COVID19 Diagnosis Prediction

Group : MaoLiang Liu, Chenlu Huang, Jun Guo

Table of Contents

1. Introduction

In December 2019, the first known case of COVID-19 was identified in Wuhan, China. COVID-19, coronavirus disease 2019, is a contagious disease caused by severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2). The disease has since then spread worldwide and resulted in an ongoing pandemic that changes the way of life in every way imaginable. As of August 17, 2021, there are more than 208 million people infected with COVID-19 worldwide and more than 4.4 million deaths because of it. Even in the United States alone, we have around 37 million reported cases and 623 thousand deaths. In fact, according to the CDC, COVID-19 cases have been on the rise recently because of the Delta variant, which causes more infections and spreads faster than earlier forms of the virus that causes COVID-19.

At the beginning of the COVID-19 pandemic, one of the major issues faced by every country around the world is the overwhelming stress the pandemic has put on their medical system. Even one of the most developed counties in the world, the United States, is not prepared to handle the sharp increase in the demand for medical resources let alone other less developed countries. Of course, we have come a long way since then and situations have improved compared to a year ago. However, with the recent increase in COVID-19 cases around the world because of the Delta variant, one couldn't help but wonder if our medical system is able to properly handle another wave of the COVID-19 pandemic. Therefore, we would like to know whether we can utilize supervised learning to help predict COVID-19 patients. If so, we will be able to help reduce the stress on our medical systems by identifying COVID-19 patients efficiently and reduce the spread of the virus. Of course, this will be most helpful for the less developed countries, such as India, where medical resources are extremely limited.

2. Related Work

An article, "Machine learning-based prediction of COVID-19 diagnosis based on symptoms", was published on January 4, 2021, very similar to what we accomplished in our project. The article talks about the development of a model that detects COVID-19 cases by simple features accessed by asking basic questions. In fact, they used the same public dataset from the Israeli Ministry of Health but from different dates. In the article, they utilized gradient boosting machine model to generate their predictions. In addition to the gradient boosting model, we also utilized multiple models, and then picked the model with the best performance. The dataset used is imbalanced with only 10% positive test result records but in the article, there is no mention of how it was addressed during the data preprocessing or sampling phase. We, on the other hand, utilized different ways of sampling imbalanced datasets to ensure proper training and testing of our models.

3. Solution/Method (Innovations)

Feature Selection
A wide range of feature selection methods including filter, wrapper, embedded methods are used and compared in order to refine the performance results.

Imbalanced sampling technique
To deal with an imbalanced dataset (99% negative and 1% positive), we utilized multiple imbalanced sampling methods to sample the dataset evenly.

Multiple classifiers
We are not limited to using only one or two models. Instead, we covered several mainstream models to see what differences in performance they have in predicting our dataset.

4. Data Understanding

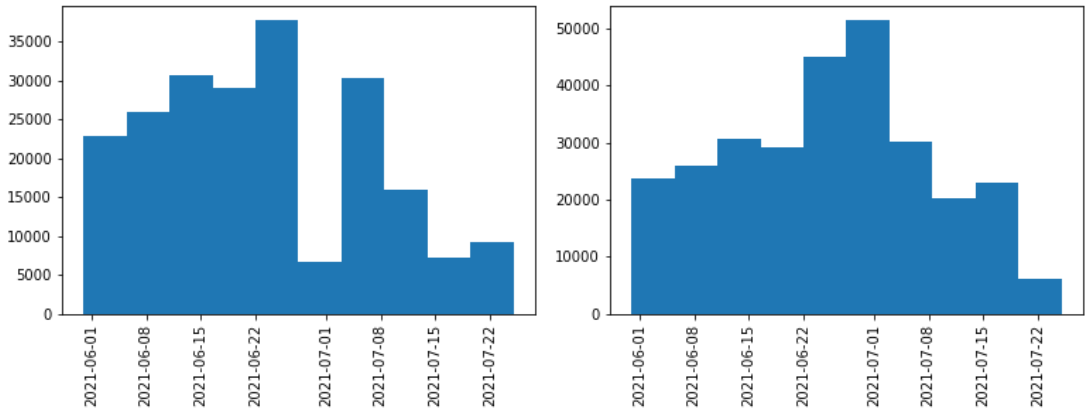
We utilized the dataset, which contains all the individual tests taken by Israeli citizens since February 2020, retrieved from Israeli Ministry of Health website for our project. Both the website and the record within the dataset is in Hebrew but, with the help of Google translator, we were able to translate the contents to English and ensure our understanding of the dataset is correct. The dataset contains the below 10 columns:

1. date: Arrival date of the test at the lab in DD/MM/YY format
2. cough: Did cough symptoms appear before testing? 1 –Yes, 0–No, NULL –Unknown
3. fever: Did fever appear before testing? 1 –Yes, 0 –No, NULL –Unknown
4. sore_throat: Did sore throat symptoms appear before testing? 1 –Yes, 0 –No, NULL –Unknown
5. shortness_of_breath: Did cough shortness of breath symptoms appear before testing? 1 –Yes, 0 –No, NULL –Unknown
6. head_ache: Did headache symptoms appear before testing? 1 –Yes, 0 –No, NULL –Unknown
7. gender: test taker's sex: male/female/ NULL (unknown)
8. corona_result: Results of first Covid-19 test, by category:
   Positive –carrying Covid-19
   Negative –not carrying Covid-19
   Other –Not tested, in testing, inconclusive
9. age: Indicator of the test taker's age -60 or over (1) or below 60 (0)
10. test_indication: What is the indication for testing?
    Abroad –arrived from abroad
    Contact_with_confirmed –contact with a confirmed case
    Other –other indication or not specified

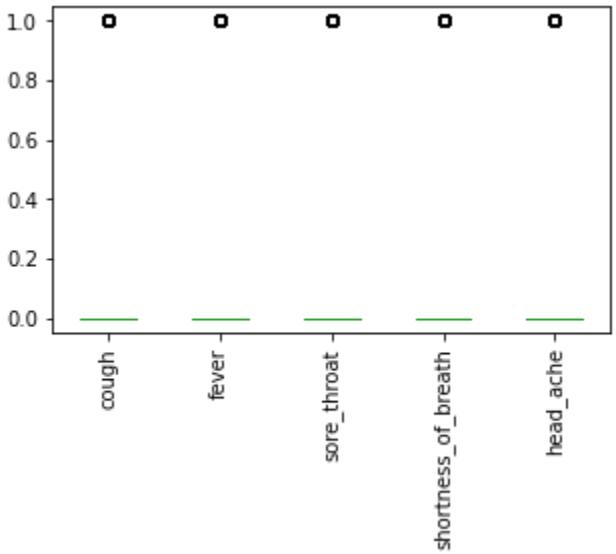
The dataset is constantly updating but, due to the size and the computation power of our laptops, we have limited our dataset to around 500 thousand records by limiting the dates to be between May 30, 2021 and July 25, 2021.

