Predicting Covid-19 Diagnosis from Clinical Data

Runsheng Wang, Jinqi Lu[1]

April. 29, 2021

Abstract

This study seeks to use machine learning to predict Covid-19 diagnosis from a multitude of clinical data. The data was collected, aggregated, de-identified, and published by Carbon Health, a U.S. primary and urgent care provider. The dataset contains a total of 46 variables, including epidemiological factors, comorbidities, vitals, clinical assessed and patient reported symptoms, as well as lab results and radiological findings. The dataset was first cleaned and encoded, before splitted into a training set and a testing set. Exploratory Data Analysis was performed on the training set to obtain basic understandings about the characteristics of input variables. Feature selection was then performed by ensembling the Chi-Squared and Mutual Information criteria. Six features, namely Cough, Fever, Headache, Loss of Smell, Loss of Taste, and Muscle Sore, were ultimately used for model construction. Due to the extreme imbalance (99:1 ratio) in the class distribution of the response variable, resampling methods were applied to the training set in order to reduce classification bias towards the majority class. Since the dataset consists solely of categorical features, SMOTE-N, a categorical variation of the Synthetic Minority Oversampling Technique (SMOTE), was adapted to the training set to achieve a negative-to-positive class ratio of 5:1. Undersampling was then performed using One Sided Selection and Neighborhood Cleaning Rule to further balance the class distributions of the response variable. To identify the best model for this particular dataset, six classifiers, namely kNN, Logistic, Decision Tree, Random Forest, Categorical Naive Bayes, XGBoost, were fitted and baseline performances were compared. To optimize the hyperparameter tuning process, Random Search CV was executed 2000 times on the Random Forest Classifier, with 4-fold CV repeated three times on each iteration. Best hyperparameters were recorded locally, and Grid Search CV was applied to further tune the hyperparameters by checking the immediate neighbors of each hyperparameter value. The model predicts the testing set quite well, and model performance is substantially better compared to the baseline. However, the model underperforms at predicting positive results, which is to be expected considering the severe class imbalance present in the original dataset. Our model could be applied to prioritize testing when testing resources are scarce or inaccessible. It could also aid the diagnosis of Covid-19 in ambiguous or conflicting cases.

1 Introduction

As Covid-19 continues to ravage countries across the globe, testing has become an indispensable tool for identifying and containing the spread of this highly contagious virus. Test results can lead to early prevention of further spread as well as early intervention and treatment for patients experiencing non-severe symptoms, resulting in less mortality and faster recovery. However, test results are often inaccurate, especially with molecular tests whose "false-negative rate was 20% when testing was performed five days after symptoms began, and much higher (up to 100%) earlier in infection" (Shmerling 2021). Given that the infection of Covid-19 comes with clinical symptoms, we hypothesized that certain symptoms are more indicative of Covid-19 test results than others (CDC 2021). We will attempt to identify features of symptoms that exert significant influences on test results, and construct a machine learning model that predicts Covid-19 diagnosis based on clinical data. Our model could be used to prioritize Covid-19 testing when testing resources/kits are limited, or when testing is inaccessible. We do not anticipate the model to replace testing, but it could be used in conjunction with test results when results are ambiguous, or when results disagree. Furthermore, by highlighting important symptoms, we hope to enable individuals to self-screen and determine whether they should consider getting tested or not.

[1] R.W., J.L. contributed equally to this work.
R.W., J.L. performed wrangling, modelling, prediction, and wrote the paper.

2 Materials and Methods

2.1 Data Collection

As a result of the Health Insurance Portability and Accountability Act of 1996 (HIPAA) Privacy Rule, we faced significant difficulties in obtaining our desired datasets. To protect patients from identity theft and fraud, HIPAA was established to hold the healthcare industry accountable for how they manage and protect the sensitive private information of their patients (HIPAA Journal 2017). Our desired dataset would contain not only patient reported symptoms on Covid-19, but also patient comorbidities and vitals, among other factors. These features are classified as highly sensitive private medical data, and are therefore rarely available on public domains, whether they have been de-identified or not. After some initial research, we found a large comprehensive patient clinical dataset actively maintained by the National Covid Cohort Collaborative (N3C, n.d.). We applied for access to this database, and our request was approved. However, accessing any actual datasets requires submitting another detailed project proposal for review. Since waiting for approval would further delay our progress, we opted to search for de-identified datasets elsewhere.

Our extensive search led us to two datasets with our desired information. The first dataset was sourced from the nationwide clinical data publicly reported by the Israeli Ministry of Health (Nshomron 2021). However, the original dataset has been removed from the official website, and the accessible data is a preprocessed version containing only eight input features. The second dataset was sourced from Carbon Health, a U.S. primary and urgent care provider (Carbon Health 2020). This dataset contains de-identified clinical data on 93995 patients who received Covid-19 testing at Carbon Health, aggregated over the course of six months. This dataset includes 46 features sorted into nine major categories (See Appendix A for Data Availability and Data Dictionary). Since the difference in feature space is large, merging the two datasets, or utilizing one as test data for the other, would result in significant loss of useful information. Therefore, only the Carbon Health dataset is considered for the purpose of this study. The data collection process highlights not only the difficulties in obtaining restricted data, but also the importance of maintaining data privacy and protecting anonymity. As described by Narayanan and Shmatikov, it is possible to de-anonymize personal data even when proper de-identification protocols are applied (Narayanan and Shmatikov 2008). In our future studies, we will make sure that we apply to restricted databases as early as possible, and that we strictly follow the data usage guidelines when working with sensitive dataset.

2.2 Data Concatenation and NA Processing

The raw dataset was given as 29 separate CSV files, each representing data for a particular week beginning from Apr. 12, 2020 and ending at Oct. 20, 2020. The files were first concatenated into a larger CSV and saved locally.

There are 46 features in the dataset, but several features contain large amounts of NA values. This is to be expected, partially as a direct result of the de-identification process which sets data items to null if the data item could potentially be used for re-identification. Features with over 40% of its values as NAs are excluded from model building, as it is simply impossible to impute such large amounts of missing values, and imputing would necessarily result in bias. Several imputation methods were considered for features with less NAs. However, none were eventually adopted, due to the categorical nature of this dataset. Categorical variables are often imputed with the most frequent class (mode-imputing) or through a classifier (e.g., kNN). Since all categorical features are binary in this dataset, mode-imputing would lead to significant bias towards the most frequent class. kNN imputation is also unsuitable; not only is the distance metric ambiguous for categorical features, but also due to the high dimensionality of the feature space, resulting in Curse of Dimensionality for all distance-based metrics.

