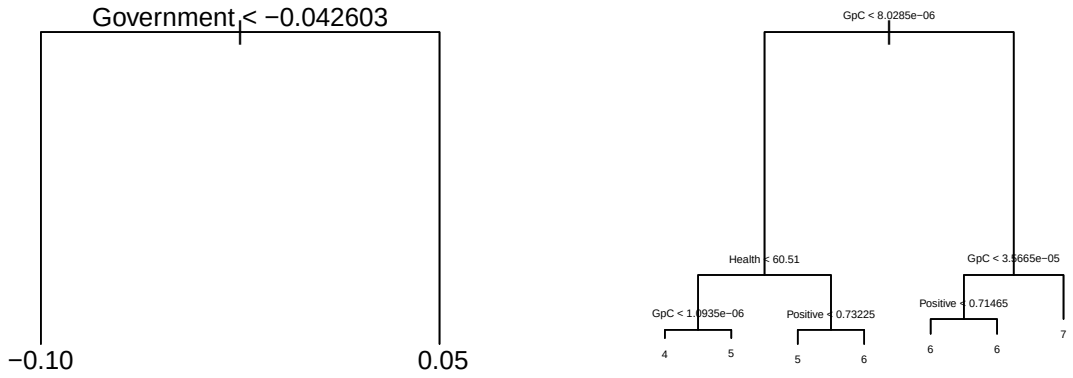


Figure 9: Pruned Trees

After pruning, we can see that the tree model using original data stay unchanged, thus there is no improvement for the RMSE. On the other hand, the tree model using difference data now has significantly lower tree depth and its RMSE decreased by a considerably amount. Overall, the out of sample RMSE is still slightly

Table 4: RMSE for Pruned Tree

RMSE for original data	RMSE for difference data
0.6833	0.285

larger than the BMA model with unit information prior, but we expect it to continue to improve after we apply aggregated models.

4.4 XGBoost

For the XGboost model, we use `xgboost()` function in package {xgboost} to do the training. Because of the rule, we set `eval_metric`, evaluation metrics for validation data, to be "rmse". We have three hyperparameters to tune, names and their meanings in the help document are below:

- `eta`: "control the learning rate: scale the contribution of each tree by a factor of $0 < \text{eta} < 1$ when it is added to the current approximation. Used to prevent overfitting by making the boosting process more conservative. Lower value for eta implies larger value for nrounds: low eta value means model more robust to overfitting but slower to compute. Default: 0.3".
- `max.depth`: "maximum depth of a tree. Default: 6".
- `nrounds`: "max number of boosting iterations".

4.4.1 Training process

Here we first build XGboost without possible interactions we've chosen before, and see what the result will be. Then we include those interactions later and see whether there is a difference.

Due to the fact that, smaller `eta` will take us much longer time to train a model, therefore we first set `eta` to be 0.01, a "decent" value as a starting point of the learning rate, and first tune on `max.depth` and `nrounds`. A plot of how test RMSE is changed with `max.depth` from 1 to 8 and `nrounds` from 150 to 300 is shown below:

From this plot, we can find some clear patterns. First, it's obvious that `max.depth` = 1 out-performs other options of `max.depth`. This result is reasonable because we don't have many predictors now, and boosting algorithm usually can perform well with stumps. Second, `nrounds` = 250 should be a good point of stopping the training. We will be in danger of over-fitting if we go above this point too much.

Setting `max.depth` = 1, then we set `eta` to be 0.001 and try to find optimal `nrounds` again. A plot of how test RMSE is changed with `nrounds` from 1000 to 3000 is shown below:

Here we can see that, we truly need much more rounds to get the optimal test RMSE point. The curve of test RMSE gets flat after nrounds of 2500. However, this slow learning rate model doesn't give us much better test RMSE. It seems that `eta` = 0.01 is enough here.

To sum up, for XGboost without interactions, we choose `eta` to be 0.01, `max.depth` to be 1 and `nrounds` to be 250. Here we can see the first 7 most important features in the model after tuning.

