CMPT 459 Milestone Three/Course Project Report

Problem Statement:

Machine Learning has been used in medicine for many purposes like diagnosis and recommending treatment for ill patients. With the 2019 novel coronavirus (COVID-19) emerging nearly a year ago, there is still very little that is known. One such gray-area that remains to this date is predicting the outcome of COVID-19 cases. We can utilize Machine Learning to help detect the outcome of COVID-19. This can be beneficial to the overburdened healthcare systems around the world by helping them prioritize patients accurately and avoiding unnecessary deaths.

For this project, we consider four possible outcomes: Hospitalized, Nonhospitalized, Recovered, Deceased. Our ideal goal is to be able to take a specific COVID-19 case and predict its most likely outcome with a specific focus on the Deceased outcome. This outcome is the most important because it signifies that a patient will ultimately die due to COVID-19 infection. While the other three outcomes are important, the cost of predicting a case as not likely to be Deceased is far higher than misdiagnosing any of the other three categories since they are less likely to lead to certain death.

Dataset Description and EDA:

The data used for this project was made up of two datasets containing data about individual cases across several countries and summary data for several locations worldwide. The *Locations* dataset consisted of 3954 records and 12 features. Each record contained geographical data(latitude, province, country…) along with each locations 'Confirmed', 'Deaths', 'Recovered', and 'Active' COVID-19 cases. The *Cases* dataset consisted of 557364 records and 10 features. Each record was made up of data describing the case(age, gender, location…) along with its corresponding outcome. To better understand the datasets, we relied on three strategies. The first strategy involved calculating summary statistical values for each attribute like their mean or variance as seen in *Figure 1* and *Figure 2*. Since some attributes did not contain numeric values, some of their statistics could not be calculated.

	age	sex	province	country	latitude	longitude	date_confirmation	additional_information	source	outcome
count	260490	263630	550796	557340	557362.000000	557362.000000	556902	34395	348173	557364
unique	361	2	1179	135	NaN	NaN	171	20688	9596	4
top	35-59	male	Maharashtra	India	NaN	NaN	29.05.2020	Reconciled from MOHFW Table	PH Data Drop	nonhospitalized
freq	18019	145583	106515	301144	NaN	NaN	24301	1096	16053	250000
mean	NaN	NaN	NaN	NaN	18.138385	27.285771	NaN	NaN	NaN	NaN
std	NaN	NaN	NaN	NaN	20.455801	67.577194	NaN	NaN	NaN	NaN
min	NaN	NaN	NaN	NaN	-54.808030	-159.727596	NaN	NaN	NaN	NaN
25%	NaN	NaN	NaN	NaN	11.042850	-58.473080	NaN	NaN	NaN	NaN
50%	NaN	NaN	NaN	NaN	19.036810	72.834830	NaN	NaN	NaN	NaN
75%	NaN	NaN	NaN	NaN	28.456000	77.209100	NaN	NaN	NaN	NaN
max	NaN	NaN	NaN	NaN	70.071800	174.740000	NaN	NaN	NaN	NaN