Classifiers such as XGBoost have built-in methods to deal with NA values, where a default direction is assigned to each decision node and missing values are directed to the default direction. However, assigning the default direction to each node correctly is an extremely complex problem, and improved results are often not guaranteed. To avoid introducing any bias, we decided to discard rows with any NA values after we excluded features with over 40% NAs.

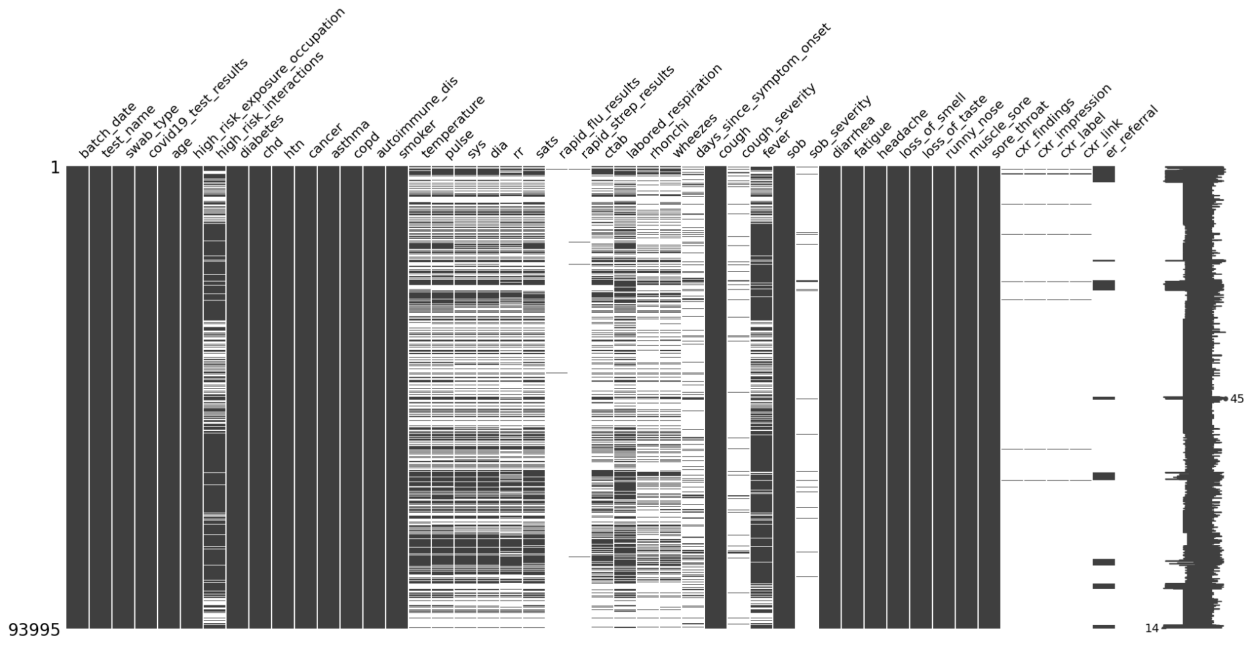


Figure 1. NA Distribution

In Figure 1, the distributions of NA values in each feature were plotted using the Missingno package. Each line white line represents a single missing entry in a particular column, and each black line represents a non-NA entry. After excluding features with over 40% NAs and other irrelevant features (e.g., batch_date, test_name, etc.), 22 features remain in the dataset. They are:

- Covid19_test_results, age, high_risk_exposure_occupation, diabetes, chd, htn, cancer, asthma,
- copd, autoimmune_dis, smoker, cough, fever, sob, diarrhea, fatigue, headache, loss_of_smell,
- loss_of_taste, runny_nose, muscle_sore, sore_throat

The original dataset has 93995 observations. After dropping all NAs row-wise, 71035 samples remain. The dataset is still considerably large, providing sufficient samples for machine learning algorithms to draw conclusions.

We considered splitting the data into training and testing prior to handling NAs, as unseen data could technically include NA values. We decided to split after dealing with NAs, primarily because we anticipate to apply a classifier that only works on complete datasets. Moreover, very few classifiers handle NA values as input without using some form of imputation. We have justified that imputation is not appropriate for this particular dataset. Therefore, we will be using a classifier without NA capabilities.

2.3 Encoding

The categorical features in the raw dataset are encoded as binary string variables or boolean True/False. Most machine learning algorithms only accept numeric types as input and output; model runtime and model performance can also vary based on how the input is encoded. Thus, encoding categoricals to numerics are necessary prior to any modeling.

Since the raw dataset contains missing values, the “read_csv” function from pandas uses type “object” to represent both booleans and string binaries in order to account for the existence of NAs. To ensure proper encoding, categorical features are first type-casted to strings and lower-cased. The Label Encoder and One Hot Encoder from sklearn.preprocessing were considered, but neither were eventually used due to their slow runtime. Encoding after cleaning NA values also helps to avoid errors, as most prebuilt encoders do not handle missing values. We designed a custom encoder using pandas’ vectorized string replace function. Since all categoricals have binary classes, One-Hot-Encoding is not necessary. The problem of one class being further away or closer to another class in a multi-class (3 or more) classification is not present in this scenario. We proceeded to encode “False” as 0 and “True” as 1, while encoding “Negative” as 0 and “Positive” as 1.

The raw dataset contains one continuous variable “Age”. Several classifiers (e.g., Naive Bayes, Random Forest) do not directly operate on continuous values. For instance, Naive Bayes turns continuous features into quartiles and encodes it from 1 to 4. Random Forest, on the other hand, determines a threshold for splitting the continuous feature on each node. Both methods are similar, if not equivalent, to binning continuous values into discrete categories. Numerous prior Covid-19 research have indicated that seniors are at greater risk of contracting Covid-19 and experiencing severe illness (CDC 2021). Using this domain knowledge and the knowledge about classifier designs, we decided to engineer “Age” into a new categorical feature with entry = 1 if the patient is above 60 years old and 0 otherwise.

The order in which encoding and train-test-split are performed is not of major concern because the test set would have to be encoded as well upon acquiring. In scenarios where both the validation and testing set are to be accessed later in the study, one can always apply the same preprocessing pipeline to the unseen sets. The sklearn.preprocessing package offers encoders that “memorize” the encoding schema, which can be fitted later. The vectorized replace can perform the same thing, with the added benefit of handling missing values when necessary.