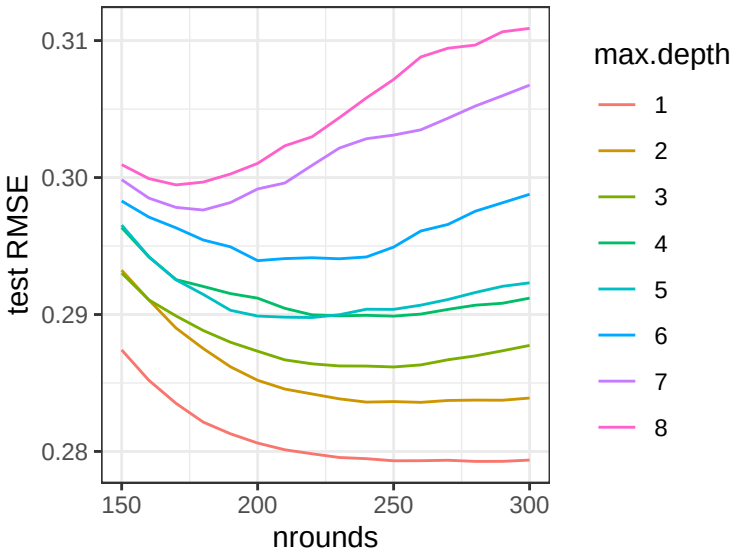


Figure 10: test RMSE vs max.depth and nrounds (XGB without Interactions)

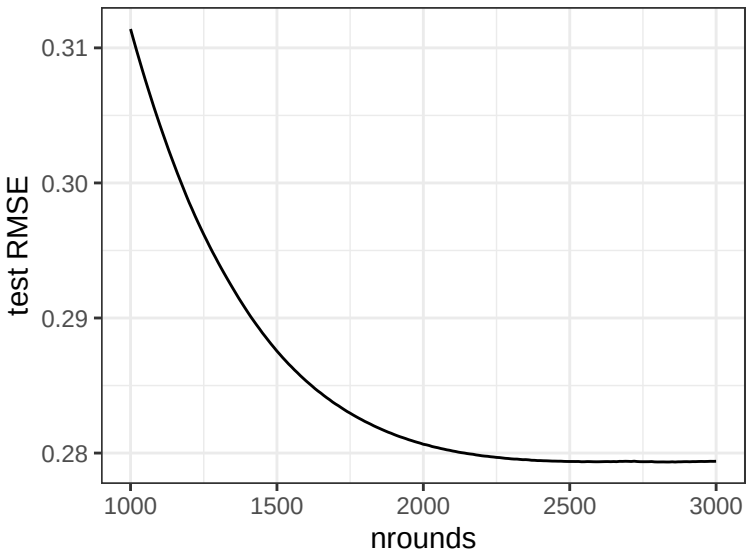
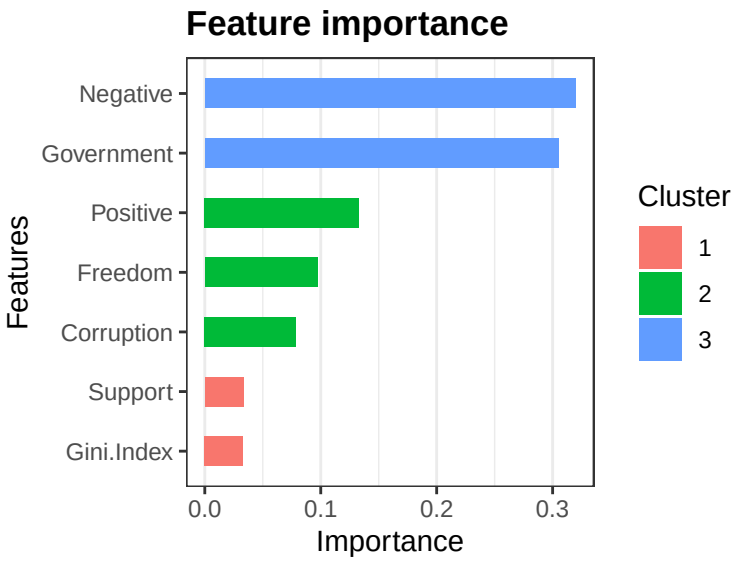


Figure 11: test RMSE vs nrounds (XGB without Interactions)



It seems that Negative and Government are the most important order-one predictors, which is consistent with the single tree model which tells us the importance of Government feature.