Within our dataset, we have the below null values in the age and gender columns:

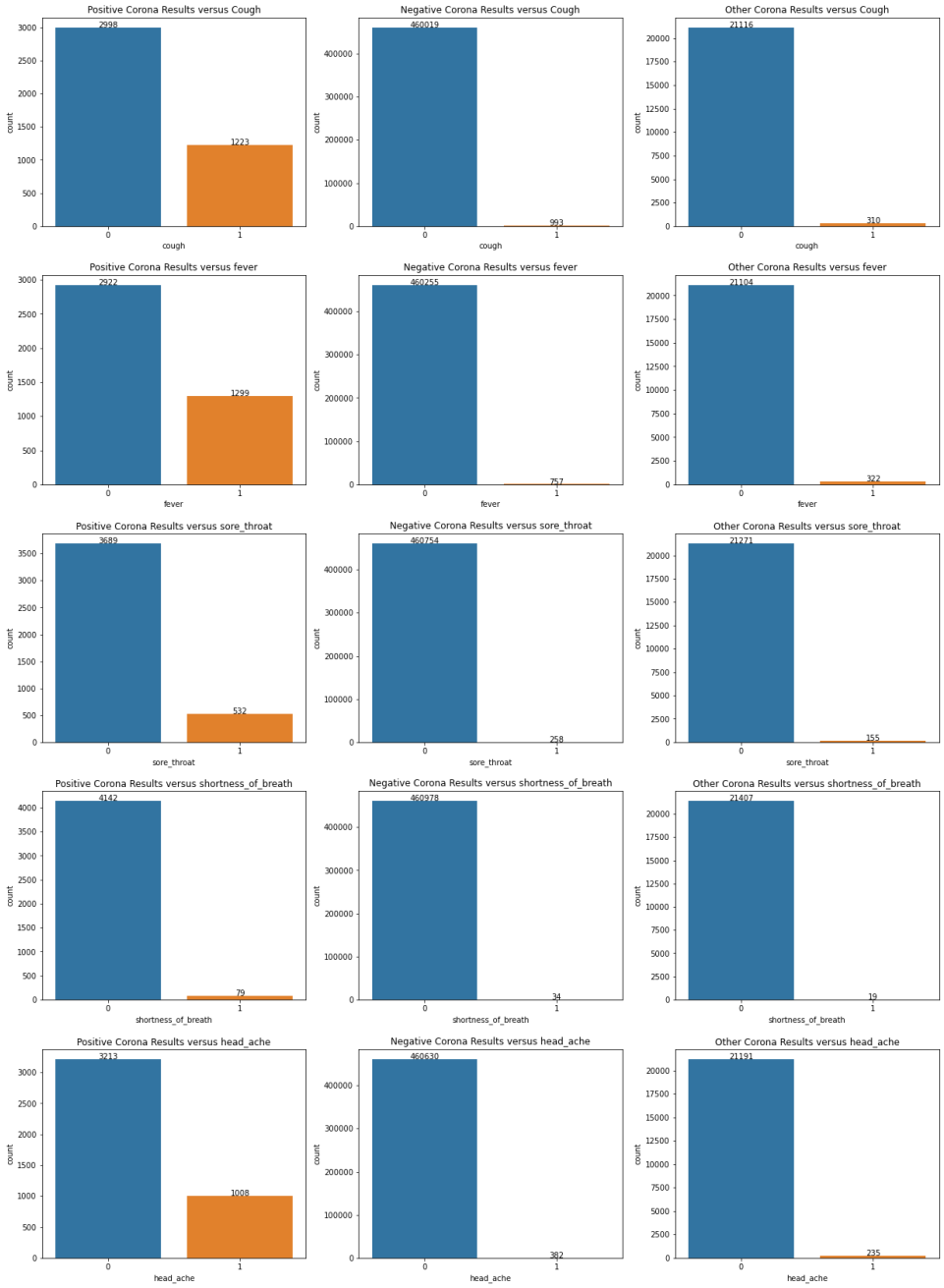
```
date                       0
cough                      0
fever                      0
sore_throat                0
shortness_of_breath        0
head_ache                  0
corona_result              0
age                   285179
gender                215468
test_indication            0
dtype: int64
```



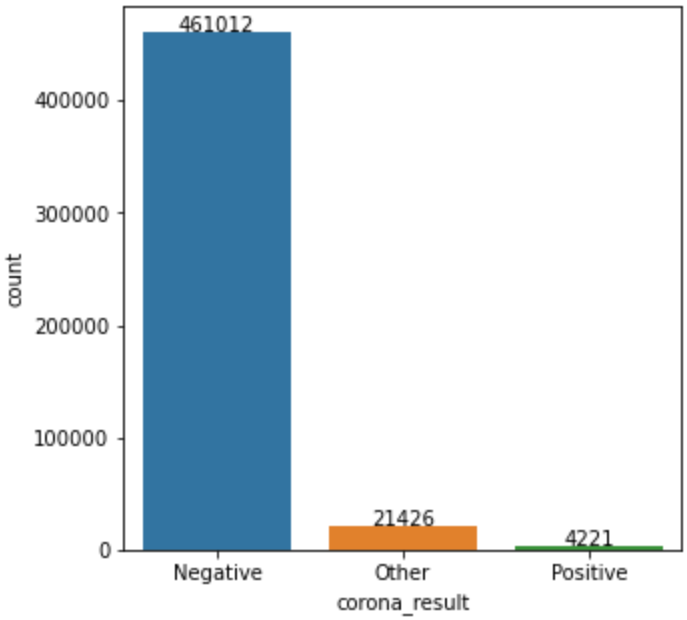
We visualized the distribution of our 5 symptoms below and noticed that they are highly skewed with over 75% being 0.



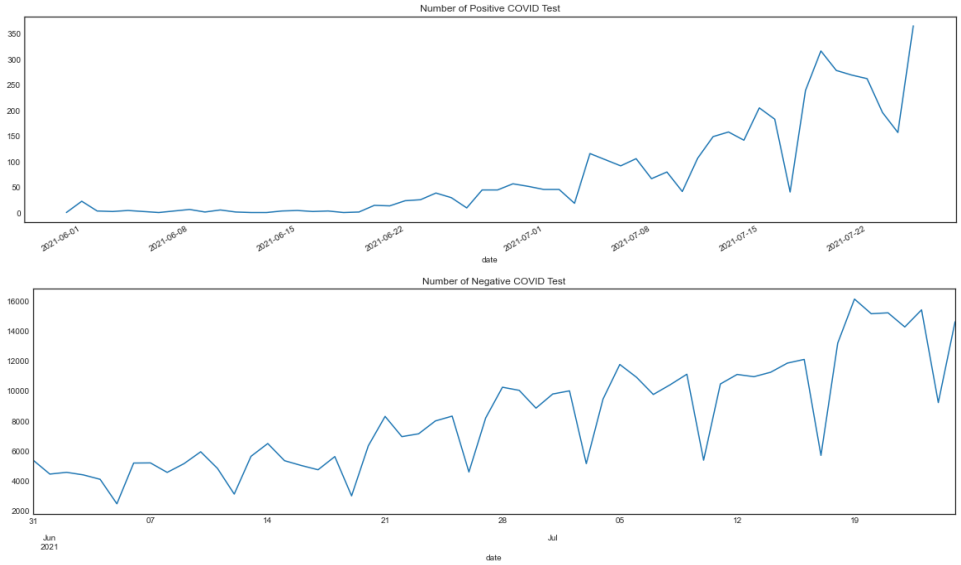
We also broke down the 5 symptoms against corona_result to see if there are any obvious patterns or linkage between the symptoms and the test results. Based on the comparison graphs below, even though there's no obvious direct link between the symptoms and positive covid result, it does show that these 5 symptoms are more likely to be present in the positive result cases than negative.