Figure One: Cases Description

	Province_State	Country_Region	Last_Update	Lat	Long_	Confirmed	Deaths	Recovered	Active	Combined_Key	Incidence_Rate	Case-Fatality_Ratio
count	3786	3954	3954	3874.000000	3874.000000	3.954000e+03	3954.000000	3.954000e+03	3.952000e+03	3954	3874.000000	3906.000000
unique	562	188	3	NaN	NaN	NaN	NaN	NaN	NaN	3954	NaN	NaN
top	Texas	US	2020-09-20 04:22:56	NaN	NaN	NaN	NaN	NaN	NaN	Lewis, Idaho, US	NaN	NaN
freq	255	3270	3951	NaN	NaN	NaN	NaN	NaN	NaN	1	NaN	NaN
first	NaN	NaN	2020-08-04 02:27:56	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
last	NaN	NaN	2020-09-20 04:22:56	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mean	NaN	NaN	NaN	35.987924	-72.337046	7.760496e+03	241.740769	5.291398e+03	2.228632e+03	NaN	1538.572813	2.355818
std	NaN	NaN	NaN	12.872793	53.019022	4.184036e+04	1476.661533	5.315287e+04	4.440693e+04	NaN	1368.457332	3.538503
min	NaN	NaN	NaN	-52.368000	-174.159600	0.000000e+00	0.000000	0.000000e+00	-2.577446e+06	NaN	0.000000	0.000000
25%	NaN	NaN	NaN	33.270421	-96.611164	1.370000e+02	1.000000	0.000000e+00	1.140000e+02	NaN	618.904146	0.637729
50%	NaN	NaN	NaN	37.938284	-86.878438	4.985000e+02	9.000000	0.000000e+00	4.210000e+02	NaN	1204.351179	1.633422
75%	NaN	NaN	NaN	42.158587	-77.639009	2.129000e+03	48.000000	0.000000e+00	1.453000e+03	NaN	2097.993633	3.006621
max	NaN	NaN	NaN	71.706900	178.065000	1.167496e+06	37076.000000	2.577446e+06	3.379130e+05	NaN	14871.183644	108.812950

Figure Two: Locations Description

Our second strategy of Exploratory Data Analysis focused on understanding how incomplete our datasets were. This involved creating graphs of the data and seen in *Figure Three* and *Figure Four* which depict the number of missing values for each dataset. The purpose of this is to both estimate the amount of work needed to be done and come up with a strategy for imputing missing values. Since some attributes are related(latitude/longitude and country/province), it would be wise to use the attributes with less missing values to impute those which have more missing values(Ex: Imputing province using latitude/longitude).