4.4.2 Including Interactions

Now we include interactions and go through the procedures above again.

We also first set `eta` to be 0.01, a plot of how test RMSE is changed with `max.depth` from 1 to 8 and `nrounds` from 150 to 300 is in Figure 12.

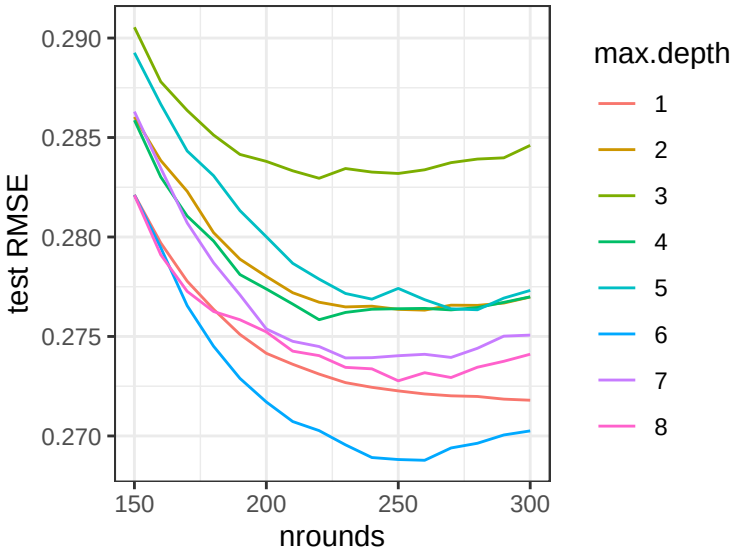
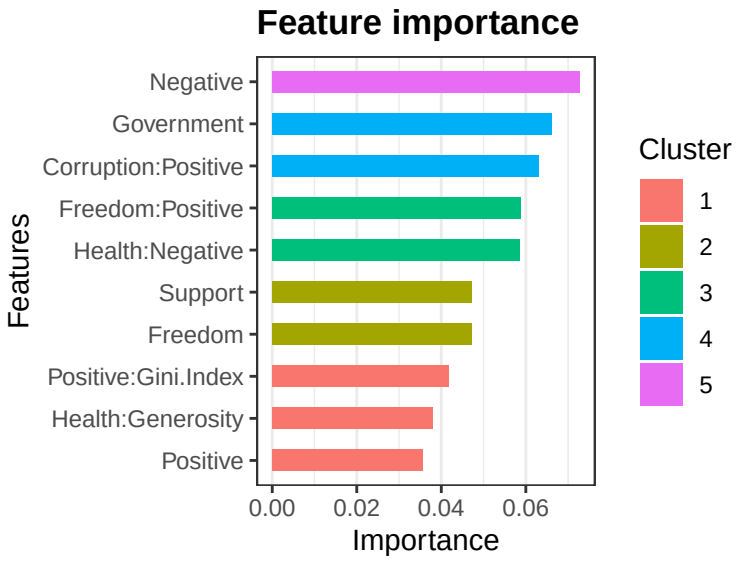


Figure 12: test RMSE vs nrounds (XGB with Interactions)

Here we get a different result: the best `max.depth` is no longer to be 1. Now it's obvious that we should choose `max.depth` to be 6. This is reasonable because now we have wider predictor "space", which results in deeper depth. On the other hand, `nrounds` = 250 is still a good point of stopping the training. We will also be in danger of over-fitting if we go above this point too much.

To sum up, for XGboost with interactions, we choose `eta` to be 0.01, `max.depth` to be 6 and `nrounds` to be 250. Here we can also see the first 10 most important features in the model after tuning.



It automatically gives us 5 clusters. The government feature is quite important, consistent with the analysis in single tree model. Also, we can see that many interaction terms are included, showing that adding these terms really help us improve the model.

