Predicting Fatality of Covid-19

Based On Patient-Level Information

Yang Qu
Xiaoran Liu

Nov 23rd 2020

Introduction

The new type of coronavirus pneumonia (COVID-19) is spreading globally, posing an enormous threat to the world economy and human health. So far, there have been 58,969,491 confirmed cases and 1,393,225 deaths because of COVID-19 all over the world(Worldometer,2020). Throughout the pandemic, the shortage of medical resources and appropriate plans for the effective allocation of medical resources becomes a big problem for healthcare providers(Mukherjee, 2020). Therefore,the purpose of our project is to predict whether patients who have been diagnosed with COVID-19 will die, given their basic personal information, including History of present illness and Past History.

Methodology

Data Source And Acknowledgement

The dataset which contains a huge number of anonymized patient-related information was released by the Mexican government and then sorted by Tanmoy Mukherjee on Kaggle.

Dataset Overview

There are 566,602 patients in total, each patient represents a row in this dataset. We have both categorical predictor variables (such as sex, obesity, and ICU) and continuous predictor variables(for instance, age). Besides, the dataset includes the dates when patients entranced the hospital, confirmed, and died.

Procedure

We will first do the data pre-processing, to get rid of extreme values(like outliers) and missing values. At that stage, we need to do the variable transformation

and selection as well. After data pre-processing, we will use three different models: single classification tree, random forest, and boosting, to predict whether patients who have been diagnosed with COVID-19 will die and then evaluate the performance of each model.

Data Pre-processing

Data Cleaning

We import the data and use the code below to have a general understanding of this dataset.

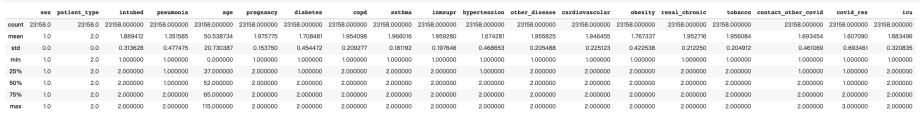
```
df.head()   df.columns   df.describe()
```

There are total 24 columns, except 'age' and the date-related data,the others are all categorical variables.

	id	sex	patient_type	entry_date	date_symptoms	date_died	intubed	pneumonia	age	pregnancy	diabetes	copd	asthma	inmsupr	hypertension	other_disease	cardiovascular	obesity	renal_chronic	tobacco	contact_other_covid	covid_res	icu
0	16169f	2	1	04-05-2020	02-05-2020	9999-99-99	97	2	27	97	2	2	2	2	2	2	2	2	2	2	2	1	97
1	1009bf	2	1	19-03-2020	17-03-2020	9999-99-99	97	2	24	97	2	2	2	2	2	2	2	2	2	2	99	1	97
2	167386	1	2	06-04-2020	01-04-2020	9999-99-99	2	2	54	2	2	2	2	2	2	2	2	1	2	2	99	1	2
3	0b5948	2	2	17-04-2020	10-04-2020	9999-99-99	2	1	30	97	2	2	2	2	2	2	2	2	2	2	99	1	2
4	0d01b5	1	2	13-04-2020	13-04-2020	22-04-2020	2	2	60	2	1	2	2	2	1	2	1	2	2	2	99	1	2

Figure 1: Part of the original dataset

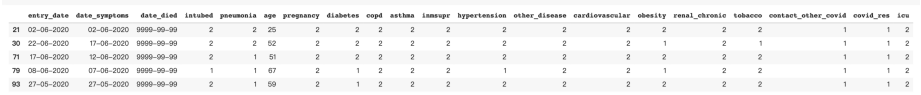
Since "97,98,99" in the nominal variables represent "Data missing or NA", we first try to delete the missing values, and also avoid accidentally delete the cases when patients' age are 97,98,99.