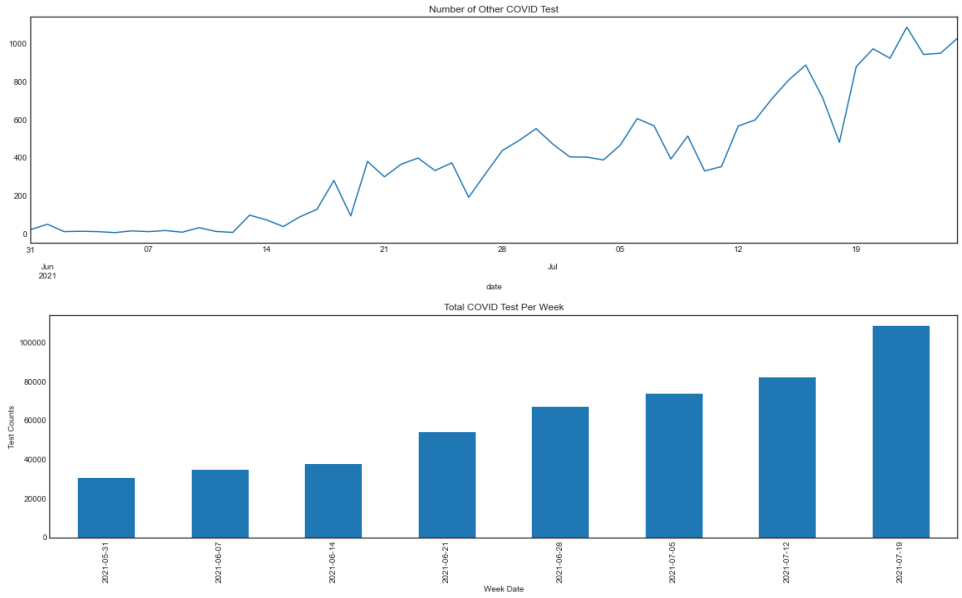


We created the below corona_result count graph to get a better understanding of its distribution. Base on the graph, our dataset is highly imbalanced toward the negative result. In fact, less than 1 percent of our entire dataset is positive.



Lastly, to see if there's a pattern to the number of COVID tests taken over time, we have created the below three graphs that show the daily counts of COVID tests based on the test result. We have also group COVID test counts from daily to weekly bins so we can better determine if there's a pattern in the number of tests taken over time. Overall, there seems to be a general upward trend in the number of people getting covid tests.





5. Data Preprocessing

For the test result marked as “other”, i.e. not tested, in testing, inconclusive, we dropped these rows as they are not meaningful to the prediction. The prediction target variable is expected to be either positive or negative.

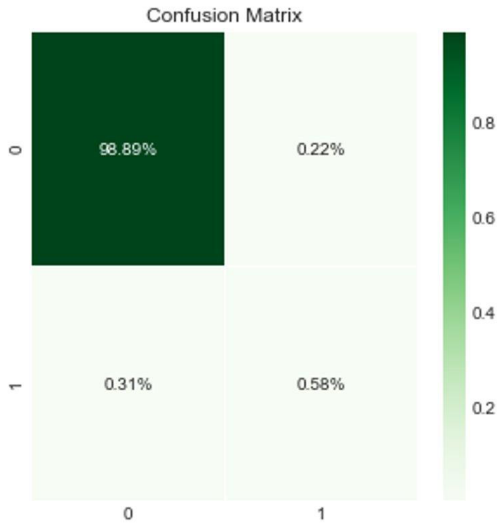
For the two features found with missing values, the missingness of the data is significant and completely at random. As the age and gender are nominal data types, we filled in those empty cells with a string instead of imputing using an average or KNN. At the end of this section, we also compared if dropping those rows with missing values can better improve the performance.

	Total	**Percent**
age	275466	0.592103
gender	208245	0.447614

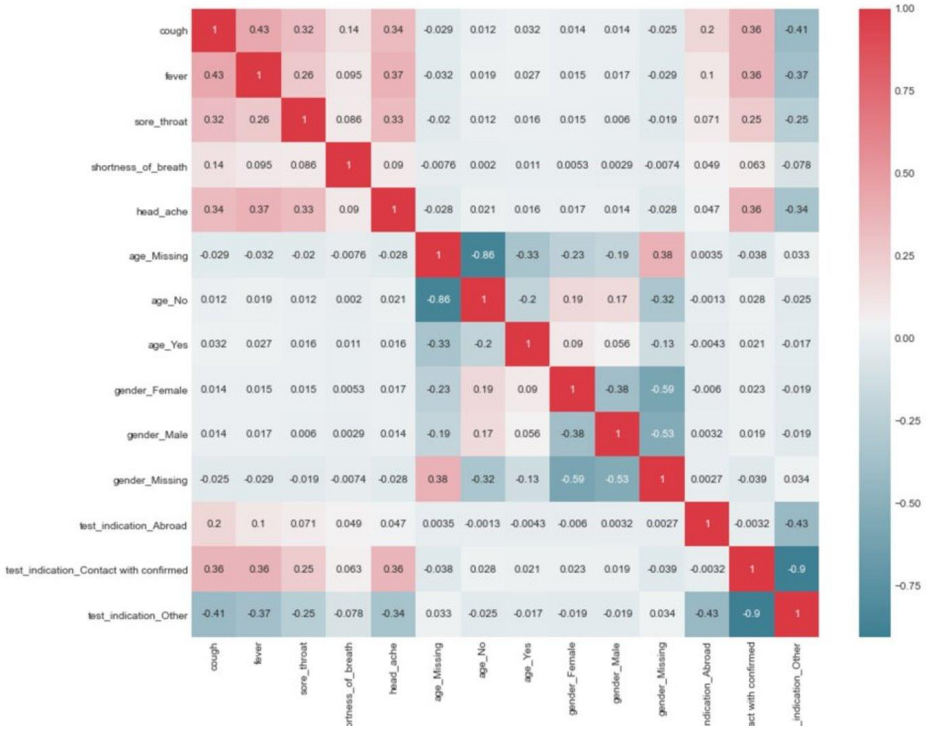
6. Feature Selection

With regard to performance evaluation, next to the true positive rate, the false negative is more important than the false positive as false positive cases can be further confirmed using blood test for example, but a high false negative rate and negligence of false negative cases will lead to delayed treatment of the infected individuals and extending the spread of the infection.
As suggested by the confusion matrix, the true positive rate is significant and it will outweigh the false negative or positive rates if we use accuracy, precision or recall scores;
those scores may not be sensitive to false negative or positive rate. Instead, we used performance metrics tailored for imbalanced data. For example, sensitivity and specificity quantifies the ability to avoid false negatives and false positives. But still these scores do not look very promising; we expect to see more metrics above 90%. To maximize the performance, we resorted to feature selection to eliminate the redundant features.