The resulting RMSE of models after tuning is shown in Table 5.

Table 5: RMSE for XGBoost

	w/o Interaction	w/ Interaction
RMSE	0.279	0.269

We can see that the model with interactions truly does better than the one without interactions.

4.5 Random Forest

For the Random Forest model, we use `randomForest()` function in package {randomForest} to do the training. Because the data is naturally stratified, we set `strata` argument according to the Country. Also we set `replace` to be TRUE to allow "sampling of cases be done with replacement", which will let us do the quantification of uncertainty easier.

As known to us, bagging algorithm typically has less hyperparameters to tune than boosting. Here, We have two hyperparameters to tune, names and their meanings in the help document are below:

- `ntree`: "Number of trees to grow. This should not be set to too small a number, to ensure that every input row gets predicted at least a few times".
- `mtry`: "Number of variables randomly sampled as candidates at each split. Default value for regression is p/3".

4.5.1 Training process

As the same as XGboost, here we first build without possible interactions we've chosen before, and see what the result will be. Then we include those interactions later and see whether there is a difference.

Due to the fact that, for Random Forest, as long as `ntree` is large enough, we don't need to worry too much about its exact value. Therefore, our main focus here is `mtry`. We first set `ntree` to be 2000 to ensure enough number of trees. A plot of how test RMSE is changed with `mtry` from 1 to 5 is shown in Figure 13.

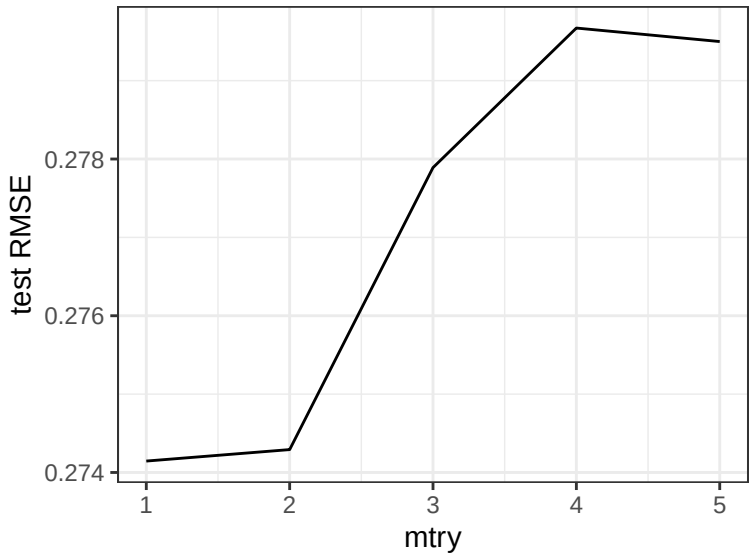


Figure 13: test RMSE vs mtry (RF with Interactions)

As shown in Figure 13, we can see that test RMSE get the lowest value when `mtry` is 1. This is reasonable because we don't have many predictors now, so only one variable randomly sampled as candidates at each split.

Given `mtry` to be 1, then we test whether the input value of `ntree` will affect the result a lot. A plot of how test RMSE is changed with `ntree` from 250 to 5000 is shown in Figure 14.

As shown in Figure 14, we can see that the test RMSE will fluctuate even when the `ntree` gets large enough. This result is consistent with the property of Random Forest Algorithm. So here we just need to set `ntree` to be 1000 and it's enough.

To sum up, for Random Forest without interactions, we choose `mtry` to be 1 and `ntree` to be 1000. Here we can take a look at the first 7 most important predictors in the model after tuning.