2.4 Train-Test Split

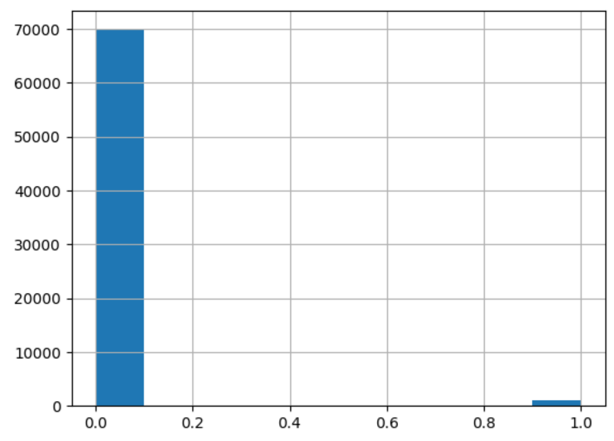


Figure 2. Class Distribution of Covid-19 Test Resultin Processed Dataset

To enable testing on unseen data and prevent the model from overfitting, train-test split was applied to the processed dataset. Random sampling (shuffle-then-select) and stratified sampling were both considered. Stratified sampling was ultimately used due to the extremely severe class imbalance (99:1) of the target variable, as shown in Figure 2. Since the number of positive samples are limited, it is very likely for random sampling to distribute the already limited samples unevenly among the training and testing set. If the majority of the positive samples are assigned to the training set, a classifier would overfit the training data and there would not be enough positive samples in the testing set to validate performance. If the majority of the positive samples are assigned to the testing set, a classifier might fail to extrapolate patterns and relationships that separates a positive sample from a negative sample. Furthermore, classification algorithms are too easily biased by the over or under representation of classes; stratified sampling most closely resembles the class distribution of the raw data, allowing further resampling methods to be applied prior to model-building. Since the dataset is large, the train-test split ratio is less influential on model results. We used a 80:20 ratio, commonly used in data science research and partially justified by the Pareto Principle. We did not use a three-way split as we do not want to impact training performance by further reducing the scarce positive samples in the training set. Instead, we will be using cross validation to perform hyperparameter tuning.

2.5 Feature Selection

With 21 input features, the feature space of the processed dataset exhibits high dimensionality. Not only is high-dimensional feature space more prone to overfitting, one extra dimension requires an exponential increase in the volume of data to produce statistically meaningful results. All L2-norm based methods suffer as a direct result of the Curse of Dimensionality, in that object appears to be sparse and dissimilar in many ways in higher dimensional space, or that tiny pairwise differences in higher dimensions result in huge differences in similarity/dissimilarity. Including more features does not necessarily improve model performance, and doing so might introduce noise and hamper the predictive power of both distance and non-distance based models. Given that the health conditions of individuals vary greatly and certain symptoms could be more indicative of Covid-19 results than others, it might be useful to select features that offer greater predictive power than others. Although some classifiers have built-in methods that control feature selection, such as the "max_features" parameter that exists in multiple sklearn classifiers, eliminating less relevant features prior to modeling would allow the algorithm to fine tune the preselected set of features and further improve performance while avoiding overfitting.

In order to make a statistically justified feature selection and obtain the best model possible with optimized performance and lowest possible computational cost, an ensemble of two feature selection methods were applied to the processed dataset. We originally considered PCA as the dimensionality reduction method, but whether it is appropriate to apply PCA to binary data remains a statistically controversial topic with limited research available. One argument against PCA is that the variance minimization object breaks down with binary variables, and the concept of computing the mean of a categorical variable might not be easily interpretable. Some also argue that PCA performed on a binary dataset is the same as performing a Multiple Correspondence Analysis, but justifications for this statement remain unclear.

We did not bother with PCA as there are other methods suited for categorical features, namely the Chi-Squared statistics and the Mutual Information criterion.

$$\chi^2 = \sum_{i=1}^{R} \sum_{j=1}^{C} \frac{(o_{ij} - e_{ij})^2}{e_{ij}}$$

Figure 3. Chi-Squared Statistics

The Chi-Squared statistics can be used to test whether two categorical variables are correlated or not based on the contingency table. The equation in Figure 3 computes the Chi-Squared statistics. The term o_{ij} correspond to the observed cell count in the i^{th} row and j^{th} column of the contingency table, and the term e_{ij} correspond to the expected cell count in the i^{th} row and j^{th} column of the contingency table, which can be computed by

$$e_{ij} = \frac{\text{row } i \text{ total} * \text{col } j \text{ total}}{\text{grand total}}$$

Figure 4. Chi-Squared Statistics

The capital R and C in the Chi-Squared statistics correspond to the number of rows and number of columns of the contingency table, respectively. Each individual term in the double summation can be interpreted as: how much different is the observed frequency of a specific pair of factor levels different from the expected frequency of this pair, if there is no relationship at all between this pair. The double summation adds up this difference for each possible pair of factor levels between the two categorical variables of interest. If the observed frequency of a pair is exactly the same as its expected frequency, it would match the null hypothesis that the pair is uncorrelated. Mathematically, observed frequency being close to the expected frequency results in the $o_{ij} - e_{ij}$ term approaching 0, hence contributing a small amount to the overall summation. If all pairs have their observed frequency close to their expected frequency, then the total value of the summation would be small. Therefore, smaller Chi-Squared statistics is an indication of weak correlation between two variables, and feature selection can be performed by ranking the larger Chi-Squared values.

We first constructed a function that uses sklearn's SelectKBest to perform Chi-Squared Test of Independence on all input features in the training set with respect to Covid-19 test results. The function returns an array of scores, or values of the test statistics corresponding to each feature. This output array is used as the input for another function, which produced the plot below showing the Chi-Squared statistics of each feature

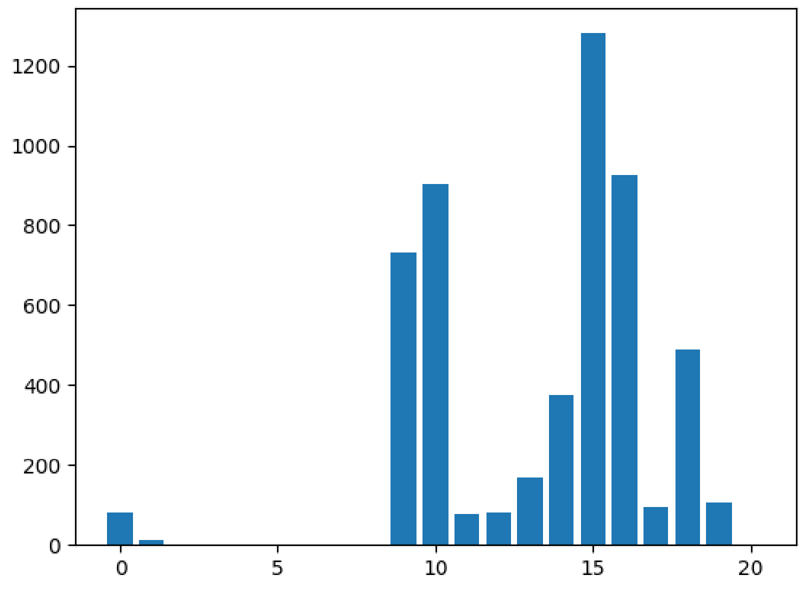


Figure 5. Chi-Squared Statistics for Input Features

As shown in Figure 5, features 9, 10, 14, 15, 16, and 18 have significantly higher Chi-Squared statistics compared to other features, and the difference reaches up to ten orders of magnitude. These features are

- Feature 9: Cough
- Feature 10: Fever
- Feature 14: Headache
- Feature 15: Loss of Smell
- Feature 16: Loss of Taste
- Feature 18: Muscle Sore

It is worth noting that SelectKBest does not use p-values when selecting features, according to its source code. However, high t-values intrinsically mean low p-values and greater confidence in rejecting the null hypothesis. Therefore, we used SelectKBest by specifying k to be all the features, and ranking the Chi-Squared statistics afterwards.