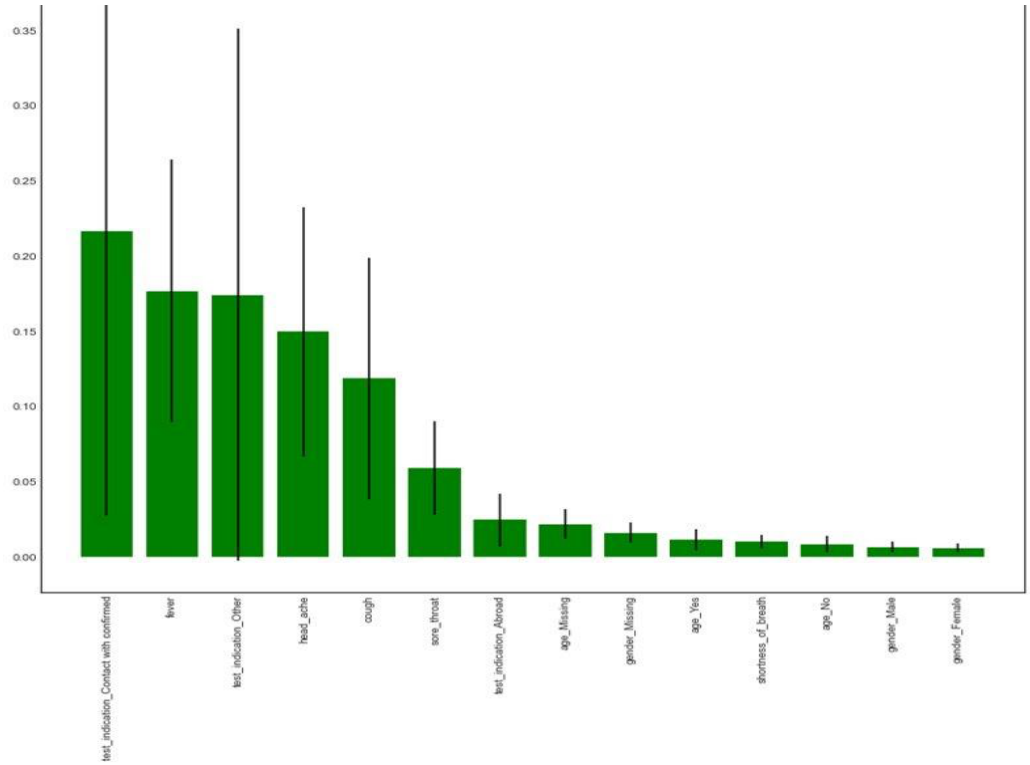
	Method	**Original**
0	Balanced Accuracy	0.824516
1	Sensitivity	0.994691
2	Specificity	0.654340
3	AUROC	0.824516
4	F1	0.684547
5	Geometric Mean	0.806763
6	Macro Average MAE	0.175484

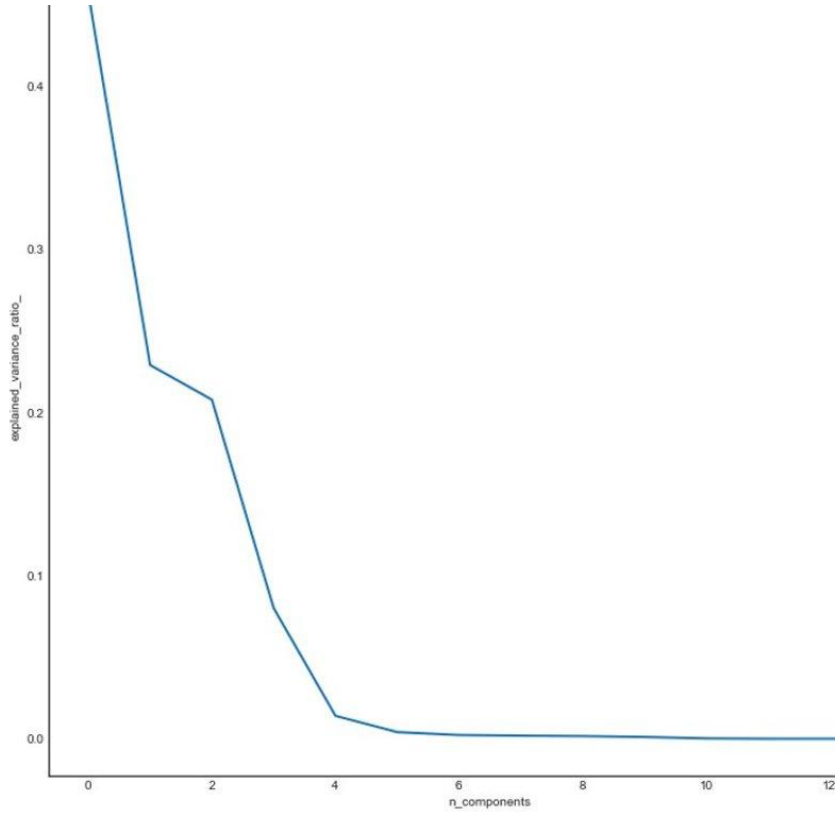


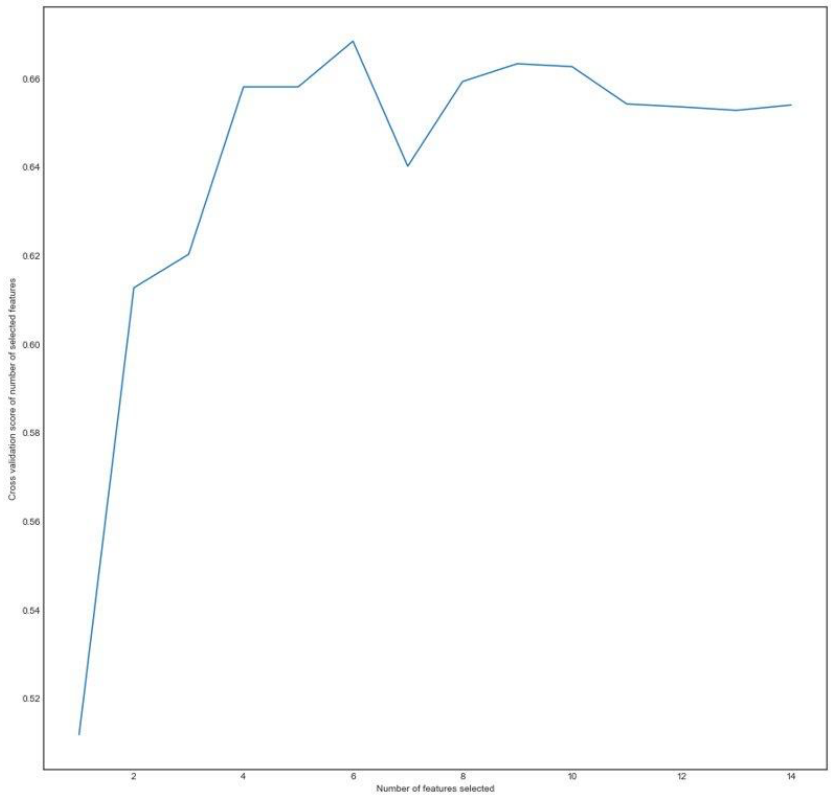
The correlation matrix below reveals the independence of each feature. As indicated by the lightness of the color, the correlations scores are all quite low, the highest being 0.43. This suggests that all features are independent of each other.



