CMPT 459 - Milestone Report

Simon Fraser University
Authored By:
Team Data Diggers: *Neil Shah, Murtaza Mushtaq, Ronit Chawla*
Supervised By:
Prof: Dr. Martin Ester
TA: Madana Krishnan Vadakandara Krishnan, Rhea Rodriguez

I. Problem Statement

The problem we are tackling is accurately predicting the ouctome of COVID-19 patients. Our chosen metric to measure success is accuracy score, precision, recall and f1-score of our predicted outcome.

The goal of this project was to choose, build, train, and test three classification models using two publicly available COVID-19 datasets. The project is divided into three milestones, each comes with specific task goals that were necessary to achieve in order to move on to next milestone.

Here are the task goals for each milestone:

- **Milestone 1:** Data Preprocessing
- **Milestone 2:** Building Baseline models and used metrics to evaluate performance of each model
- **Milestone 3:** Predict outcomes after hypertuning the baseline models. Also predict the "deceased" label correctly to achieve high recall on class "deceased"

II. Introduction

We are working with two publicly COVID-19 datasets:

- **Case dataset:** contains data for individual cases that tested positive for COVID-19
- **Location dataset:** contains the number of cases based on location

The patient-level datasets we are working with comes from the public domain, and more specifically from various government disease control centres and news sources around the world. To give a bit more context on the datasets, the Case dataset had 557450 total cases and the location dataset had a total of 3905 locations for the COVID-19 cases.

The broad approach we undertook for this project was:

1) Data exploration
2) Data preprocessing
3) Baseline model training, testing and validation
4) Hyperparameter tuning for improvements in predictions and higer recall on class "deceased"

Initially, we considered multiple classification models including XGBoost, AdaBoost, SVMs, KNN, Decision Trees, Random Forests, and Naïve Bayes, but concluded that XGBoost, KNN, and Random Forests were our best choice. This was due to observing that XGBoost and Random Forest Trees performed faster than AdaBoost and Individual decision trees on the project dataset, respectively. In addition to this, Naïve Bayes assumes that data is independent and since the data provided was highly correlated with both numerical and categorical features, we decided to go with KNN.[1]

III. Dataset Description and EDA

For the exploratory data analysis, we went through the datasets and identified the basic information about both the datasets that included, the count of total rows, columns, data type for the individual columns and number of non null values. Later, We digged more and learnt how many unique grouped values each attribute holds. Also, their statistical attributes were described to learn about the mean, median, mode, standard deviation etc. To calculate how many null values are there in every attribute we first used a heat map [Fig 1] and later used code to get the actual figures.

Having done the initial analysis we tried to explore the data and get some deep insights, For thus we used data visualization software, PowerBI. We were able to explore in a location dataset which country had the most number of confirmed cases, deaths, recovered and active patients [Fig 2]. We were able to explore in a location dataset, there were 30685001 confirmed cases out which 8807554 were active and 20922189 had recovered. Through some

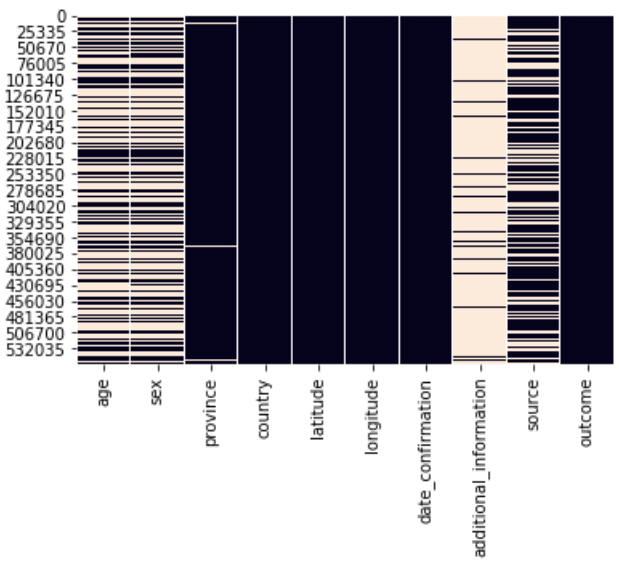

Fig. 1. Case Dataset Heat Map to detect NULL values

bar graphs, we were able to identify which countries had the most number of confirmed cases, deaths, recovered and active patients.

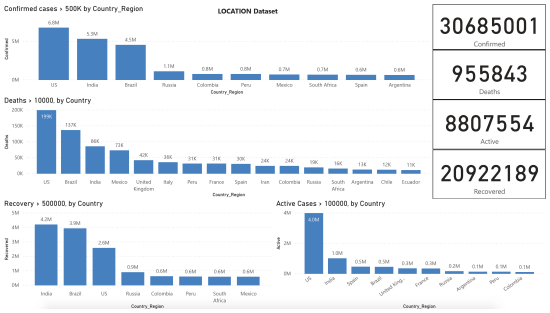

Fig. 2. Location dataset confirmed cases, deaths, recovered and active patients

Via pie charts we inspected that US provinces had the maximum case-fatality ratio and incidence rate amongst the entire location dataset[Fig 3].