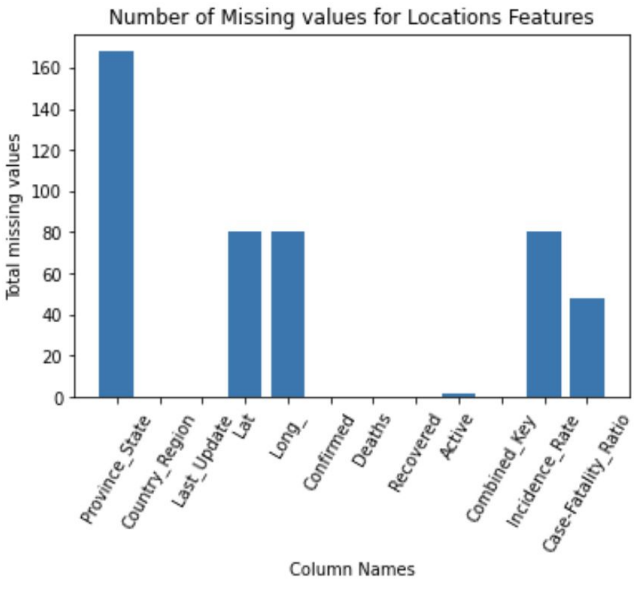

Figure Three: Missing Values for Locations

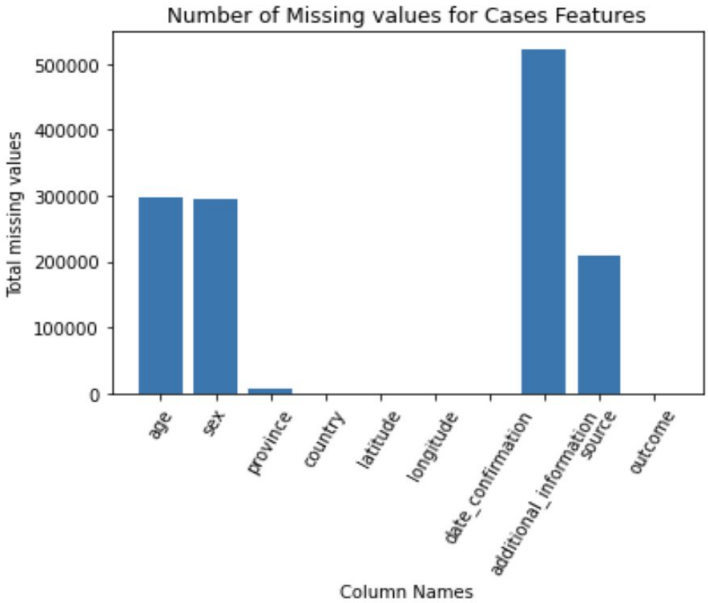

Figure Four: Missing Values for Cases

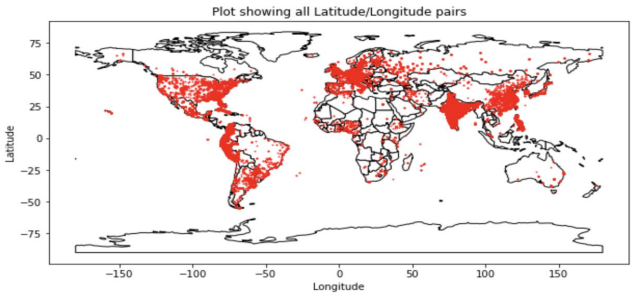

Figure Five: Worldwide Cases

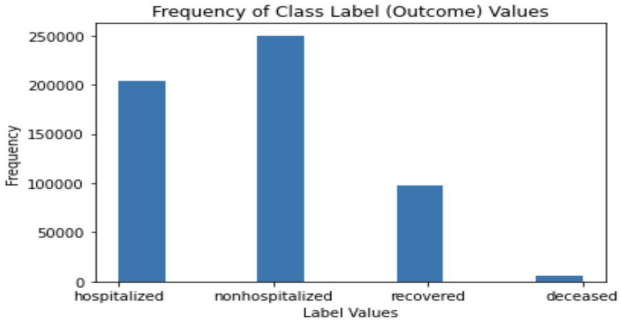

Figure Six: Class Label Values

The third strategy was manually inspecting certain attributes whose values were not immediately obvious. A good example of this is the "source" attribute from the individual locations dataset. After inspecting it, we found out that the majority of its values were website links containing the source of the information. Due to the difficulty in imputing the values, we immediately marked it as one possible feature which might be removed due to not being sufficient for our purposes.

In addition to the three strategies mentioned, we also decided to explore the geographical distribution of our individual cases as well as the overall distribution of the cases in terms of their outcome as shown in *Figure Five* and *Figure Six,* respectively. By looking into the geographical distribution, we were able to conclude that the majority of our cases were from the USA, Europe, India and East Asia. By exploring the distribution of the potential outcomes we were able to conclude that the "Deceased" class is a minority class and training a Machine Learning model which is accurate in general but also prioritizes the "Deceased" outcome will be challenging.

Data Preparation:

We began with the Individual *Cases* Dataset since it was the largest. To handle outliers and missing values, we started from the attributes which were missing the least or those which we thought were not going to be useful in the long run.

The "additional_information" attribute was immediately dropped since roughly 90% of its values were missing and the remainder seemed to contain additional notes made by whoever wrote the record. The "source" attribute had roughly 60% of its values missing and the majority of values were links to websites. To impute the "source" attribute, we filled all missing values with an empty string. The "latitude" and "longitude" attribute had only 2 missing values each, so we dropped both of them. The "date_confirmation" attribute had 500 values missing and they were imputed using the mean value of all dates.