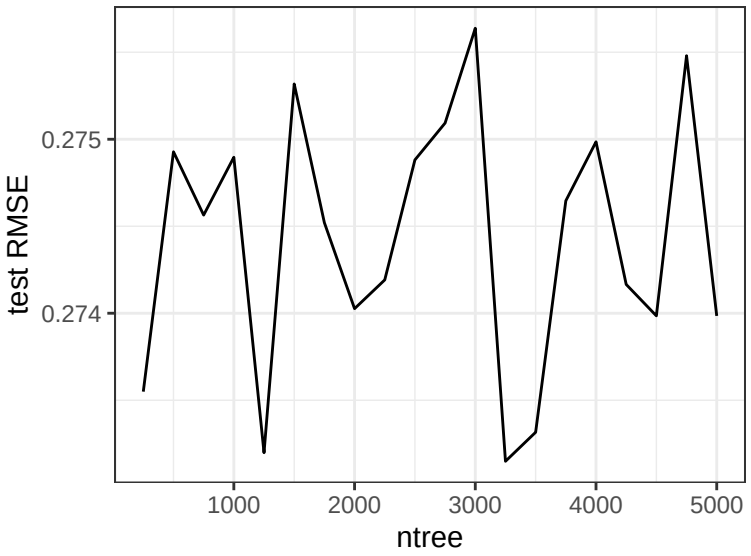
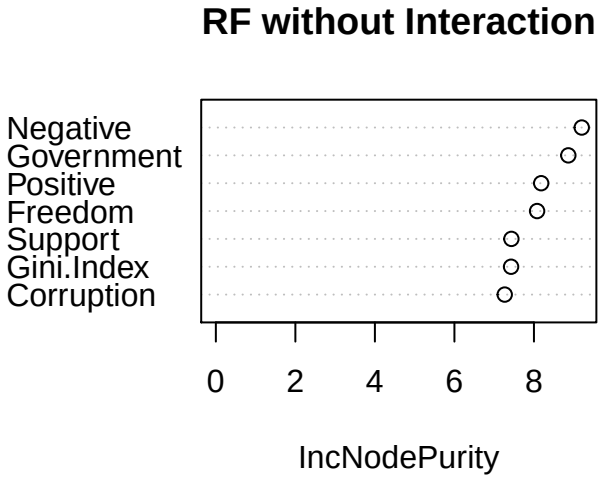


Figure 14: test RMSE vs ntree (RF without Interactions)



The result is quite consistent with the XGboost: Negative and Government are the most important order-one predictors.

4.5.2 Including interactions

Now we include interactions and go through the procedures above again.

Here we get a different result: the best `mtry` is no longer to be 1. Now we'd better choose `mtry` to be 20. This is reasonable because now we have wider predictor "space", which results in more choices. We will be in danger of losing the advantage of bootstrapping in Random Forest if we go above 20 too much.

Given `mtry` to be 20, then we test whether the input value of `ntree` will affect the result a lot. A plot of how test RMSE is changed with `ntree` from 500 to 10000 is shown in Figure 16.

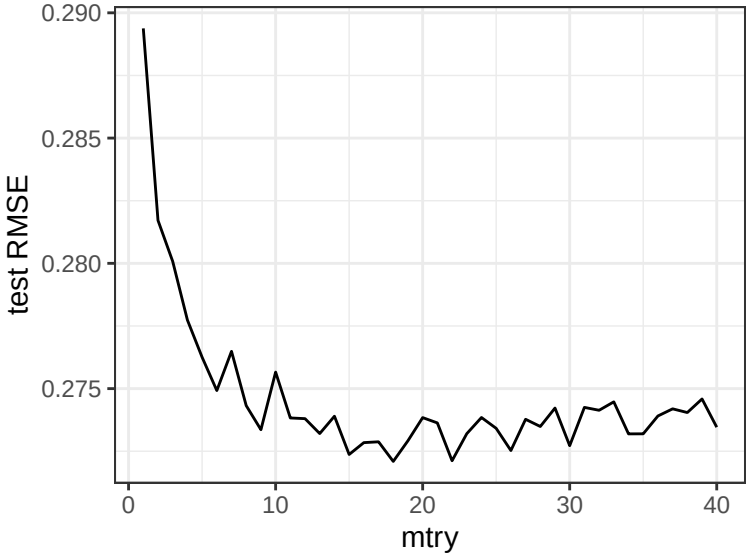


Figure 15: test RMSE vs mtry (RF with Interactions)