	sex	patient_type	intubed	pneumonia	age	pregnancy	diabetes	copd	asthma	inmsupr	hypertension	other_disease	cardiovascular	obesity	renal_chronic	tobacco	contact_other_covid	covid_res	icu
count	23158.0	23158.0	23158.000000	23158.000000	23158.000000	23158.000000	23158.000000	23158.000000	23158.000000	23158.000000	23158.000000	23158.000000	23158.000000	23158.000000	23158.000000	23158.000000	23158.000000	23158.000000	23158.000000
mean	1.0	2.0	1.889412	1.351585	50.538734	1.975775	1.708481	1.954098	1.966016	1.959280	1.674281	1.859825	1.946455	1.767237	1.952716	1.956084	1.693454	1.607090	1.883496
std	0.0	0.0	0.313628	0.477475	20.730387	0.153750	0.454472	0.209277	0.181192	0.197646	0.468653	0.205488	0.225123	0.422538	0.212250	0.204912	0.461069	0.693461	0.320835
min	1.0	2.0	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
25%	1.0	2.0	2.000000	1.000000	37.000000	2.000000	1.000000	2.000000	2.000000	2.000000	1.000000	2.000000	2.000000	2.000000	2.000000	2.000000	1.000000	1.000000	2.000000
50%	1.0	2.0	2.000000	1.000000	52.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	1.000000	2.000000
75%	1.0	2.0	2.000000	2.000000	65.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000
max	1.0	2.0	2.000000	2.000000	115.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	3.000000	2.000000

Figure 2: Basic statistical details after deleting missing values

After deleting the missing values, we find that the patients remaining are all female and have been diagnosed with COVID-19, also, we do not care about the id, so we drop these three columns.



	entry_date	date_symptoms	date_died	intubed	pneumonia	age	pregnancy	diabetes	copd	asthma	inmsupr	hypertension	other_disease	cardiovascular	obesity	renal_chronic	tobacco	contact_other_covid	covid_res	icu
21	02-06-2020	02-06-2020	9999-99-99	2	2	25	2	2	2	2	2	2	2	2	2	2	2	1	1	2
30	22-06-2020	17-06-2020	9999-99-99	2	2	52	2	2	2	2	2	2	2	2	1	2	1	1	1	2
71	17-06-2020	12-06-2020	9999-99-99	2	1	51	2	2	2	2	2	2	2	2	2	2	2	1	1	2
79	08-06-2020	07-06-2020	9999-99-99	1	1	67	2	1	2	2	2	1	2	2	1	2	2	1	1	2
93	27-05-2020	27-05-2020	9999-99-99	2	1	59	2	1	2	2	2	2	2	2	2	2	2	1	1	2

Figure 3: Part of dataset after dropping sex, id, and patient_type

Variable Transformation

Next, we deal with the date-related data, including three columns, entry_date, date_symptoms, and date_died. In this step, we will generally complete two tasks, one is to convert the date-related values into the number of days, and the other is to classify the patient as dead or recovered using the column called date_died.

For the first task, we create a new variable called 'entry_symptoms', which is the number of days between the patients' entrance date and their symptoms date. For the second one, we assume that the patients whose 'date_died' are NA(9999-99-99) are recovered, while the others who have certain death dates are dead, and then create a new variable called 'fatality'.

intubed	pneumonia	age	pregnancy	diabetes	copd	asthma	inmsupr	hypertension	other_disease	cardiovascular	obesity	renal_chronic	tobacco	contact_other_covid	covid_res	icu	fatality	entry_symptoms
2.0	2.0	25.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	1.0	1.0	2.0	2	0
2.0	2.0	52.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	1.0	2.0	1.0	1.0	1.0	2.0	2	5
1.0	1.0	67.0	2.0	1.0	2.0	2.0	2.0	1.0	2.0	2.0	1.0	2.0	2.0	1.0	1.0	2.0	2	31
2.0	1.0	59.0	2.0	1.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	1.0	1.0	2.0	2	0
2.0	2.0	52.0	2.0	1.0	2.0	2.0	2.0	1.0	2.0	1.0	2.0	2.0	2.0	2.0	1.0	2.0	2	2

Figure 4: Part of the dataset after creating 'entry_symptoms' and 'fatality'

	intubed	pneumonia	age	pregnancy	diabetes	copd	asthma	inmsupr	hypertension	other_disease	cardiovascular	obesity	renal_chronic	tobacco	contact_other_covid	covid_res	icu	fatality	entry_symptoms
count	15332.000000	15332.000000	15332.000000	15332.000000	15332.000000	15332.000000	15332.000000	15332.000000	15332.000000	15332.000000	15332.000000	15332.000000	15332.000000	15332.000000	15332.000000	15332.000000	15332.000000	15333.000000	15333.000000
mean	1.883968	1.348356	50.201474	1.973911	1.706627	1.952191	1.966541	1.958518	1.677602	1.952322	1.945213	1.758132	1.950430	1.957214	1.697691	1.411623	1.882598	1.813344	37.497750
std	0.320274	0.476465	21.057919	0.159406	0.455323	0.213368	0.179839	0.199408	0.467409	0.213091	0.227572	0.422039	0.217061	0.202382	0.459273	0.492144	0.321809	0.389648	60.811735
min	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000
25%	2.000000	1.000000	37.000000	2.000000	1.000000	2.000000	2.000000	2.000000	1.000000	2.000000	2.000000	2.000000	2.000000	2.000000	1.000000	1.000000	2.000000	2.000000	2.000000
50%	2.000000	1.000000	52.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	1.000000	2.000000	2.000000	5.000000
75%	2.000000	2.000000	65.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	61.000000
max	2.000000	2.000000	115.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	335.000000

Figure 5: Basic statistical details about dataset above

From Figure 5, it can be seen that the entry_symptoms predictor has a very big variance. We draw a boxplot to be more intuitive. The boxplot is shown below.