To further validate this finding, we decided to use a different selection criterion in conjunction with the Chi-Squared statistics.

In information theory, Mutual Information (MI) is a commonly used measure for feature independence. Specifically, it measures how much knowing one variable reduces the uncertainty about the other. In the case of two jointly discrete random variables, it can be computed as follows

$$\mathrm{I}(X;Y) = \sum_{y \in \mathcal{Y}} \sum_{x \in \mathcal{X}} p_{(X,Y)}(x,y) \log \left(\frac{p_{(X,Y)}(x,y)}{p_X(x)\, p_Y(y)} \right)$$

Figure 6. Mutual Information Criterion for Jointly Discrete R.V. X and Y

The term $p_{(X,Y)}$is the joint PMF of $X$and Y; $p_X$and $p_Y$are the marginal PMF of $X$and Y, respectively. From the Figure 6, it can be seen that MI is a measure of the inherent dependence expressed in the joint distribution of $X$and$Y$relative to the marginal distribution of X and Yunder the assumption that they are independent. When $X$and$Y$are independent, the term $p_{(X,Y)}(X, Y) = p_X \cdot p_Y$, which results in

$$\log \left(\frac{p_{(X,Y)}(x,y)}{p_X(x)\, p_Y(y)} \right) = \log 1 = 0$$

Figure 7. Mutual Information Criterion when X and Y are Independent

On the other end of the spectrum, if X determines the value of Y, or that by knowing Xwe automatically know Y, then all information of $Y$is contained in X, and the MI value would be high. Therefore, smaller MI values is an indication of weak correlation between two variables, and feature selection can be performed by ranking the larger MI values, similar to how we used Chi-Squared.

According to sklearn's documentation, the sklearn implementation of SelectKBest using MI relies on nonparametric methods based on entropy estimation from k-nearest neighbors distances. Therefore, it would produce slightly different results each time, and the ranking of the weakly correlated features can vary a lot at each iteration. Unfortunately, SelectBestK does not allow the use of random seed for MI feature selection. We have devised an alternative method to not only improve feature selection confidence but also allow for reproducibility of results.

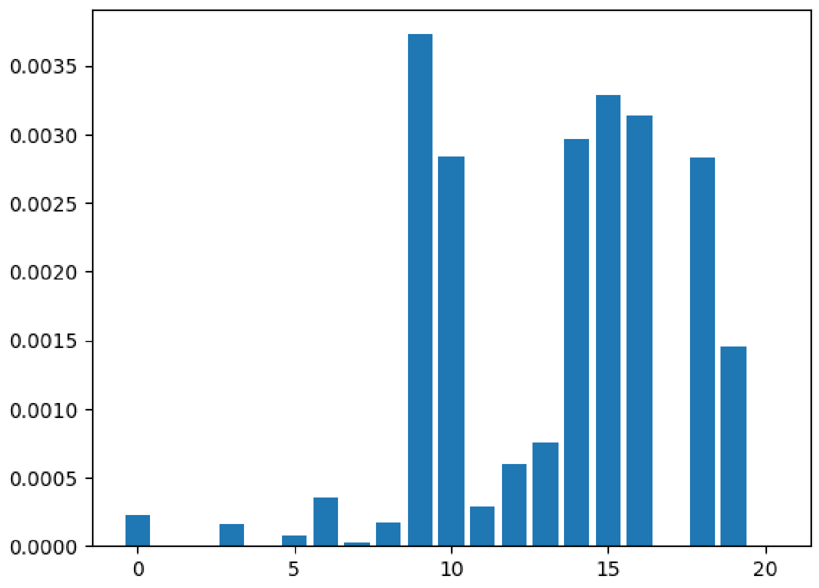


Figure 8. Mutual Information Criterion for Input Features

Figure 8 illustrates one random iteration of the MI feature selection process. Similar to how the Chi-Squared output was constructed, we first computed MI for each feature, and plotted Figure 8 via a separate function. The resultant MI values are in close agreement with the output from Chi-Squared. However, the MI method appears to be more sensitive to weakly correlated features. To combine information from both criteria, we developed an ensemble method that takes into account both the input from Chi-Squared and MI.

We first created a union of all features with a Chi-Squared statistics greater than 1 and a MI value greater than 0.0001 (call this set S). Due to the stochastic nature of the implementation of the MI method, we performed MI feature selection 10 additional times. On each iteration, we ranked all features with MI greater than 0.0001 (17 of them) by their MI value and selected only the top n features to be intersected with set S. The intersection set then replaces the old S set before the start of the next iteration. The value n is also decremented by 1 on each iteration, starting at 17 and ending at 8. This repeated feature selection method removes uncertainty associated with entropy estimation in the implementation of SelectKBest.

This method also confirms that features 9, 10, 14, 15, 16, and 18 are indeed the significant features that should be considered in modeling. There could be correlations between these selected features, as certain symptoms are often concurring. However, correlations between input features could actually be indications of particular patterns. For instance, a patient experiencing cough, fever, headache, loss of smell, and muscle sore all at once could be a strong indication for Covid-19 infection. On the other hand, if a patient only experiences headache without cough and fever, he/she might be less likely to be diagnosed as positive due to critical symptoms not present/concurring. For this reason, we are more concerned about obtained correlated features, rather than obtaining independent features.

To further confirm the validity of these findings, we cross referenced the common Covid-19 symptoms published on CDC's website as well as other research findings (CDC 2021). It appears that the features selected are top symptoms in multiple studies, which justifies the correctness of feature selection from the domain knowledge perspective. Since our repeated feature selection takes a while to run, we hardcoded the significant features to save time and avoid repeating previous work. It is also worth noting that feature selection helps with distance-based undersampling method, as dimensionality reduction makes these algorithms less prone to the Curse of Dimensionality.