We utilized feature importance generated by random forest, PCA and REFCV to find out how many features should be selected. All 3 graphs below show that including the top 6 features will be sufficient for the prediction.




explained_variance_ratio_
0.4
0.3
0.2
0.1
0.0
0
2
4
6
8
10
12
n_components


Cross validation score of number of selected features
0.66
0.64
0.62
0.60
0.58
0.56
0.54
0.52
2
4
6
8
10
12
14
Number of features selected

To find out how many features should be selected, the table below summarizes the performance scores from 4 different methods. SelectKBest-chia2 differs from the other 3 methods on the last feature to be selected: it selects shortness of breath while the other 3 methods select test_indication_other. This is probably due to the fact that SelectKBest-chia2 is correlation-based, so the selection mirrors the correlation matrix, while the other 3 methods are information-gain-based or information gain weighted by the probability of reaching that node in tree-based models, so they share the same selection.

	mutual_info_reg	SelectKBest-chi2	RFE	RF_importance
test_indication_Contact with confirmed	0.016566	97792.214762	True	0.216310
fever	0.011165	70120.384480	True	0.176802
head_ache	0.006089	62286.259488	True	0.149830
cough	0.010973	56200.021071	True	0.118591
sore_throat	0.004071	30008.323534	True	0.059027
shortness_of_breath	0.001829	4462.521708	False	0.010115
age_Yes	0.000325	754.507236	False	0.011694
test_indication_Other	0.017961	710.261205	True	0.174052
gender_Missing	0.001968	661.563719	False	0.016072
age_Missing	0.003338	598.631176	False	0.021852
test_indication_Abroad	0.001024	550.946819	False	0.024571
age_No	0.000000	393.129661	False	0.008519
gender_Male	0.002026	340.249068	False	0.006630
gender_Female	0.000289	209.165600	False	0.005935

In order to compare which selection method delivers a better performance, we compute the performance of datasets with two different selections in comparison to the performance of the original dataset with no feature selected. The dataset choosing test_indication_other as the last feature has higher performance than the dataset choosing shortness of breath. We can confidently finalize the 6 features with the last one being test_indication_other.

Method	Original	SelectedKBest-chi2	All other 3 methods	original vs other 3 %	original vs sel-chi2 %
Balanced Accuracy	0.824516	0.831920	0.855055	3.7%	0.9%
Sensitivity	0.994691	0.993853	0.994358	-0.03%	-0.08%
Specificity	0.654340	0.669988	0.715753	9.39%	2.39%
AUROC	0.824516	0.831920	0.855055	3.7%	0.9%
F1	0.684547	0.657485	0.690995	0.94%	-3.95%
Geometric Mean	0.806763	0.816008	0.843632	4.57%	1.15%
Macro Average MAE	0.175484	0.168080	0.144945	-17.4%	-4.22%

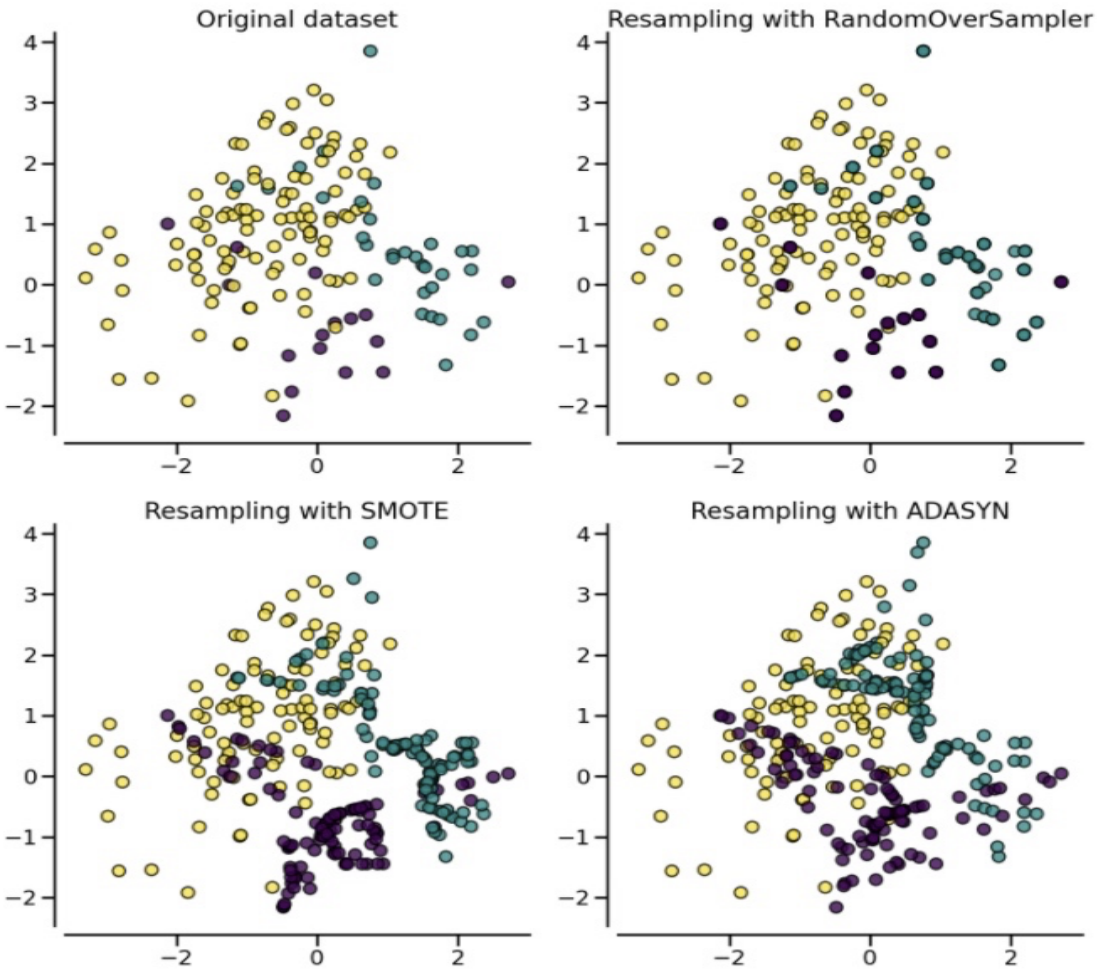
Finally, after selecting the 6 features, we compared whether dropping the rows with null values can improve the performance. It turns out that dropping will impair the performance. Indeed, dropping the rows with null values may drop positive cases in parallel, thus making the data even more imbalanced. Since age and gender features are not selected, it would make no difference either keeping those rows filled or unfilled. We used the filled dataset so that more rows, especially those with positive results, can be kept. We concluded with a dataset with 6 features selected and the missing values filled.