The next attribute we focused on was “country”. This attribute had only 22 records with missing values and all of them had their “province” attribute set to “Taiwan”. To impute, we set the “country” attribute to “Taiwan” and left the “province” attribute with an N/A value which is imputed next. The next attribute we focused on was “province” and imputing the attribute was done in two steps. First, we created a lookup table consisting of all unique 3-tuples of a non missing “province”, “latitude” and “longitude” attributes. The lookup table was then used to fill in the “province” attribute for all records which had matching “latitude” and “longitude” values. While this approach seemed sound, it was only able to fill roughly 40 out of nearly 7,000 missing values. The remainder of the missing “province” values were imputed using reverse geocoding(i.e. Turn latitude/longitude into a location) via the Nominatim API.

For the “sex” attribute we computed the most frequent gender for each province, country and the entire world. Each record with a missing value had its “sex” attribute approximated by using either the most frequent province gender, country gender or world gender(in that priority). The “age” attribute was more difficult due to the inconsistent representation of its values. There were three types of representation used: an age range(“35-55” or “85+”), age in months(“18 months”) and exact age(“44”). The first two representations were converted into an exact year. Once the values were standardized, any missing values were imputed using the same frequency table method as the “sex” attribute.

Cleaning Locations Dataset

For the *Locations* dataset, we reused the majority of strategies mentioned previously. Province names, countries, latitude, longitude and dates were all cleaned and imputed in the same way. After filtering out bad records and outliers(mentioned in the next section), we were left with records that have 0 deaths and 0 confirmed cases. Since the Case Fatality Rate is defined as: Number Deaths/Confirmed Cases, we set any missing values to 0. Two records missing an “Active” attribute were dropped since they were missing many other attributes as well. For all records missing an “Incidence” attribute, we just set it to a “0” value. All other attributes in the locations dataset were not missing any values.

Handling Outliers

Outliers for the *Cases* datasets were handled in several ways. All records with ages below 0 or above 100 were removed. Records whose Latitude or Longitude values did not fit in the standard scale were removed. To check for any outliers in the “country” attribute, we used a premade list of all country names in the world. This let us catch inconsistencies like some records containing “Democratic Republic of Congo” while others used the name “Republic of Congo”. The province attribute contained too many values to check in any systematic way. Instead, we looked for any provinces containing any punctuation like “,” or “.” and replaced their value with an equivalent one not containing punctuation.

Outlier detection in the *Locations* dataset overlapped with those in the individual *Cases* dataset. The "Country_Region" attribute was checked for inconsistencies using the premade list of countries. It is also worth noting that while cleaning the dataset in the previous step, we noticed that some records missing a province or country were actually cruise ships(verified by looking up the name manually). These records were removed. Any records containing a "Last_Updated" value that is not in the year 2020 were dropped. Any records containing any summary statistics(Total Confirmed, Active, Deaths…) that were negative were dropped.

Merging Datasets

Before beginning to merge records we noticed that there were inconsistencies between the two datasets which would prevent a complete merge. The first inconsistency is that not all province/country names are written using ASCII letters. To fix this issue, we converted all province/country names into their ASCII versions. While this may not be a completely accurate conversion, it helped improve the odds of merging more records. The second issue was related to some province/country names being written in uppercase while others in lower case. To fix this issue, we converted all province/country values into lowercase only.

The two datasets were merged using two merges. The first was on country/province pairs. This left about 100,000 out of 560,000 records unmerged. In an effort to not lose this data, we decided to merge again but this time only on the country attribute. As a result, we were able to get the total number of unmerged records from 100,000 down to just 7 which were then removed. In summary, we lost roughly 200-300 records from our original dataset.

Classification Models:

During this project we trained three classification models: AdaBoost, Random Forest and K-Nearest Neighbours(KNN). One of the project restrictions was that one of our models must be a variant of the Boosting tree. It could be either XGBoost, AdaBoost or LightGBM. Hence,we picked AdaBoost as our first classifier. The reason behind it was that the ensemble of trees in this model is not independent like in the case of Random Forest, but rather each tree is designed such that all the trees can work together. This allows us to weigh the training data which increases the accuracy and hence reduces overfitting. All the training points get assigned equal weights at the beginning of the training and with each iteration, the weights of the correctly classified points are decreased and the weights of the incorrectly classified are increased. When used with decision tree learning, information gathered at each stage of the AdaBoost algorithm about the relative hardness of each training sample is fed into the tree growing algorithm such that later trees tend to focus on harder-to-classify examples.The decision trees are grown so that they favor splitting sets of samples with high weights.