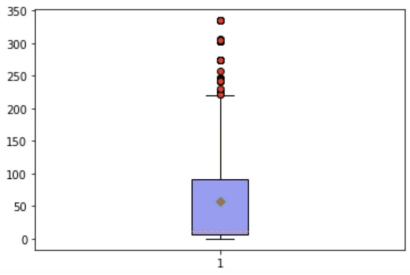


Figure 6: Boxplot for entry_symptoms

To solve this problem,we use discretization. We transform this continuous variable to a categorical one, by dividing the number of days between entrance and symptoms into six intervals, which are 0, 1, 1-3, 3-5, 5-10, and more than 10 days. The histogram for entry_symptoms after discretization is shown below.

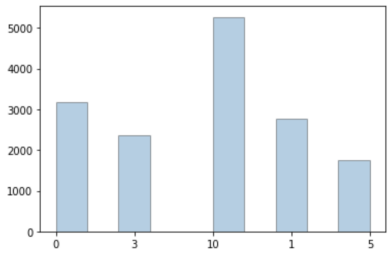


Figure 7: Histogram for entry_symptoms after discretization

Finally,we finish the data pre-processing and generate the dataset below.

	intubed	pneumonia	age	pregnancy	diabetes	copd	asthma	inmsupr	hypertension	other_disease	cardiovascular	obesity	renal_chronic	tobacco	contact_other_covid	covid_res	icu	fatality	date_diff_level
21	2.0	2.0	25.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	1.0	1.0	2.0	2	0
30	2.0	2.0	52.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	1.0	2.0	1.0	1.0	1.0	2.0	2	3
79	1.0	1.0	67.0	2.0	1.0	2.0	2.0	2.0	1.0	2.0	2.0	1.0	2.0	2.0	1.0	1.0	2.0	2	10
93	2.0	1.0	59.0	2.0	1.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	1.0	1.0	2.0	2	0
215	2.0	2.0	52.0	2.0	1.0	2.0	2.0	2.0	1.0	2.0	1.0	2.0	2.0	2.0	2.0	1.0	2.0	2	1

Figure 8: Part of the final dataset

	intubed	pneumonia	age	pregnancy	diabetes	copd	asthma	inmsupr	hypertension	other_disease	cardiovascular	obesity	renal_chronic	tobacco	contact_other_covid	covid_res	icu	fatality
count	18458.000000	18458.000000	18458.000000	18458.000000	18458.000000	18458.000000	18458.000000	18458.000000	18458.000000	18458.000000	18458.000000	18458.000000	18458.000000	18458.000000	18458.000000	18458.000000	18458.000000	18458.000000
mean	1.889479	1.366508	50.030285	1.974808	1.711561	1.953624	1.966410	1.958338	1.681006	1.954383	1.946527	1.776303	1.951457	1.957688	1.695092	1.643407	1.883899	1.834381
std	0.313547	0.481863	21.140026	0.156713	0.453048	0.210303	0.180776	0.199821	0.466100	0.208659	0.224980	0.416733	0.214916	0.201306	0.460381	0.702664	0.320355	0.377149
min	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
25%	2.000000	1.000000	36.000000	2.000000	1.000000	2.000000	2.000000	2.000000	1.000000	2.000000	2.000000	2.000000	2.000000	2.000000	1.000000	1.000000	2.000000	2.000000
50%	2.000000	1.000000	52.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000
75%	2.000000	2.000000	65.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000
max	2.000000	2.000000	115.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	3.000000	2.000000	2.000000

Figure 9: Basic statistical details about the final dataset

Modelling

Classification Tree

We create and train the classification tree model using DecisionTreeClassifier from Scikit-learn library. We split the dataset into traning set and test set by using the following code:

```
y = df['fatality']
x = df.drop('fatality',axis = 1)

x_train, x_test, y_train, y_test = train_test_split(x, y,
                                   test_size=0.2, random_state=42)
```