2.6 Resampling

Before proceeding with model construction, a critical problem needs to be addressed. As shown in Figure 2, the target variable exhibits a severe class imbalance (99:1). Any attempt to directly classify this dataset would result in the minority class being swamped by the overwhelming number of samples in the majority class. Training a model on such an imbalanced dataset might yield a great accuracy score, but high accuracy is a direct result of the severe imbalance and fails to capture misclassifications in the minority class. For our specific dataset with a class ratio of 99:1 for the target variable, an algorithm that simply classifies all results as negative would have an accuracy of 99%. Moreover, the class imbalance in this particular set is so severe that any recognizable patterns in the positive samples might not be properly captured by the classifier. Therefore, close inspections and adjustments are needed in order to achieve our desired outcome.

The best approach to solve this problem is to collect more data, but the significant difficulties in data collection have been discussed earlier in the report. Another great alternative is to apply resampling methods. Originally, a combination of random oversampling and random undersampling were considered, which consists of randomly duplicating samples from the minority class and randomly deleting samples from the majority class. However, random resampling introduces several serious problems. Specifically, random oversampling can result in model overfit, as duplicate samples increase the classifier's bias towards the input features that make up these samples. Random undersampling runs the risk of losing invaluable information that contributes to classification, especially on boundary conditions. We decided to apply more rigorous methods in order to improve resampling outcomes.

For oversampling, we decided to use SMOTE-N, a categorical variation of the Synthetic Minority Oversampling Technique (SMOTE). Traditional SMOTE is a synthetic data generation technique which works by connecting minority samples that are close in the feature space via a hyperplane and drawing a new sample at a random location on the hyperplane. More specifically, the process can be described as follows

1. A random sample for the minority class is selected, call it x_i
2. The k nearest neighbors of x_i that belongs to the minority class is identified
3. One of the k neighbors is selected at random, call it x_{zi}, and a hyperplane is constructed by connecting x_i and x_{zi}
4. A random location on the hyperplane is selected by computing the formula below, where λ is a vector of the same dimension as the feature space, and x_{new} is the new synthetic sample.

$$x_{new} = x_i + \lambda \times (x_{zi} - x_i)$$

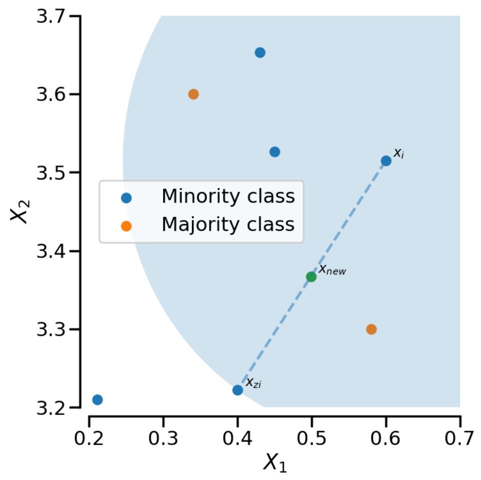


Figure 9. SMOTE Formula **Figure 10. Traditional SMOTE**

A visualization of SMOTE on continuous features in a two-dimensional feature space is shown in Figure 10. The same concept extends to multiple dimensions by using hyperplanes instead of lines.

Nevertheless, SMOTE can only be applied to continuous data. Even though the one-hot-encoding of our dataset allows the notion of "distance" to be interpreted to some extent, randomly picking a location in the hyperplane would result in non-discrete data entries, which violates the binary property of every input feature. This is where SMOTE-N shines. The SMOTE-N variation differs from traditional SMOTE in two aspects. Firstly, the k nearest neighbor search uses the Value Difference Metric (VDM) as the distance metric, which is more suitable for pure categorical feature space. Secondly, a new sample is generated where each feature value corresponds to the most common category seen in the neighboring samples belonging to the same class (Imblearn, n.d.). Instead of using the traditional SMOTE formula in Figure 9, if the k nearest neighbor (computed using VDM) of a random selected minority sample x_i all have "cough" as True then the newly generated sample X_{new} would have "cough" as True. The majority assignment results in some slight variations in the input features of the synthetic samples, which is better than randomly duplicating the samples from the minority class.

We implemented SMOTE-N via the imblearn library, and generated synthetic samples with a target negative-to-positive ratio of 5:1. We did not aim for a 1:1 proportion, as we hope to balance out the class distribution of the target variable while maintaining the innate structure of the raw dataset to some extent. Even though the synthetic samples generated via SMOTE-N have near-identical features to the real samples, a decrease in sample quality is inevitable, as "fake" samples are never as good as real samples.

Undersampling was then performed to further correct the class imbalance. One-Sided-Selection from imblearn was first applied, which is an ensemble method consisting of Tomek Links and Condensed Nearest Neighbors. Tomek Links identifies samples of different classes that are in close vicinity to each other. A Tomek Link is said to exist between x and y if the equation in Figure 10 is satisfied for all z in the sample space.

$$d(x, y) < d(x, z) \text{ and } d(x, y) < d(y, z)$$

Figure 11. Tomek Link Condition

In other words, if two samples from different classes are the nearest neighbor of one another, then a Tomek Link exists between them. Even though the euclidean distance metric is used for computing Tomek Links, applying it to our one-hot-encoded (binary) dataset is certainly justified and practiced in real-world applications, especially considering that only 6 features remain after feature selection.

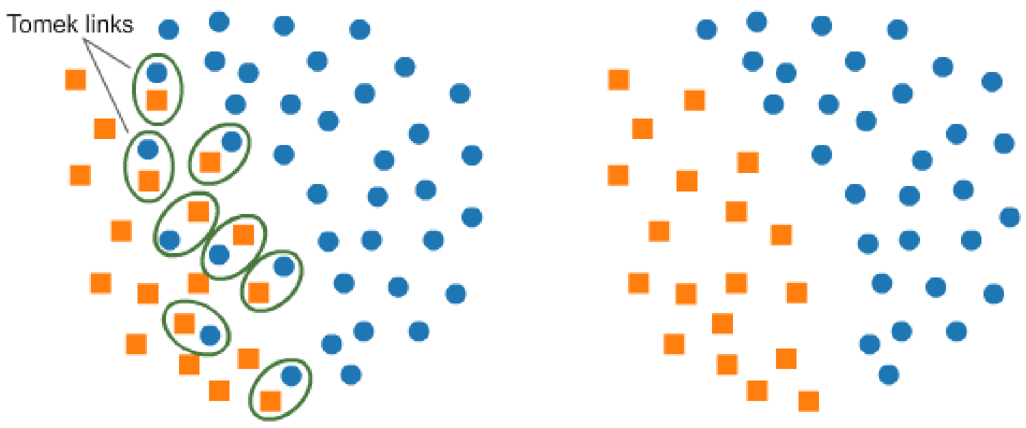


Figure 12. Tomek Links

Once Tomek Links have been identified, samples can be removed according to this criterion. In undersampling, samples from the majority class which belongs to a Tomek Link is removed. The underlying assumption is that majority samples which participate in a Tomek Link could be rare observations or noises, as only boundary and noisy instances will have the nearest neighbor from the other class. Removing these observations clarifies the decision boundaries (not necessarily linear) between the minority and the majority classes, making the minority regions more distinct. The primary goal of Tomek Links is not to make the class distribution more balanced, but rather to make the decision boundary less ambiguous. Undersampling can be seen as an added benefit from the Tomek Links technique.