	Method	All other 3 methods_filled	All other 3 methods_dropped	dropped vs filled
0	Balanced Accuracy	0.855055	0.784831	-8.21%
1	Sensitivity	0.994358	0.988189	-0.62%
2	Specificity	0.715753	0.581473	-18.76%
3	AUROC	0.855055	0.784831	-8.21%
4	F1	0.690995	0.637591	-7.73%
5	Geometric Mean	0.843632	0.758027	-10.15%
6	Macro Average MAE	0.144945	0.215169	48.45%

7. Sampling

Due to the highly imbalanced property of our dataset, the prediction results derived from vanilla original dataset may be biased and perform badly from the perspective of predicting the positive instance which is the minority class. To emphasize the effect of positive samples in our training dataset, we decided to leverage several sampling techniques for imbalanced data.

The sampling method we are going to use is from **Imblearn** package. The strategy to fight with imbalanced data is to either increase the number of samples which are under-represented or downsize the scale of the samples which belong to the majority class. The naive oversampling and under-sampling method provide us the ability to randomly duplicate or delete samples from the original dataset. However, the repeating samples could be an issue. Therefore, we also applied the variant method of oversampling method – SMOTE (Synthetic Minority Oversampling Technique) which generates new samples by interpolation. There is also a technique called ADASYN (Adaptive Synthetic) which focuses on generating new samples which are similar to original samples and are wrongly classified by KNN methods. We did not apply this method to our dataset because of the poor features of our dataset and the memory constraints to run ADASYN. The different sampling methods are illustrated by the following charts.



Oversampling: Naive Random Over-Sampling (ROS) to increase the sample size of under-represented data by replication with replacement

93]:
```
from imblearn.over_sampling import RandomOverSampler
def oversample(X, Y):
    ros = RandomOverSampler(random_state=0)
    return ros.fit_resample(X, Y)
```

Oversampling:SMOTE based on KNN and generating new samples

94]:
```
from imblearn.over_sampling import SMOTE
def oversample_smote(X, Y):
    return SMOTE().fit_resample(X, Y)
```

Under-sampling:Random Under-Sampler (RUS) is a fast and easy way to balance the data by randomly selecting a subset of data for the targeted classes

95]:
```
from imblearn.under_sampling import RandomUnderSampler
def undersample(X, Y):
    rus = RandomUnderSampler(random_state=0)
    return rus.fit_resample(X, Y)
```

Different sampling methods are used in our prediction, including original sampling, random over-sampling, SMOTE, random under_sampling. We compared the performance of our models when applying various sampling technologies.

8. Modeling

Several mainstream models were adopted in training and prediction of our dataset such as **Perceptron, Decision Tree, Gaussian Naive Bayes, Random Forest, GradientBoostingForest, XGBoost and Neural Network.**

```
models = {
    "Dummy Classifier" : DummyClassifier(strategy='uniform'), # dummy classifier to predict randomly
    "Most Frequent Clasifier" : DummyClassifier(strategy='most_frequent'), # dummy classifier to always predict the majority
    "Perceptron": Perceptron(alpha = 0.1),
    "Decision Tree (max_depth=None)": DecisionTreeClassifier(max_depth=None, random_state=0),
    "GNB": GaussianNB(),
    "RandomForest": RandomForestClassifier(random_state=0, n_estimators = 101),
    "GradientBoostingForest": GradientBoostingClassifier(random_state=0, n_estimators = 101),
    "XGBoost": XGBClassifier(n_estimators=1000, tree_method='hist', random_state=0, eval_metric='mlogloss')
}
```

The parameters of the above classifiers were tuned through demo tests to ensure that they have the best performance in prediction. For the neural network, we used one hidden layer and sigmoid activation function for the final output.

We introduced two other classifiers: **dummy classifier** which randomly predicts the results to be positive or negative and most frequent classifier which always predicts the samples to be the majority class. We used these two classifiers because we have known that our dataset is highly imbalanced so that the accuracy is less informative though the accuracy is high. Therefore, we used these two dummy classifiers serving as the baseline performance indicators to give us a reference on how well our models perform in the sampling dataset. We trained our models and make predictions using the following the steps below:

1. Using Stratified K-Fold splitting (k = 10) method to split train-test samples
2. For training data, using different sampling methods (over sample, SMOTE, under sample) to resample the training data and keep test data intact.
3. Run model to fit and predict for each model and record the metrics (for training data and test data)
4. Calculate the average metrics for 10 folds.

The code in Jupyter notebook is shown below:

```
n_splits = 10;
skf = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=0)
prediction = dict() # store the metrics
prediction_train = dict() # store the metrics for the trainning set
for train_index, test_index in skf.split(X, Y):

    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = Y[train_index], Y[test_index]
    for classif_name in models.keys():
        classif = models[classif_name]
        classif.fit(X_train, y_train)
        y_pred = classif.predict(X_test) # test
        # threasholding the continuous predicted value to binary labels
        y_pred = thresholding(y_pred, y_test)
        output = evaluate_imbalanced(y_test, y_pred)
        y_pred_train = classif.predict(X_train) # train
        # threasholding the continuous predicted value to binary labels
        y_pred_train = thresholding(y_pred_train, y_train)
        output_train = evaluate_imbalanced(y_train, y_pred_train)
        ## store the metrics in a dict
        if classif_name in prediction:
            for metric in prediction[classif_name].keys():
                prediction[classif_name][metric] += output[metric]
                prediction_train[classif_name][metric] += output_train[metric]
        else:
            prediction[classif_name] = output
            prediction_train[classif_name] = output_train
```

In the case that some classifiers predict continuous values instead of categorical results, we used the **thresholding** method to map continuous floating value to categorical labels and maximize the g-mean value of the prediction.

```
def thresholding(pred, y): # threasholding the continuous predicted value to binary labels
    # calculate roc curves
    fpr, tpr, thresholds = roc_curve(y, pred)
    # use gemans to select the optimal threshold
    gmeans = sqrt(tpr * (1-fpr))
    # locate the index of the largest g-mean
    ix = argmax(gmeans)
    threshold = thresholds[ix]
    for i in range(len(y)):
        pred[i] = 1 if pred[i] >= threshold else 0
    return pred
```

9. Evaluation

The performances from different models were sorted based on their AUROC in descending order.