Default Version

We first examine the performance of the model where we set all parameters as default values. By reading the Scikit-learn documentation, we use the default setting as the following:

Parameters	Values
criterion	"gini"
max_depth	None
min_samples_leaf	1
min_impurity_decrease	0.0

```
dt = DecisionTreeClassifier()
dt.fit(x_train,y_train)
dt.score(x_test, y_test)
```

The output score is:

```
0.7789815817984832
```

Comparision between Gini and Entropy

We use the following code to determine whether gini criterion performs better or entropy criterion performs better for our data.

```
dt1 = DecisionTreeClassifier(random_state = 66)
score = cross_val_score(dt1,x_train,y_train,cv=10).mean()
print('gini score: %.5f'%score)
dt2 = DecisionTreeClassifier(criterion = 'entropy',random_state = 66)
score = cross_val_score(dt2,x_train,y_train,cv=10).mean()
print('entropy score: %.5f'%score)
```

The output is:

```
gini score: 0.77841
entropy score: 0.77888
```

It is readily apparent that the result using entropy is slightly better than using gini, therefore, we decide to use criterion = 'entropy' throughout fitting the classification tree.

Tuning Parameter: Max_depth

We now change the tuning parameters to improve the score. We change only one parameter, the max_depth first, and plot the graph of the test score versus the max depth of the classification tree.

```
ScoreAll = []
for i in range(1,30,1):
    dt = DecisionTreeClassifier(criterion = 'entropy',
    max_depth = i,random_state = 66)
    score = cross_val_score(dt,x,y,cv=10).mean()
    ScoreAll.append([i,score])
ScoreAll = np.array(ScoreAll)
```

```
max_score = np.where(ScoreAll==np.max(ScoreAll[:,1]))[0][0]
print("best parameter and score:",ScoreAll[max_score])
# print(ScoreAll[,0])
plt.figure(figsize=[20,5])
plt.plot(ScoreAll[:,0],ScoreAll[:,1])
plt.show()
```

The output is:

```
best parameter and score: [6.         0.85231224]
```

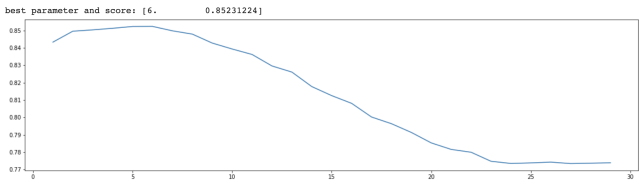


Figure 10: Plot of the score v.s. max_depth

The score first increases as we increases the max depth of the classification tree and then decreases when the max depth is greater than 8.

Tuning Parameters: Grid Search

We import GridSearchCV from Scikit-learn library. By grid searching, we tune parameters in order to optimize the accuracy.

```
param = {'criterion':['entropy'],'max_depth':[3,4,5,6,7],
         'min_samples_leaf': [1,2,3,4,5],
         'min_impurity_decrease':[0.01,0.02,0.03,0.04,0.05]}
grid = GridSearchCV(DecisionTreeClassifier(),
                    param_grid=param,cv=5)
grid.fit(x_train,y_train)
print('best classifier:',grid.best_params_,
        'best score:', grid.best_score_)
```

The output is:

```
best classifier: {'criterion': 'entropy',
                  'max_depth': 3,
                  'min_impurity_decrease': 0.01,
                  'min_samples_leaf': 1}
best score: 0.84247604251761
```

Refitting: Train Score and Test Score

Now we re-fit the model using the corresponding tuning parameters.

Parameters	Values
criterion	“entropy”
max_depth	3
min_samples_leaf	1
min_impurity_decrease	0.01

```
dt3 = DecisionTreeClassifier(criterion = 'entropy',
                             max_depth=3,
                             min_samples_leaf=1,
                             min_impurity_decrease=0.01,
                             random_state = 66)
dt3.fit(x_train,y_train)
y_pred = dt3.predict(x_test)
print('train set score', dt3.score(x_train,y_train),
        'test set score',dt3.score(x_test,y_test))
```

The output is:

```
train set score 0.842475958282541 test set score 0.8464247020585048
```

To visualize the classification tree,we run the following Python code:

```
dt3.feature_importances_

dot_data = tree.export_graphviz(dt3,
                        feature_names = x.columns,
                       filled=True)
graph = graphviz.Source(dot_data)
graph
```

The output is:

```
array([0.6144566 , 0.        , 0.13051284, 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.25503057, 0.        , 0.        ])
```

The graph shown as below:

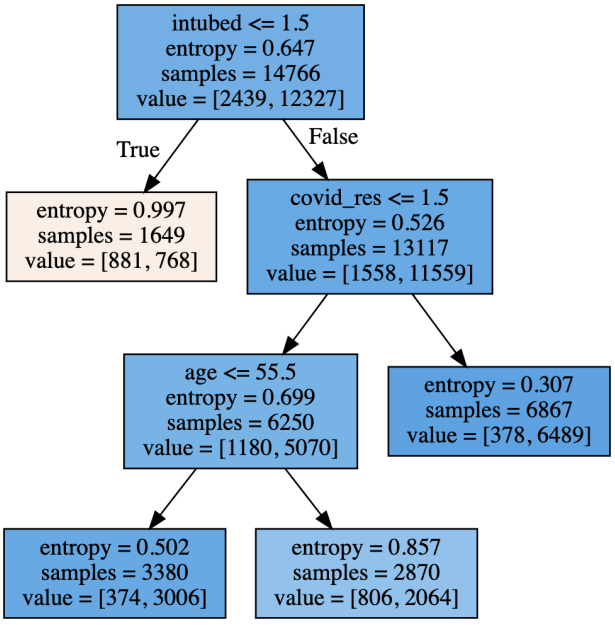


Figure 11: Classification Tree

Random Forest

Default Version

We create and train the random forest model using RandomForestClassifier from Scikit-learn library. We use all the observations to train the model since one of the advantages of using random forest is for-free cross validation.
We first examine the performance of the model where we set all parameters as default values. By reading the scikitlearn documentation, we use the default setting as the following:

Parameters	Values
n_estimators	100
criterion	"gini"
max_depth	None
min_samples_split	2
min_samples_leaf	1
max_features	"auto"

We use the following Python code to build the model and examine the performance of it:

```
from sklearn.ensemble import RandomForestClassifier
```

```
y = df['fatality']
x = df.drop('fatality',axis = 1)

# all default
rf0 = RandomForestClassifier(oob_score=True, random_state=10)
rf0.fit(x,y)
print(rf0.oob_score_)
y_predprob = rf0.predict_proba(x)[:,1]
print("AUC Score (Train): %f" % metrics.roc_auc_score(y, y_predprob))
print(rf0.feature_importances_)
```

The output is:

```
0.8252248347599956
AUC Score (Train): 0.990963
[0.12004049 0.0456253  0.42619197 0.00258172 0.02771941 0.01518592
 0.00950621 0.01235143 0.02652667 0.01306905 0.01601859 0.02990498
 0.0152589  0.0142912  0.02859993 0.06717499 0.02562033 0.10433292]
```

To find the top 5 important variables,we run the following Python code:

```
feat_importances = pd.Series(rf0.feature_importances_, index=x.columns)
feat_importances.nlargest(5).plot(kind='barh')
```

The plot shown as below:

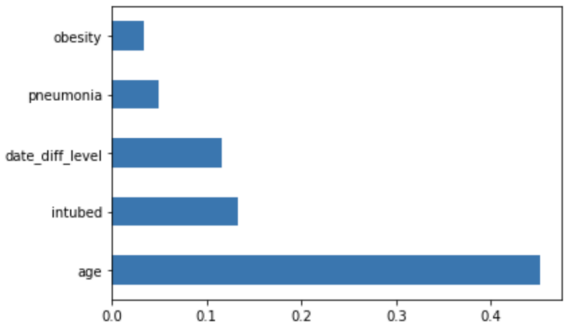


Figure 12: Variable Importance Plot_RF Default

By using the default setting,we have a prediction accuracy of 0.825. The top 5 important variables are Age,Intubed, date_diff_level, Pneumonia and Obesity.

Tuning Parameters - Grid Search

We now adjust the tuning parameters and try to get a improvement on the predict accuracy/oob score by grid search.