Condensed Nearest Neighbor was then applied to the dataset after the Tomek Links have been processed. Condensed Nearest Neighbor seeks to find a minimal consistent set of the entire dataset such that 1-Nearest-Neighbor on this subset can still classify the samples as accurate as a 1-Nearest-Neighbor on the entire dataset. For imbalance undersampling, the process can be described as follows

1. Initialize a set S consisting of all the samples from the minority class
2. Add one sample from the majority class to S
3. For all samples in the majority class, classify it, sample by sample, using a 1NN based on the set S
4. If the sample is misclassified, add it to S. Otherwise, do nothing
5. Repeat step 3 and 4 until all samples in the majority class is classified

The idea behind Condensed Nearest Neighbor is that samples which are far away from decision boundaries do not contribute much to the construction of decision boundaries. This way, samples that are unnecessary for the task of separating classes can be removed. Condensed Nearest Neighbor benefits from Tomek Links in that Condensed Nearest Neighbor is sensitive to noise. By removing noise beforehand, we are able to establish clear decision boundaries for Condensed Nearest Neighbor to operate on. The method of combining Tomek Links and Condensed Nearest Neighbor is often referred to as One-Sided-Selection, discussed by Kubat and Matwin in their 1997 paper *"Addressing the Curse of Imbalanced Training Sets: One-Sided Selection"*.

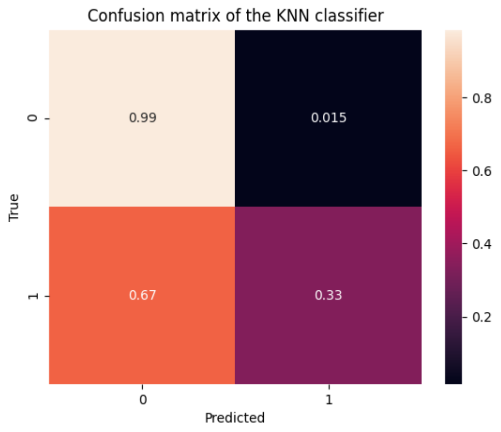
After applying One-Sided-Selection, we further reduced the dataset via another undersampling technique named Neighborhood Cleaning Rule, which is an ensemble of Condensed Nearest Neighbors and Edited Nearest Neighbors. Edited Nearest Neighbors is a method of removing samples that do not agree "enough" with its neighboring samples. In the case of imbalance undersampling, Edited Nearest Neighbors uses a kNN method to identify majority samples that disagree with the majority of all of its k nearest neighbors. The idea is similar to Tomek Links, in that samples who strongly disagree with its neighbors can be considered as noises or rare observations. Since Condensed Nearest Neighbors has already been applied to the dataset, using Neighborhood Cleaning Rule is roughly equivalent to applying Edited Nearest Neighbors alone, as another round of Condensed Nearest Neighbors would very likely yield the same result.

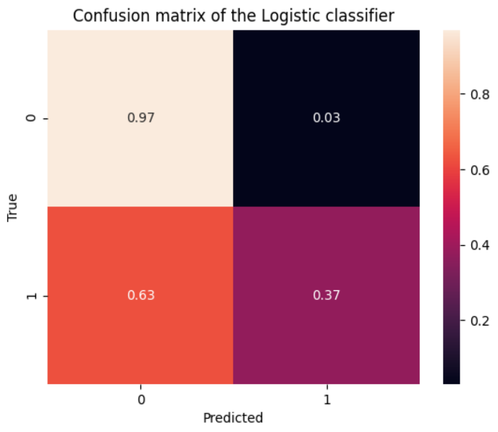
Undersampling resulted in the removal of approximately 1.8% of the observations from the majority class. This is both expected and optimal. We expect to see small amounts of removals because all methods except Condensed Nearest Neighbors focus on cleaning decision boundaries rather than reducing sample size. We think the result is optimal because we do not want to lose too much original data, and we think 1.8% is within a reasonable range.

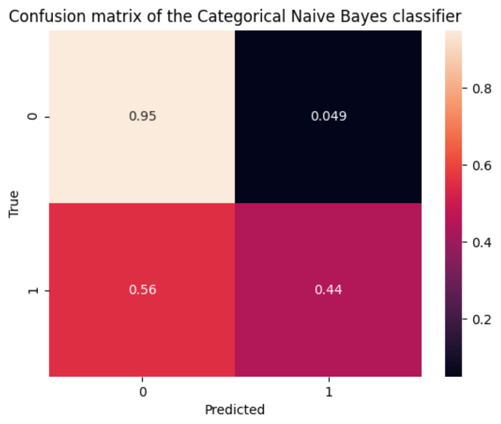
Notice that feature selection was performed prior to resampling, as we want to extract correlations based on the original data. The addition of synthetic samples and deletion of majority samples could heavily impact the outcome of feature selection. On the other hand, resampling techniques should never be applied to the test set under any circumstances, because the purpose of the test set is to evaluate model performance on real unseen data, and resampling techniques defeat this purpose. We attempted oversampling before undersampling and vice versa, as some authors suggest oversampling before undersampling results in slightly better performance. In our experiments, the results are nearly identical. We ultimately used oversampling before undersampling, and we do not think the order is significant.