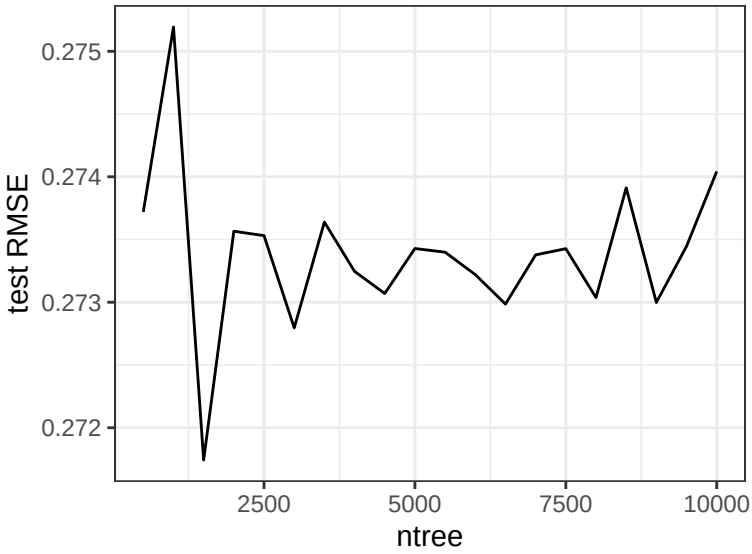
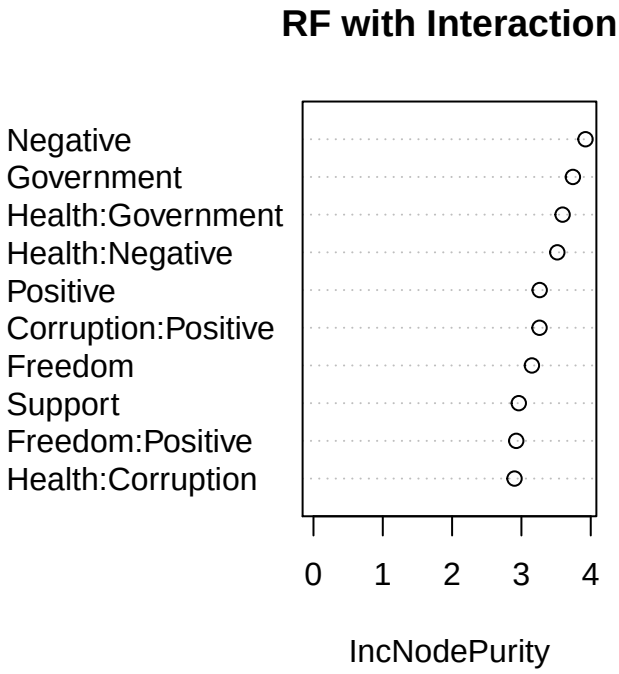


Figure 16: test RMSE vs ntree (RF with Interactions)

As shown in Figure 16, we can see that the test RMSE will also fluctuate even when the `ntree` gets large enough. Here we set `ntree` to be 2500, which is a little larger than the former setting.

To sum up, for Random Forest with interactions, we choose `mtry` to be 20 and `ntree` to be 2500. Here we can also take a look at the first 10 most important predictors in the model after tuning.



The result is also quite consistent with the XGboost: Negative and Government are the most important ones; many interaction terms also help improve the model.

The resulting test RMSE of both models after tuning is shown in Table 6.

Table 6: RMSE for Random Forest

	w/o Interaction	w/ Interaction
RMSE	0.276	0.273

We can see that the model with interactions truly does better than the one without interactions.

4.6 Selecting the final model

Now we can look at the table to select the best model based on testing RMSE.

Table 7: Difference and Original Data RMSE Comparison

	RMSE for original data	RMSE for difference data
Bayesian Model Averaging	0.417	0.277
Pruned Tree Model	0.683	0.285

Table 8: With/Without Interaction RMSE Comparison

	RMSE No Interaction	RMSE with Interaction
XGBoosting	0.279	0.269
Random Forest	0.276	0.273

It's quite obvious that the XGBoosting Tree Model with interaction has the smallest testing RMSE. Thus, we select XGBoosting Model with interaction as our final model.