We import GridSearchCV from Scikit-learn library. Using greedy search, the later parameter choosing will depend on the previous one.We start from choosing the n_estimators, i.e, the number of tree in the forest.

```
from sklearn.model_selection import GridSearchCV

param_test1 = {'n_estimators':range(10,201,10)}
gsearch1 = GridSearchCV(estimator =
RandomForestClassifier(min_samples_split=100,
                min_samples_leaf=20,
                max_depth=8,max_features='sqrt',
                random_state=10),
                param_grid = param_test1, scoring='roc_auc',cv=5)
gsearch1.fit(x,y)
print(gsearch1.best_params_, gsearch1.best_score_)
```

The output is:

```
{'n_estimators': 70} 0.8173103675660242
```

We fix the number of estimators as 70. Now we adjust the max_depth.

```
param_test2 = {'max_depth':range(2,18,2)}
gsearch2 = GridSearchCV(estimator = RandomForestClassifier(
                    n_estimators=70,
                    min_samples_split=100,
                    min_samples_leaf=20,max_features='sqrt',
                    oob_score=True, random_state=10),
                    param_grid = param_test2,
                    scoring='roc_auc',iid=False, cv=5)
gsearch2.fit(x,y)
print(gsearch2.best_params_, gsearch2.best_score_)
```

The output is:

```
{'max_depth': 8} 0.8173103675660242
```

We fix the max_depth as 8.Now we adjust the min_samples_split and min_samples_leaf together.

```
param_test3 = {'min_samples_split':range(80,150,20),
                'min_samples_leaf':range(5,50,5)}
gsearch3 = GridSearchCV(estimator =RandomForestClassifier(
                        n_estimators=70, max_depth=8,
                        max_features='sqrt',oob_score=True,
                        random_state=10),
                        param_grid = param_test3,
```

```
                        scoring='roc_auc',iid=False, cv=5)
gsearch3.fit(x,y)
print(gsearch3.best_params_, gsearch3.best_score_)
```

The output is:

```
{'min_samples_leaf': 10, 'min_samples_split': 80} 0.8182691645547792
```

We fix the min_samples_split as 80 and min_samples_leaf as 10.Now we adjust the max_features.

```
param_test4 = {'max_features':range(2,18,1)}
gsearch4 = GridSearchCV(estimator = RandomForestClassifier(
                        n_estimators=70,
                        max_depth=8,
                        min_samples_split=80,
                        min_samples_leaf=10 ,
                        oob_score=True, random_state=10),
                        param_grid = param_test4,
                        scoring='roc_auc',iid=False, cv=5)
gsearch4.fit(x,y)
print(gsearch4.best_params_, gsearch4.best_score_)
```

The output is:

```
{'max_features': 4} 0.8182691645547792
```

Now we re-fit the model using the corresponding tuning parameters.

Parameters	Values
n_estimators	70
max_depth	8
min_samples_split	80
min_samples_leaf	10
max_features	4

```
rf1 = RandomForestClassifier(n_estimators= 70, max_depth=8,
                                min_samples_leaf=10,
                                max_features=4,
                                oob_score=True, random_state=10)
rf1.fit(x,y)
rf1.oob_score_
print(rf1.feature_importances_)
```

The output is:

```
0.8590313143352476
[0.42408717 0.12972798 0.15079871 0.00139062 0.02527071 0.00403375
 0.00133224 0.0023524  0.02621899 0.00175527 0.00342501 0.00564984
 0.00579134 0.0020879  0.01153909 0.15610273 0.03071886 0.01771739]
```

To find the top 5 important variables,we run the following Python code:

```
feat_importances = pd.Series(rf0.feature_importances_, index=x.columns)
feat_importances.nlargest(5).plot(kind='barh')
```

The plot shown as below:

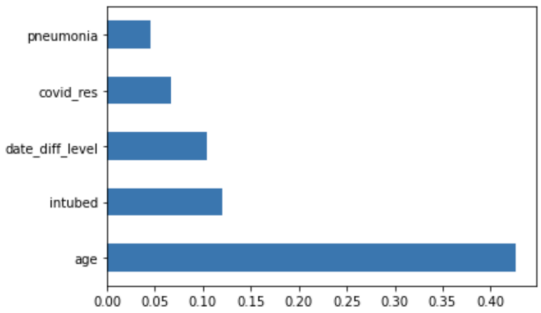


Figure 13: Variable Importance Plot_RF Tuning Parameters

After tuning parameters,we have a improved prediction accuracy of 0.859. The top 5 important variables are Age,Intubed, date_diff_level, Covid_res and Pneumonia.