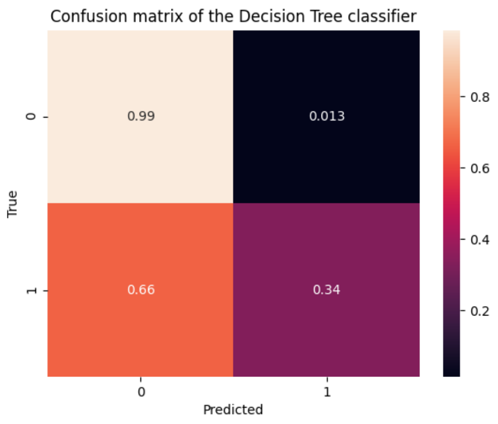
2.7 Model Selection

To enable spot-checking without accessing and contaminating the test set, we performed another stratified train-test split using the 80:20 ratio. We strictly “locked” the test set, as we are only allowed to access it once to evaluate model performance after model construction. To select the best model possible for this particular dataset, we spot-checked six potential classifiers, namely kNN, Logistic, Categorical Naive Bayes, Decision Tree, Random Forest, and XGBoost, all with default hyperparameters. The performance of these classifiers on the validation set is shown below as confusion matrices









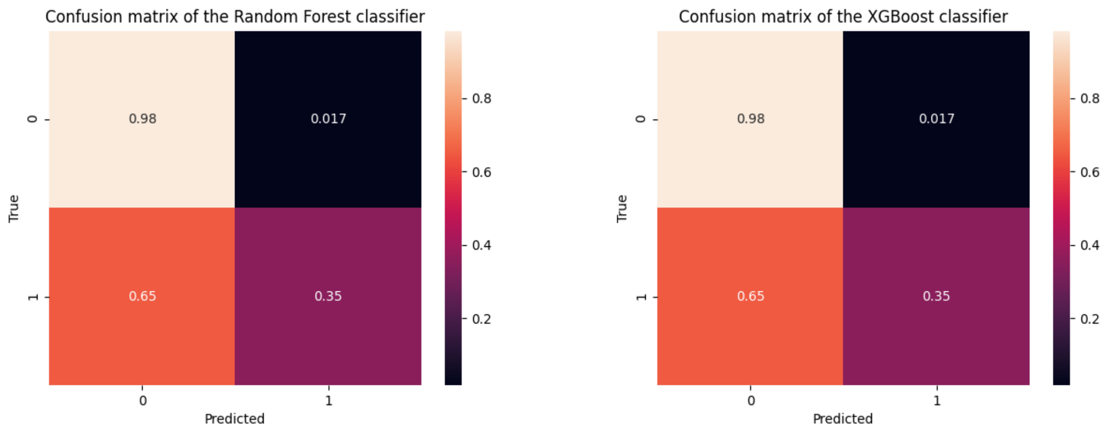


Figure 13. Spot-checking Baseline Performances

As shown in Figure 13, all classifiers have adequate performances, with kNN and Decision Tree having the lowest false positive rate and Categorical Naive Bayes having the lowest false negative rate. Ideally, we would like both rates to be as low as possible, but our imbalance dataset limits our ability to achieve both simultaneously. Since our primary goal is to help prioritize test kit distribution, we would like our model to identify as many positive cases as possible, even if it comes at the cost of wrongly identifying a negative patient as positive. In other words, we would rather test people who are potentially "positive", and have the result come out to be "negative", than leaving some true positive patients untested. From a machine learning perspective, this idea directly translates into minimizing false negative rate, even at the cost of increased false positive rate, as FN is a lot more important than FP. The best metric for this goal is recall.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

Figure 14. Recall Metric

As more positive samples are being correctly classified as "positive", the false negative rate decreases and the true positive rate increases. As a result, the recall metric increases.

The six classifiers constructed all have strengths and weaknesses. However, some classifiers are, by design, more suited for this specific problem. We ultimately decided to use Random Forest, and the reasons for eliminating the other classifiers are as follows

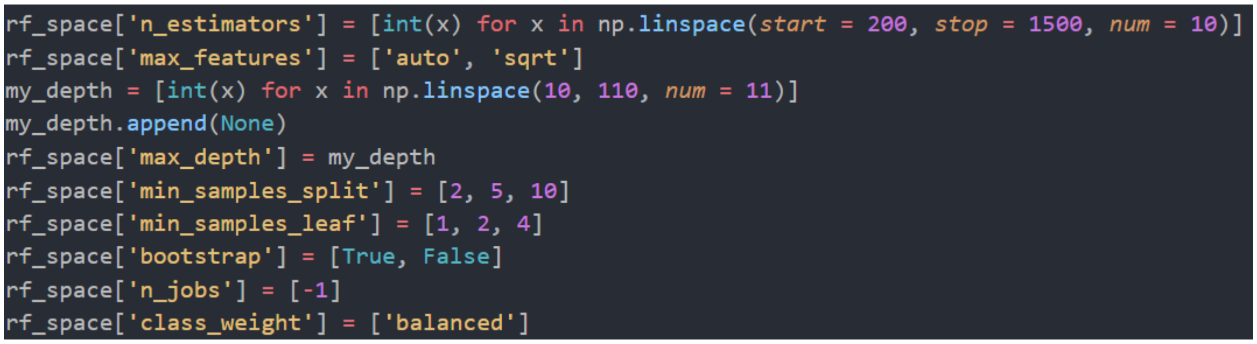
- kNN has the worst recall among all classifiers, and it suffers from the Curse of Dimensionality on high-dimensional feature spaces. kNN also allows very limited hyperparameter tuning (only the k value) and is slow to train and cross validate on large dataset. Furthermore, distance-based methods are not the best for categorical variables; there are other better methods available.
- Logistic regression constructs a linear boundary and assumes the existence of a linear relationship between the input features and the target variable. This assumption might not necessarily hold true. It is also not suited to model complex relationships, such as our six-dimensional feature space. It is also not the best performing classifiers, and hyperparameter tuning is limited.
- Categorical Naive Bayes is the best performing model in terms of recall, but tuning is almost non-existent. Tuning the Laplace smoothing hyperparameter does not necessarily guarantee performance increase. The best performing version of the Categorical Naive Bayes is very likely the model with default parameters. Therefore, no further tuning is needed

- Decision Tree is a weak learner which is prone to overfitting. It is also highly dependent on the training sample; small changes in training can result in huge changes in tree structure. Generally, Random Forest is preferred if applicable. The baseline decision tree model also underperforms compared to the Random Forest model.
- XGBoost performs the same as Random Forest. The two are in fact closely related. One difference is that Random Forest constructs each tree independently, while XGBoost constructs them in a sequence. These two ensemble methods generally offer similar performances, but XGBoost tends to be a lot slower to trai. We performed some experimental random search CV with XGboost, and it did not terminate in over 48 hours, while Random Forest terminated in less than 5 hours.

Random Forest might not have the best baseline performance in terms of recall, but it has huge potentials for hyperparameter tuning. Also, it is less prone to class imbalance, as it is forced to look at how classes are separated at each tree node.