Original sampling

1. Test data

	Balanced Accuracy	Sensitivity	Specificity	AUROC	F1	Geometric Mean	Macro Average MAE
GNB	0.876196	0.992606	0.759785	0.876196	0.650455	0.868369	0.123804
Neural Network	0.876196	0.992606	0.759785	0.876196	0.650455	0.868369	0.123804
Decision Tree (max_depth=None)	0.846892	0.993848	0.699935	0.846892	0.672924	0.833985	0.153108
RandomForest	0.846892	0.993848	0.699935	0.846892	0.672924	0.833985	0.153108
XGBoost	0.846892	0.993848	0.699935	0.846892	0.672924	0.833985	0.153108
GradientBoostingForest	0.837085	0.993719	0.680451	0.837085	0.661969	0.822250	0.162915
Perceptron	0.624147	0.991983	0.256311	0.624147	0.323432	0.456407	0.375853
Dummy Classifier	0.510758	0.500734	0.520782	0.510758	0.018583	0.510554	0.489242
Most Frequent Clasifier	0.500000	0.990927	0.009073	0.500000	0.000000	0.094819	0.500000

2. Train data

	Balanced Accuracy	Sensitivity	Specificity	AUROC	F1	Geometric Mean	Macro Average MAE
GNB	0.876196	0.992606	0.759785	0.876196	0.650455	0.868369	0.123804
Neural Network	0.876196	0.992606	0.759785	0.876196	0.650455	0.868369	0.123804
Decision Tree (max_depth=None)	0.846892	0.993848	0.699935	0.846892	0.672924	0.833985	0.153108
RandomForest	0.846892	0.993848	0.699935	0.846892	0.672924	0.833985	0.153108
XGBoost	0.846892	0.993848	0.699935	0.846892	0.672924	0.833985	0.153108
GradientBoostingForest	0.837085	0.993719	0.680451	0.837085	0.661969	0.822250	0.162915
Perceptron	0.624147	0.991983	0.256311	0.624147	0.323432	0.456407	0.375853
Dummy Classifier	0.510758	0.500734	0.520782	0.510758	0.018583	0.510554	0.489242
Most Frequent Clasifier	0.500000	0.990927	0.009073	0.500000	0.000000	0.094819	0.500000

The Gaussian Naïve Bayes and Neural Network perform the best with AUROC achieving 0.876 among all models but the differences between models are nuanced. The Perceptron has a substandard performance because it cannot predict positive samples well. The two dummy classifiers perform the worst. Noticeably, the models have the same performance on test data as on train data which means our models are not overfitting on the train data.

Randomly over-sampling

1. Test data

	Balanced Accuracy	Sensitivity	Specificity	AUROC	F1	Geometric Mean	Macro Average MAE
Neural Network	0.876350	0.992911	0.759788	0.876350	0.660111	0.868504	0.123650
GradientBoostingForest	0.876205	0.992625	0.759785	0.876205	0.651057	0.868377	0.123795
GNB	0.876196	0.992606	0.759785	0.876196	0.650455	0.868369	0.123804
XGBoost	0.875883	0.992685	0.759082	0.875883	0.652677	0.868000	0.124117
Decision Tree (max_depth=None)	0.875882	0.992683	0.759082	0.875882	0.652609	0.867999	0.124118
RandomForest	0.875882	0.992683	0.759082	0.875882	0.652610	0.867999	0.124118
Perceptron	0.872702	0.992662	0.752742	0.872702	0.650019	0.864267	0.127298
Dummy Classifier	0.501012	0.500721	0.501303	0.501012	0.017893	0.500857	0.498988
Most Frequent Clasifier	0.500000	0.990927	0.009073	0.500000	0.000000	0.094819	0.500000

2. Train data

	Balanced Accuracy	Sensitivity	Specificity	AUROC	F1	Geometric Mean	Macro Average MAE
Decision Tree (max_depth=None)	0.876370	0.876370	0.876370	0.876370	0.859753	0.876370	0.123630
RandomForest	0.876370	0.876370	0.876370	0.876370	0.859753	0.876370	0.123630
XGBoost	0.876370	0.876370	0.876370	0.876370	0.859753	0.876370	0.123630
Neural Network	0.876343	0.876343	0.876343	0.876343	0.859727	0.876343	0.123657
GradientBoostingForest	0.876333	0.876333	0.876333	0.876333	0.859717	0.876333	0.123667
GNB	0.876326	0.876326	0.876326	0.876326	0.859711	0.876326	0.123674
Perceptron	0.872319	0.872319	0.872319	0.872319	0.854322	0.872319	0.127681
Most Frequent Clasifier	0.500000	0.500000	0.500000	0.500000	0.000000	0.500000	0.500000
Dummy Classifier	0.499952	0.499952	0.499952	0.499952	0.499976	0.499952	0.500048

The performance is similar compared with that of original samples. However, there are two main differences.

1) The performance from Perceptron is aligned with other models
2) Our models may be overfitting on the train data according to the F1 score.

SMOTE oversampling

1. Test data

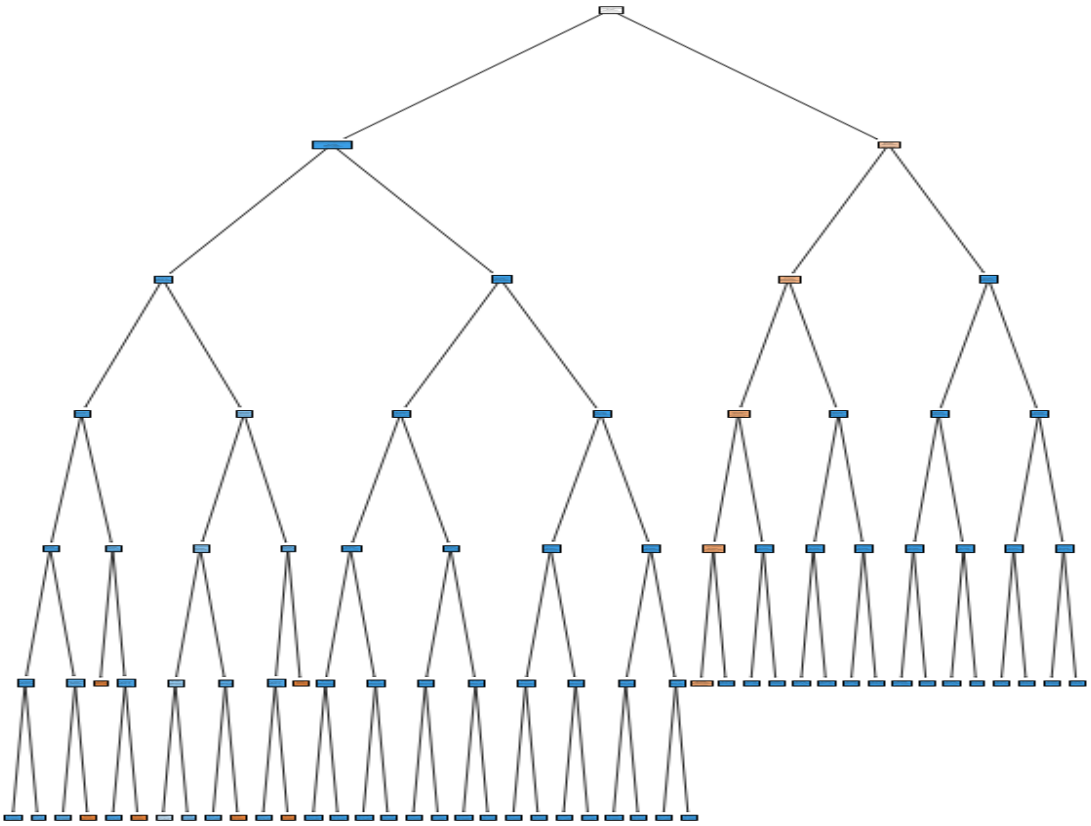
	Balanced Accuracy	Sensitivity	Specificity	AUROC	F1	Geometric Mean	Macro Average MAE
Neural Network	0.876383	0.992978	0.759789	0.876383	0.662245	0.868533	0.123617
GNB	0.876196	0.992606	0.759785	0.876196	0.650455	0.868369	0.123804
GradientBoostingForest	0.875975	0.992634	0.759316	0.875975	0.651184	0.868113	0.124025
Decision Tree (max_depth=None)	0.875883	0.992685	0.759082	0.875883	0.652677	0.868000	0.124117
XGBoost	0.875883	0.992685	0.759082	0.875883	0.652677	0.868000	0.124117
RandomForest	0.875882	0.992683	0.759082	0.875882	0.652610	0.867999	0.124118
Perceptron	0.872382	0.992726	0.752038	0.872382	0.651872	0.863884	0.127618
Dummy Classifier	0.500124	0.500594	0.499653	0.500124	0.017829	0.499965	0.499876
Most Frequent Clasifier	0.500000	0.990927	0.009073	0.500000	0.000000	0.094819	0.500000