Our second chosen model is a Random Forest classifier. Random Forest Classifiers are an ensemble of individual decision trees, they generally outperform decision trees (we did not

strictly base it off of that assumption), but their accuracy is lower than gradient boosted trees. However, data characteristics can affect their performance. Random decision forests correct the decision trees' habit of overfitting to their training set. Trees that are grown very deep overfit their training sets as they tend to learn highly irregular patterns. They have high variance with low bias. Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance.

The third model that we used was the K-Nearest Neighbours (KNN) model. A KNN model calculates similarity using the distance between two points on a graph. The greater the distance between the points, the less similar they are. We used this model because it requires no assumptions and is relatively simple to implement. It consists of a single parameter which can be tuned to observe the variance. We tried using one-hot encoding for province and country features, however, it resulted in too many columns. We could not possibly train our model with those many columns and we kept on running out of memory (in CSIL,GoogleCollab and on our local computer as well).

Initial Evaluation and Overfitting:

Before training the initial versions of our models, we used the Mutual Information scores in an effort to reduce the total number of features to improve both total training time taken and the accuracy of our models. The majority of our features had a score between 0.65 and 0.70. Surprisingly, the “sex”, “age”, “date_confirmation” and “Last_Update” attributes had an mutual information score at or below 0.5. Due to this, they were removed. Additionally, we removed the second set of latitude/longitude pairs which were from the locations dataset since their information was duplicated in the latitude/longitude pairs from the individual cases dataset.

To evaluate each model, we used the Train/Test accuracy scores, the classification report for Recall/Precision and a confusion matrix. The train and test scores for the AdaBoost were 0.8086 and 0.8069 respectively. By observing the confusion matrix and the overall classification report, we concluded that the “deceased” and “recovered” outcomes had very low scores due to them being minority classes. This tells us that AdaBoost is accurate in predicting the majority classes but it will need additional tuning to better predict the “deceased” outcome.

For our Random Forest, both the Train and Test sets resulted in an accuracy score of 0.81 as well. One of the major differences was that the Precision/Recall and the Confusion Matrix show that the Random Forest performed much better in identifying the “deceased” class when compared to AdaBoost. Our KNN model had a training score of 0.79 and a testing score of 0.79. The Precision/Recall and Confusion matrices seem to indicate that this model is still weak in terms of predicting the “deceased” and “recovered” labels. Even though it might have a lower test score than other models, it still did better than AdaBoost when it comes to handling the minority classes.

To evaluate whether our models are overfitting, we used two methods. The first method was the train/test score and the second method was to check for any difference in the score while varying only one of the available hyperparameters.

AdaBoost obtained 0.8086 and 0.8069 as its train and test scores respectively. Since the scores are almost identical, they do not indicate any overfitting. For our second method, we trained AdaBoost with a varied depth of the Decision Tree Classifier used to build it as seen in *Figure Seven*. The depth was between 1 - 10. While the overall accuracy changed and some values of the hyperparameter ended up leading to an underfit, the train and test data stayed within 2% of each other which suggests that our model is at the very least not overfitting our data.

Our Random Forest obtained a train/test score of 0.81 which does not indicate any overfitting. For our second method, we altered the max_depth hyperparameter of the decision tree used for the random forest. To check for overfitting, we trained the RandomForest Classifier with a depth between 1 - 10. With increasing tree depth, the Random Forest Classifier Train/Test scores also increased in proportional amounts with them amounts differing by approximately +/- 0.0025 which indicates that the Random Forest Classifier is not overfitting as well.