2.8 Hyperparameter Tuning

We decided to first use Random Search Cross Validation to find the best hyperparameters for our Random Forest. We set up a repeated stratified 4-fold cross validation that repeats for 3 times at each iteration. The search space is attached below



```
rf_space['n_estimators'] = [int(x) for x in np.linspace(start = 200, stop = 1500, num = 10)]
rf_space['max_features'] = ['auto', 'sqrt']
my_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
my_depth.append(None)
rf_space['max_depth'] = my_depth
rf_space['min_samples_split'] = [2, 5, 10]
rf_space['min_samples_leaf'] = [1, 2, 4]
rf_space['bootstrap'] = [True, False]
rf_space['n_jobs'] = [-1]
rf_space['class_weight'] = ['balanced']
```

Figure 15. Random Search CV Search Space

We were primarily testing some commonly used hyperparameter values. This search space contains 4320 possible combinations of hyperparameters. Our Randomized Search CV runs for 2000 times, each time picking a random set of hyperparameter values and performing repeated 4-fold cross validation on it. Since the class distribution is imbalanced for the target variable, we want to optimize a metric that treats each class equally instead of giving more weight to the class with more samples. "Recall_macro" was used as the metric, which computes the recall independently for each class label of the target variable, and finds their unweighted mean. The best recall score and its associated hyperparameter values were recorded to a local text file, and Grid Search CV was applied to further tune the best hyperparameters by checking the immediate neighbors of each hyperparameter value. The result from Grid Search CV is then saved for prediction uses.

3 Predictions

3.1 Prediction Results

Random Forest was applied to the test set with the hyperparameters returned from Grid Search CV, the results are shown as follows.

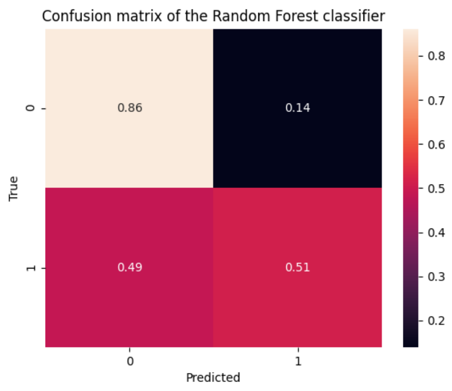


Figure 16. Confusion Matrix for Random Forest

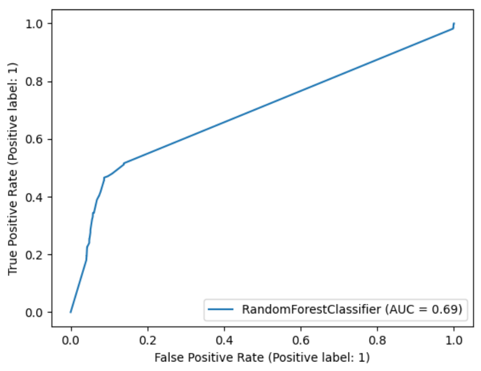


Figure 17. ROC Curve for Random Forest

The confusion matrix in Figure 16 shows a significant 16% improvement on classifying true positive samples compared to the baseline Random Forest model. It is also 7% better at classifying true positives compared to Categorical Naive Bayes, which is the best performing baseline model. The macro recall on the test set is 0.6863367892779657, and the ROC curve shows an AUC of 0.69. For comparison, we applied a baseline Random Forest model to the raw dataset, and it achieved a true positive rate of 4.8%. Clearly, feature selection, resampling, and hyperparameter tuning offers significant improvement to model performance.

4 Discussion

Given that the original dataset is severely imbalanced (99:1 ratio) with 46 features, classification is naturally a difficult task. Since we're not aiming to be as accurate as a swab test, we are quite satisfied with our model performance, which relies solely on six input features. It is worth noting that the testing results themselves might not be accurate, especially if a patient is early in his/her infection. We were able to correctly predict over 50% of the actual positive cases, which is quite helpful for prioritizing the allocation of test kits. Our model relies on the condition that symptoms and Covid-19 infection are related. This relationship is a comprehensively researched topic rather than an assumption. The primary limitation of our study is the availability of data and the class imbalance. If we were able to obtain more positive data to achieve a ratio anywhere from 5:1 to 1:1, our classification results would have been much better.

Our study raises several interest ideas that prompts further investigations. First, we hope to build an anomaly detection model on the original dataset, as extreme class imbalance suits the goal of anomaly detection. We also hope to access a more complete dataset, potentially with mortality information for those infected with Covid-19. We would like to apply a similar classification algorithm to predict the probability of developing severe illness or death based on clinical symptoms. This model would allow healthcare workers to allocate their time and resources to those who are in need the most. We are aware of the difficulties that we might face in obtaining such data. We will do our best to collect as much related data as early as possible, if we decide to pursue further investigations.

We have set random states for all function calls, and our results can be fully replicated. See Appendix A and B for data availability and code availability.

5 References

Carbon Health. 2020. “Covid Clinical Data.” Github. https://github.com/mdcollab/covidclinicaldata.

CDC. 2021. “Symptoms of COVID-19.” CDC.
https://www.cdc.gov/coronavirus/2019-ncov/symptoms-testing/symptoms.html.

CDC. 2021. “Older Adults.” Center for Disease Control and Prevention.
https://www.cdc.gov/coronavirus/2019-ncov/need-extra-precautions/older-adults.html.

HIPPA Journal. 2017. “Why is HIPPA Important.” HIPPA Journal.
https://www.hipaajournal.com/why-is-hipaa-important/.

Imblearn. n.d. “Over-sampling.” Imbalanced Learn Documentation.
https://imbalanced-learn.org/stable/over_sampling.html.

N3C. n.d. “N3C Data Enclave.” National Covid Cohort Collaborative. https://covid.cd2h.org/enclave.

Narayanan, Arvind, and Vitaly Shmatikov. 2008. “Robust De-anonymization of Large Sparse Datasets.”
IEEE Xplore, (May). 10.1109/SP.2008.33.

Nshomron. 2021. “covidpred.” Github. https://github.com/nshomron/covidpred/tree/master/data.

Shmerling, Robert H. 2021. “Which test is best for COVID-19?” Harvard Health Publishing.
https://www.health.harvard.edu/blog/which-test-is-best-for-covid-19-2020081020734.

6 Appendix

A Data Availability

The data for this project is located under the “data” folder of our project folder on github. The data folder contains all the data needed, with files structure organized to automatically work with our written code. The file “data_dictionary.csv” contains the data dictionary and explanations for each variable. Data is sourced from Carbon Health at https://github.com/mdcollab/covidclinicaldata/tree/master/data

B Code Availability

The code for this project is located directly under the main folder of our project. The code file is named “process-predict.py”. It can be run directly without modifying anything. The required packages are listed on top as imports. Text outputs such as hyperparameter values are printed to the command line, while graphs are both displayed and saved locally to the “output” folder. The Randomized Search CV step took over 4 hours to run on a 100 core CPU; expect to wait!