5 Assessment of the final model

5.1 Point estimates & Prediction intervals

As shown above, our final model will be XGBoost with Interactions. We have already gotten the point estimate when computing the test RMSE. Here the focus will be how we get prediction intervals. We will use Bootstrap to simulate different training data sets and then build our model for each data. Therefore we can get a number of predicting values, where we can derive the prediction interval.

Due to the fact that it takes some time in training model one time, so we decide to only do 1000 times Bootstrap, which means we will get 1000 predicting values for each test point. Also, to accelerate the loop, we use 4 cores paralyzing computing with `foreach()` function.

The result is shown in Table 9.

Table 9: Point Estimates w/ Prediction Intervals

Country	True Value	Point Estimate	2.5% Bound	97.5% Bound
Afghanistan	2.694	2.606	2.451	2.771
Albania	5.004	4.717	4.558	4.843
Argentina	5.793	6.009	5.923	6.106
Armenia	5.062	4.462	4.228	4.728
Australia	7.177	7.276	7.188	7.380
Austria	7.396	7.269	7.156	7.389
Azerbaijan	5.168	5.228	5.137	5.349
Bangladesh	4.499	4.327	4.141	4.561
Belarus	5.234	5.567	5.434	5.677
Belgium	6.892	6.906	6.782	7.077
Benin	5.820	4.996	4.882	5.112
Bolivia	5.916	5.772	5.645	5.868
Bosnia and Herzegovina	5.887	5.191	5.042	5.327
Botswana	3.461	3.615	3.412	3.803
Brazil	6.191	6.374	6.266	6.473
Bulgaria	5.099	5.065	4.979	5.173
Burkina Faso	4.927	4.714	4.477	4.948
Cambodia	5.122	4.628	4.523	4.724
Cameroon	5.251	5.048	4.895	5.187
Canada	7.175	7.360	7.247	7.477
Chad	4.486	4.492	4.350	4.626
Chile	6.436	6.295	6.187	6.387
China	5.131	5.153	5.051	5.241
Colombia	5.984	6.200	6.113	6.295

Congo (Brazzaville)	5.490	4.884	4.703	5.059
Costa Rica	7.141	7.278	7.089	7.457
Croatia	5.536	5.405	5.276	5.552
Cyprus	6.276	6.062	5.944	6.155
Czech Republic	7.034	6.850	6.718	7.007
Denmark	7.649	7.498	7.394	7.609
Dominican Republic	5.433	5.575	5.449	5.672
Ecuador	6.128	5.708	5.559	5.870
Egypt	4.005	4.172	3.956	4.359
El Salvador	6.241	6.457	6.324	6.602
Estonia	6.091	6.007	5.914	6.091
Finland	7.858	7.756	7.644	7.862
France	6.666	6.673	6.591	6.770
Gabon	4.783	4.795	4.664	4.921
Georgia	4.659	4.309	4.073	4.521
Germany	7.118	7.105	6.965	7.253
Ghana	5.004	5.623	5.400	5.847
Greece	5.409	5.349	5.182	5.478
Guatemala	6.627	6.423	6.275	6.540
Guinea	5.252	4.825	4.582	5.027
Haiti	3.615	3.714	3.557	3.894
Honduras	5.908	5.946	5.805	6.067
Hungary	5.936	6.010	5.851	6.146
India	3.818	3.995	3.856	4.155
Indonesia	5.340	5.178	5.103	5.254
Iran	4.278	4.478	4.188	4.764
Iraq	5.100	4.570	4.329	4.801
Ireland	6.962	7.057	6.956	7.154
Israel	6.927	7.307	7.214	7.396
Italy	6.517	6.114	6.006	6.227
Japan	5.794	5.844	5.749	5.936
Jordan	4.639	4.817	4.682	4.940
Kazakhstan	6.008	5.851	5.663	6.047
Kenya	4.656	4.448	4.317	4.597
Kosovo	6.392	6.154	5.924	6.335
Kyrgyzstan	5.297	5.577	5.432	5.732
Latvia	5.901	5.983	5.875	6.119
Lebanon	5.167	5.138	5.039	5.247
Lithuania	6.309	6.255	6.090	6.420
Luxembourg	7.243	7.037	6.952	7.145
Macedonia	5.240	5.302	5.192	5.409
Madagascar	4.071	4.092	3.986	4.193
Malawi	3.335	3.295	3.114	3.459
Mali	4.416	4.807	4.664	4.957
Malta	6.910	6.780	6.702	6.855
Mauritania	4.314	4.702	4.586	4.817
Mexico	6.550	6.537	6.437	6.678
Moldova	5.682	5.323	5.107	5.526
Mongolia	5.465	5.365	5.279	5.440
Montenegro	5.650	5.697	5.614	5.784
Nepal	4.910	4.614	4.471	4.758