To check our KNN model for overfitting, we changed the nearest_neighbors hyperparameter from its default value of 5 to all even values in the range of 2-10, and then we did a comparison of the accuracy score of each. After observing the values, we realized that the scores for the test data and the training data were very close. It means that there is no overfitting.

Hyperparameter Tuning:

The initial recall of our models on the "Deceased" outcome was between 0% and 1%. In an attempt to maximize this, we used the Grid Search technique for optimizing the hyperparameters of our models. Grid Search can be seen as being a "brute force" approach to hyperparameter optimization. Given a set of hyperparameters and a set of values for each hyperparameter, Grid Search attempts to find the maximum value of a certain metric(Recall on the "Deceased" outcome in this case) by training and fitting all possible combinations of hyperparameter values. This approach is computationally expensive and might be infeasible for a large number of parameters but for the purposes of this project it seemed like the best choice which will provide the most accurate values.

The Random Forest Classifier had 19 hyperparameters to tune. We decided to tune the 'n_estimators,' 'max_depth,', and 'min_samples_split' hyperparameters as it gave us the best results. The n_estimators hyperparameter indicates the number of trees in the forest (values: 10, 50, 100, 150, 200). The 'max_depth' hyperparameter indicates the max depth of the tree (values: 5,10,20, None), where None means that the nodes in the tree are expanded until all leaves contain less than min_samples_split samples or until all leaves are pure). The last

hyperparameter we tuned was ‘min_samples_split’ (values: 2,4,6) which indicates the minimum number of samples required to split an internal node.

For KNN classifier , we tuned the hyperparameters ‘n_neighbors’ (values: 2,4,5,10,20,50) ,‘p’(values: 1,2) and ‘weights’(values: uniform,distance).We used n_neighbors because it is the most important parameter.The power parameter for the Minkowski metric (p) corresponds to the kind of distance used (1:Manhattan and 2 : Euclidean). For the ‘weights’ hyperparameter, uniform weight means that all points in each neighborhood are weighted equally. For the ‘distance’ hyperparameter, the weight of the closer neighbors of a query point will have a greater influence than neighbors which are further away.

AdaBoost had 4 different hyperparameters which could be tuned. The first parameter we chose was “n_estimators”(values: 10, 50, 100, 150, 200) which dictates how many estimator models(Decision Trees in this case) are used before the model terminates. The second parameter was the “learning_rate”(values 0.001, 0.01, 0.1, 0.5,1) which dictates the contribution that each estimator has. The third and final parameter was the default Decision Tree used as an estimator. The “max_depth”(values: 1, 3) of the Decision Tree specifies its depth and controls how accurately the tree can model the data.

Initially, we tried to use Grid Search on our original cleaned dataset but due to the “Deceased” outcome being a minority class, we saw roughly 1%-2% improvement in accuracy. In an attempt to maximize the value of the “Deceased” recall, we decided to employ some sampling techniques in order to increase the value of recall. We attempted to use four sampling techniques: Oversampling the minority class, Undersampling the majority class, generating synthetic data using the SMOTE algorithm for our minority class(similar to oversampling) and using a stratified sample for our test and training sets.

Each one of our three models reacted in different ways to each sampling approach. AdaBoost performed best by using stratified samples of our dataset. Any other approach resulted in either underfitting or overfitting. Our Random Forest classifier performed best when the minority outcome(“Deceased”) was oversampled. All other approaches resulted in severe overfitting or underfitting of the model. Our KNN classifier did not seem to be impacted heavily by any sampling approach. After tuning, its recall value was roughly 0.08%. The only approach which significantly improved the accuracy of KNN was using drastic undersampling of all classes such that each possible outcome had roughly 6,000 records in total. While this approach did manage to boost the recall of the model, it was unacceptable since it required sacrificing nearly 90% of our training data.