2. Train data

	Balanced Accuracy	Sensitivity	Specificity	AUROC	F1	Geometric Mean	Macro Average MAE
Decision Tree (max_depth=None)	0.876366	0.876366	0.876366	0.876366	0.859748	0.876366	0.123634
RandomForest	0.876366	0.876366	0.876366	0.876366	0.859748	0.876366	0.123634
XGBoost	0.876366	0.876366	0.876366	0.876366	0.859748	0.876366	0.123634
Neural Network	0.876343	0.876343	0.876343	0.876343	0.859725	0.876343	0.123657
GradientBoostingForest	0.876329	0.876329	0.876329	0.876329	0.859708	0.876329	0.123671
GNB	0.876323	0.876323	0.876323	0.876323	0.859706	0.876323	0.123677
Perceptron	0.872208	0.872208	0.872208	0.872208	0.854205	0.872208	0.127792
Most Frequent Clasifier	0.500000	0.500000	0.500000	0.500000	0.000000	0.500000	0.500000
Dummy Classifier	0.499942	0.499942	0.499942	0.499942	0.499742	0.499942	0.500058

We achieved similar results with over-sampling using SMOTE. The Neural Network turns out to be the best and we are not overfitting on train data.

Below is the decision tree graph generated from model fit with train data oversampled using SMOTE. We can have a basic understanding on how the decision tree predicts the results. The detail is not very clear due to too many cases in the leaf layer. However, the depth of the tree is only 7 layers.



Randomly under-sampling

1. Test data

	Balanced Accuracy	Sensitivity	Specificity	AUROC	F1	Geometric Mean	Macro Average MAE
Neural Network	0.876386	0.992984	0.759789	0.876386	0.662435	0.868536	0.123614
XGBoost	0.876203	0.992621	0.759785	0.876203	0.650904	0.868375	0.123797
GNB	0.876196	0.992606	0.759785	0.876196	0.650455	0.868369	0.123804
RandomForest	0.876106	0.992662	0.759551	0.876106	0.652138	0.868259	0.123894
GradientBoostingForest	0.875974	0.992632	0.759316	0.875974	0.651100	0.868112	0.124026
Decision Tree (max_depth=None)	0.875882	0.992683	0.759082	0.875882	0.652663	0.868000	0.124118
Perceptron	0.796337	0.992320	0.600355	0.796337	0.518617	0.708035	0.203663
Dummy Classifier	0.508344	0.501542	0.515146	0.508344	0.018413	0.508109	0.491656
Most Frequent Clasifier	0.500000	0.990927	0.009073	0.500000	0.000000	0.094819	0.500000

2. Train data

	Balanced Accuracy	Sensitivity	Specificity	AUROC	F1	Geometric Mean	Macro Average MAE
Decision Tree (max_depth=None)	0.876412	0.876412	0.876412	0.876412	0.859715	0.876412	0.123588
RandomForest	0.876412	0.876412	0.876412	0.876412	0.859728	0.876412	0.123588
XGBoost	0.876398	0.876398	0.876398	0.876398	0.859732	0.876398	0.123602
GradientBoostingForest	0.876385	0.876385	0.876385	0.876385	0.859715	0.876385	0.123615
GNB	0.876359	0.876359	0.876359	0.876359	0.859702	0.876359	0.123641
Neural Network	0.876359	0.876359	0.876359	0.876359	0.859702	0.876359	0.123641
Perceptron	0.795535	0.795535	0.795535	0.795535	0.680207	0.795535	0.204465
Most Frequent Clasifier	0.500000	0.500000	0.500000	0.500000	0.000000	0.500000	0.500000
Dummy Classifier	0.499539	0.499539	0.499539	0.499539	0.498978	0.499539	0.500461

The performance of different models is almost the same except that the Perceptron is slightly worse than the other models.

10. Conclusion

Sampling Method	Best Models	AUROC
Original sample	GNB, Neural Network	87.62
Randomly over sample	Neural Network	87.64
SMOTE	Neural Network	87.64
Randomly under sample	Neural Network	87.64

The table above summarizes the best performing models from each sampling method. The Neural Network performs best in all four sampling methods. However, the difference in performances between different models is not significant and the performance on the original dataset is not very high which means the models may be underfitting due to the nature of this dataset.

We believe the possible reasons we were unable to achieve ever higher performance are:

1. The hidden pattern behind the COVID-19 data is continuously changing over time due to virus mutation and that is why we might not be able to obtain a good prediction based on past data.
2. The features may not be well labeled and subject to bias. For example, some features are binary but it may be subjective to categorize those symptoms. Besides, the Israeli data document comments that the five symptoms labeled are merely the most representative ones but there are more symptoms to indicate a positive infection. We utilized 6 binary features which produce 64 combinations of features. For each combination, the results of diagnosis are not unary which makes these features not well-suited for predicting diagnosis results.

Nonetheless, we believe our Neural Network model can be utilized to help efficiently identify COVID-19 patients as a preliminary filter and thereby, help reduce the stress on our medical systems and the spread of the virus.

11. Reference

- https://en.wikipedia.org/wiki/COVID-19
- https://www.cnn.com/interactive/2020/health/coronavirus-maps-and-cases/
- https://www.cdc.gov/coronavirus/2019-ncov/variants/delta-variant.html
- https://www.nature.com/articles/s41746-020-00372-6#sec6
- https://github.com/guojunseven/Data-Mining-Project/blob/master/COVID19%20Diagnosis%20Prediction.ipynb
- https://www.kaggle.com/kanncaa1/feature-selection-and-data-visualization
- https://www.kaggle.com/prashant111/comprehensive-guide-on-feature-selection
- https://imbalanced-learn.org/stable/references/metrics.html#module-imblearn.metrics
- https://en.wikipedia.org/wiki/Template:COVID-19_pandemic_data
- https://www.nytimes.com/interactive/2021/us/covid-cases.html
- https://ourworldindata.org/coronavirus-data
- https://github.com/CSSEGISandData/COVID-19
- https://www.cdc.gov/coronavirus/2019-ncov/variants/delta-variant.html
- https://imbalanced-learn.org/stable/over_sampling.html#naive-random-over-sampling
- https://data.gov.il/dataset/covid-19