Netherlands	7.463	7.511	7.415	7.593
New Zealand	7.370	7.379	7.292	7.481
Nicaragua	5.819	6.188	5.988	6.374
Niger	5.164	4.679	4.471	4.898
Pakistan	5.472	5.835	5.682	5.981
Panama	6.281	6.545	6.394	6.665
Peru	5.680	5.765	5.679	5.842
Philippines	5.869	5.635	5.551	5.719
Poland	6.111	6.133	6.008	6.255
Portugal	5.920	5.690	5.583	5.803
Romania	6.151	6.079	5.944	6.200
Russia	5.513	5.535	5.434	5.639
Rwanda	3.561	3.269	3.082	3.411
Saudi Arabia	6.356	6.307	6.219	6.399
Senegal	4.769	4.702	4.615	4.802
Serbia	5.936	5.128	5.005	5.261
Slovakia	6.235	6.348	6.223	6.491
Slovenia	6.249	6.234	6.147	6.324
South Africa	4.884	4.524	4.269	4.722
South Korea	5.840	5.915	5.792	6.026
Spain	6.513	6.241	6.117	6.362
Sweden	7.375	7.311	7.214	7.423
Tajikistan	5.497	5.824	5.587	6.061
Tanzania	3.445	3.432	3.308	3.550
Thailand	6.012	6.030	5.933	6.115
Tunisia	4.741	4.243	4.074	4.426
Turkey	5.186	5.506	5.364	5.675
Turkmenistan	4.621	5.370	5.062	5.661
Uganda	4.322	3.921	3.803	4.038
Ukraine	4.662	4.368	4.276	4.458
United Arab Emirates	6.604	6.912	6.758	7.053
United Kingdom	7.233	7.116	7.004	7.220
United States	6.883	6.910	6.792	7.035
Uruguay	6.372	6.358	6.265	6.460
Uzbekistan	6.205	6.414	6.280	6.539
Venezuela	5.006	5.002	4.843	5.170
Vietnam	5.296	5.393	5.223	5.585
Yemen	3.058	3.229	3.064	3.373
Zambia	4.041	3.860	3.715	3.996
Zimbabwe	3.616	3.544	3.322	3.738

5.2 Evaluation Criteria

The result of each evalutaion criterion is shown in Table 10.

Table 10: Evaluation Criteria

Bias	Maximum.Deviation	test.RMSE	Coverage
0.048	0.824	0.273	0.452

Compared to our final model in part 1, now we have much better results regarding Maximum Deviation and

test RMSE. However, the coverage here is not ideal. It seems that we underestimate the variance of the data. On the other hand, this may be also due to the fact that we only have 1000 iterations of bootstrap to construct our prediction intervals, thus we don't get enough simulations to compute a more precise coverage rate.

5.3 Final Predictions Validation

We refit our XGBoost to the combined training and test data; create predictions and upload the results to Kaggle. Our public leaderboard score is about 0.35. This is bigger than our test MSE, which may indicate a problem of over-fitting.

Actually, we have tried lots of editions of our ensemble tree models. It seems that, as for this public leaderboard score, 0.35 is kind of a limit or bottleneck: we can't go over this value and get better score. Maybe this is also one problem with the ensemble tree models for this data, it's hard to do minor adjustment on the results.