Results:

Classification Models	Hyperparameters	Accuracy	Overall Recall (Weighted Average)	Recall on 'deceased' Outcome
AdaBoost	max_depth= 3 learning_rate=1 n_estimators=150	Train=0.70 Test=0.70	0.702	0.230
Random Forest	max_depth=20, min_samples_split=2 n_estimators=150	Train=0.70 Test=0.74	0.736	0.477
K-Nearest Neighbours	n_neighbors=2 p=2 weight=uniform	Train=0.79 Test=0.79	0.795	0.082

The results we obtained from our tuned models along with the best hyperparameters can be seen in the table above. Overall, there are three things worth noting. First, the overall accuracy of all three models dropped. This is largely due to the tradeoff between increasing the recall for a particular class("Deceased" in this case). Second, we were able to increase the recall on the "Deceased" outcome by a large amount for each classifier. Without hyperparameter tuning, we were only able to accomplish a recall of 0%-1% but after hyperparameter tuning, our values increased by 20%-45% depending on the model used. Finally, it is important to acknowledge that even though there is an improvement in our values, our models still need improvement. An ideal model would have a high overall accuracy and a high recall on the most important classes. Unfortunately this is not possible to accomplish with the current dataset. (More on this in the "Future Improvements" section)

Conclusion:

On the test dataset, we were able to achieve a maximum Recall score of 0.477 and a maximum Test Accuracy score of 0.74 with our Random Forest Classification model. Although the K-Nearest Neighbour Classification model had a higher accuracy, the Random Forest model was the one that provided us with the best overall recall which was the focus of this project.

Even though the test set accuracy was 4% higher than the training set, this does not indicate any overfitting. If the model was overfitting, the training set accuracy would be higher than the test set accuracy. The Random Forest model is also not underfitting our dataset. The original accuracy of the model was 81% and after oversampling this was reduced to 74%. While we would desire a higher accuracy, there is a tradeoff between improving the recall score of a particular class and the overall accuracy of the model. The reduction of roughly 7% in the overall

accuracy seems to be necessary for us to be able to improve the recall from roughly 1% to almost 48%.

Lessons learnt and future work:

During this project there were four main lessons learned. The first lesson was that there is a strong inverse relationship between overall accuracy and recall. As it can be seen from our results, any attempts to improve the recall on a particular class lead to a reduced overall accuracy of our model. The second lesson we learned is that Hyperparameter tuning is very computationally expensive. All of our models were trained using the Grid Search method for hyperparameter optimization and each one of our attempts took between 1.5 - 3 hours to complete while the workload was distributed between 5-8 concurrent jobs.

The third lesson learned is that dealing with a minority class("Deceased" outcome in our case) is very difficult to do without access to more data. Different models require different sampling approaches (oversampling, stratifying) and even then, the results still might not be acceptable. The fourth and final lesson we learned is that imputing data is ultimately not very accurate. Any time we impute data, we are only using approximations to its real values. As a result, datasets which require a lot of imputation(like we had to do for the "sex" or "age" attributes of the individual cases) will not be very accurate. Overall, the amount of inaccuracy in a dataset heavily depends on the number of values which must be imputed.

When it comes to future improvements for this project, there are two obvious choices. The first improvement is exploring different models like State Vector Machines or Linear Regression Classifiers. It is possible that these classifiers will be better when it comes to representing the problem we are trying to solve. The second improvement will involve gathering more data to improve the distribution of our dataset. Currently, the "Deceased" outcome is our main focus but this outcome is a minority class. If we can retrieve more records, we should be able to improve both the recall and overall accuracy of our models.

Contributions:

Yavor Konstantinov: Training AdaBoost, Cleaning Data, Tuning AdaBoost Hyperparameters, Writing/Editing Reports. Maintaining Gitlab Repository
Elia Karimi Sisan: Training Random Forest, Cleaning Data, Tuning Random Forest Hyperparameters, Writing/Editing Reports, Maintaining Gitlab Repository
Arpit Kaur: Training KNN, Cleaning Data, Writing/Editing Reports, Tuning KNN Hyperparameters, Maintaining Gitlab Repository.