Boosting

Gradient Boosting

We create and train the gradient boosting model using GradientBoostingClassifier from Scikit-learn library.
We first divide the dataset into traning set and test set and find the best learning rate by using the following code:

```
x_train, x_test, y_train, y_test = train_test_split(x, y,
                                    test_size=0.2, random_state=42)

lr_list = [0.05, 0.075, 0.1, 0.25, 0.5, 0.75, 1]

for learning_rate in lr_list:
    gb_clf = GradientBoostingClassifier(n_estimators=20,
                                learning_rate=learning_rate,
                                max_features=2, max_depth=2,
                                random_state=0)
gb_clf.fit(x_train, y_train)

print("Learning rate: ", learning_rate)
print("Accuracy score (training):", gb_clf.score(x_train, y_train))
print("Accuracy score (validation)",gb_clf.score(x_test, y_test))
```

The output is:

```
Learning rate:  0.05
Accuracy score (training): 0.8348232425843153
Accuracy score (validation) 0.83261105092091
Learning rate:  0.075
Accuracy score (training): 0.8348232425843153
Accuracy score (validation) 0.83261105092091
Learning rate:  0.1
Accuracy score (training): 0.8397670323716646
Accuracy score (validation) 0.8388407367280607
Learning rate:  0.25
Accuracy score (training): 0.8545984017337126
Accuracy score (validation) 0.8534669555796316
Learning rate:  0.5
Accuracy score (training): 0.8545306785859407
Accuracy score (validation) 0.855092091007584
Learning rate:  0.75
Accuracy score (training): 0.8548692943248002
Accuracy score (validation) 0.8567172264355363
Learning rate:  1
Accuracy score (training): 0.8533116619260463
Accuracy score (validation) 0.8545503791982665
```

We choose 0.75 as our learning rate since it gives the best performance on both training set and validation set.
We use the following Python code to build the model and examine the performance of it:

```
gb_clf2 = GradientBoostingClassifier(n_estimators=20,
                                     learning_rate=0.75,
                                     max_features=2,
                                     max_depth=2,
                                     random_state=0)
gb_clf2.fit(x_train, y_train)
predictions = gb_clf2.predict(x_test)

print("Confusion Matrix:")
print(confusion_matrix(y_test, predictions))

print("Classification Report")
print(classification_report(y_test, predictions))
```

The output is:

```
Confusion Matrix:
[[ 154  464]
 [  65 3009]]
```

```
Classification Report
              precision    recall  f1-score   support

           1       0.70      0.25      0.37       618
           2       0.87      0.98      0.92      3074

    accuracy                           0.86      3692
   macro avg       0.78      0.61      0.64      3692
weighted avg       0.84      0.86      0.83      3692
```

To find the top 5 important variables,we run the following Python code:

```
gb_clf2.feature_importances_

feat_importances = pd.Series(gb_clf2.feature_importances_,
                             index=x.columns)
feat_importances.nlargest(5).plot(kind='barh')
```

The output is:

```
array([3.82104872e-01, 1.81913329e-01, 7.75160802e-02, 3.96470952e-03,
       1.80246054e-02, 7.19061622e-04, 4.49845684e-04, 2.18785324e-04,
       1.08708797e-01, 1.01214500e-03, 7.79165942e-04, 5.15157284e-03,
       0.00000000e+00, 1.81211554e-03, 1.91654633e-02, 1.71416604e-01,
       2.41496573e-02, 2.89319013e-03])
```

The plot shown as below:

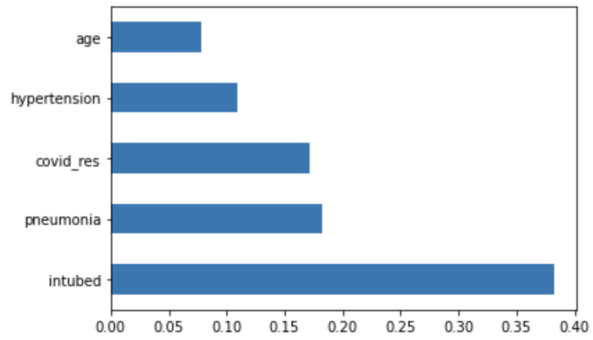


Figure 14: Variable Importance Plot_Gradient Booting

After tuning parameters,we have a improved prediction accuracy of 0.8567. The top 5 important variables are Intubed, Pneumonia, Covid_res, Hypertension and Age.

XGBoost

We create and train the XGBoost model using XGBClassifier from xgboost library.
We first divide the dataset into traning set and test set.

```
x_train, x_test, y_train, y_test = train_test_split(x, y,
                                    test_size=0.2, random_state=42)
```

We examine the performance of the model where we set all parameters as default values. By reading the XGBoost documentation, we use the default setting as the following:

Parameters	Values
base_score	0.5
learning_rate	0.1
max_depth	3
n_estimators	100
min_child_weight	1

```
xgb_clf3 = XGBClassifier()
xgb_clf3.fit(x_train, y_train)

score = xgb_clf3.score(x_test, y_test)
print(score)
```

The output is:

```
0.8580715059588299
```

To find the top 5 important variables,we run the following Python code:

```
xgb_clf3.feature_importances_

feat_importances = pd.Series(xgb_clf3.feature_importances_, index=x.columns)
feat_importances.nlargest(5).plot(kind='barh')
```

The output is:

```
array([0.34893763, 0.24987316, 0.05811411, 0.00360326, 0.02819734,
       0.01367755, 0.01048905, 0.01532699, 0.03117329, 0.01278323,
       0.0086069 , 0.01742312, 0.01083909, 0.00928206, 0.02856785,
       0.13041076, 0.00714213, 0.01555254], dtype=float32)
```

The plot shown as below:

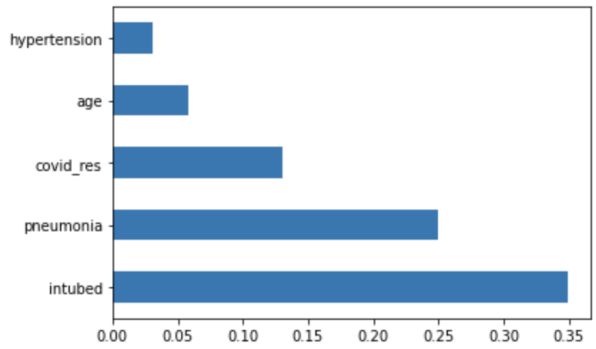


Figure 15: Variable Importance Plot_XGBoost

After tuning parameters,we have a improved prediction accuracy of 0.858. The top 5 important variables are Intubed, Pneumonia, Covid_res, Age and Hypertension.

Comparison among Models

Here we compare the accuracy of six models.

Model Name	Accurcay
Classification Tree (Default)	0.779
Classification Tree (Tuning Parameters)	0.846
Random Forest (Default)	0.825
Random Forest (Tuning Parameters)	0.859
Gradient Boosting	0.8567
XGBoost	0.858

Here we compare the variables involved in each model.

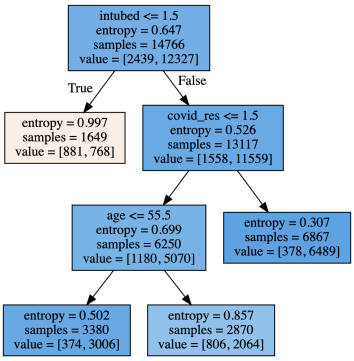


Figure 16: Classification Tree

Model Name	1st	2nd	3rd	4th	5th
RF* (Default)	Age	Intubed	date_diff_level	Pneumonia	Obesity
RF* (Tuning Parameters)	Age	Intubed	date_diff_level	Covid_res	Pneumonia
Gradient Boosting	Intubed	Pneumonia	Covid_res	Hypertension	Age
XGBoost	Intubed	Pneumonia	Covid_res	Age	Hypertension

*RF: Random Forest

Conclusion and Discussion

By tuning parameters, the classification tree model gets improved on test score from 0.779 to 0.846, the random forest model gets improved on prediction accuracy from 0.825 to 0.859. We get similar prediction accuracy on gradient boosting and XGBoost, 0.8567, and 0.858. Age, Intubed, and Pneumonia are commonly listed as one of the top 5 important variables.

This dataset is limited and does not include some other variables, such as location, economic classes, and population density, which may be essential variables for the result of our predictions. Besides, we eliminated some variables like sex and the patient type because of missing values, which can also be important. Additionally, in the stage of data pre-processing, we just simply classify the patients with NA death date as recovered, but we don't know if they are still alive, since the dataset only contains the information up to four months ago. Moreover, we do not consider the interaction terms in this study.

In the future study, we will try to introduce more reasonable variables and interaction terms and then fill out the missing values using statistical methods if possible.

Citation

Coronavirus Cases:. (2020). Retrieved November 23, 2020, from $https://www.worldometers.info/coronavirus/?utm_campaign = homeAdvegas1?$

Mukherjee, T. (2020, July 22). COVID-19 patient pre-condition dataset. Retrieved November 23, 2020, from $https://www.kaggle.com/tanmoyx/covid19-patient-precondition-dataset$

Nelson, D. (n.d.). Gradient Boosting Classifiers in Python with Scikit-Learn. Retrieved November 23, 2020, from $https://stackabuse.com/gradient-boosting-classifiers-in-python-with-scikit-learn/$

Appendix

Data Weblink

https://www.kaggle.com/tanmoyx/covid19-patient-precondition-dataset

Python code/Jupyter notebook

https://github.com/ChristyQu/CURC