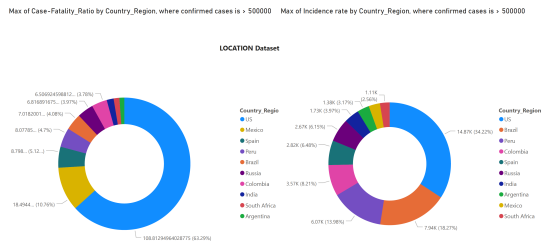

Fig. 3. Location Dataset with maximum case-fatality ratio and incidence rate

In the cases dataset there are a total of 557450 cases out of which 201377 were hospitalized, 249991 were non-hospitalized, 98112 had recovered and 5998 had deceased. From the latitudes and longitudes given, we made a world map of most cases around the world. This plays a great role in storytelling and displayed that India had the most number of hospitalized patients[Fig 4].

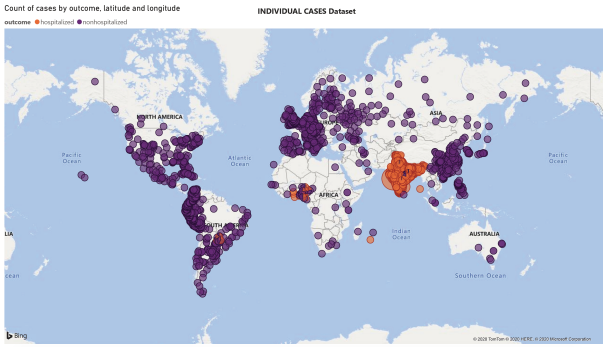

Fig. 4. World Map representing most number of hospitalized patients

Also, one very interesting thing that we noticed about the cases dataset was that more number of male gendered people tested positive for the COVID-19 when compared to female gender. This opened us many interesting questions, like is it due to the imbalance of male and female population ratio or any interesting key influencers are involved? Although this trend remained the same but the overall detection of COVID-19 as an outcome showed a downfall as and when the days passed.

In order to improve our understanding further, we systematically explored features through correlation analysis [Fig 5] and feature importance. These methods provided further insight into how the data was structured as well as any important underlying relationships between variables. The fruits of this exploration formed the basis upon which our models were built.

IV. Data Preprocessing

Prior to training any machine learning algorithm, careful preprocessing is required to convert data into the appropriate form. In our dataset, many features were not structured homogeneously or consisted of string variables. In the subsections below, we detail our preprocessing steps:

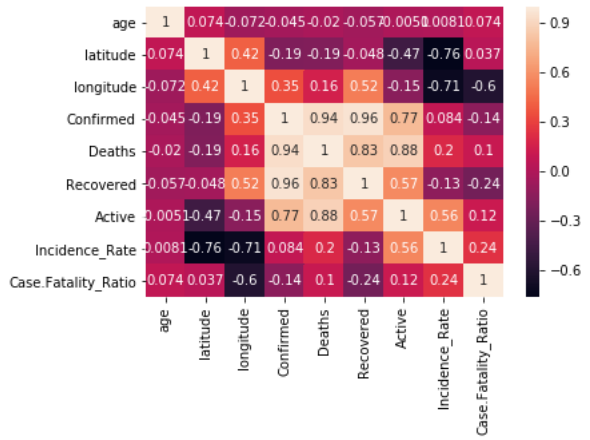

Fig. 5. Correlation Analysis Heatmap

A. Exploratory Data Analysis

We went through the datasets and identified the basic information about the dataset that included, the count of total rows, columns, data type for the individual columns and number of non null values. Later, We digged more and learnt how many unique grouped values each attribute holds. Also, their statistical attributes were described to learn about the mean, median, mode, standard deviation etc. To calculate how many null values are there in every attribute we first used a heat map and later did some coding to get the actual figures. Having done the initial analysis we tried to explore the data and get some deep insights, For thus we used data visualization software, PowerBI. We were able to explore in a location dataset which country had the most number of confirmed cases, deaths, recovered and active patients. Via pie charts we inspected that US provinces had the max case-fatality ratio and incidence rate. For the cases dataset, latitudes and longitudes were used to make a beautiful world map which did play a great role in storytelling and displayed that India had the most number of hospitalized patients. We also created line charts and bar graphs for the cases dataset to investigate which country had the most number of cases. Also, Shockingly we recognized that more of male patients were getting tested positive to covid when compared to females. This opened us many interesting questions, like is it due to the imbalance of male and female population ratio or any interesting key influencers are involved?

B. Data Cleaning and Imputation

We started off with the 'Age' column and removed all '+' signs in addition to replacing the empty 'Age' rows with 'NA' temporarily. We then replaced all the ranges in the 'Age' column with the average value of that particular range. For instance, '23-27' was now just '25' since

$$\frac{23+27}{2} = 25$$

Furthermore, all ages that were in months, were changed to years to be consistent with the rest of the values in the column. Therefore, '48 months' was now '4 years'. The temporary 'NA' values in the column were imputed with the mean ages of the province that the individual case belonged to. We also added new columns that could hold the corresponding min, max, range, missing value, the average age in binary data (0 and 1). For the missing data in the 'province' column, we used the given country to find the province of the confirmed case and imputed that for a particular case. Moreover, all the missing values in the 'Case-Fatality-ratio' column were computed by using this equation:

$$\frac{Deaths}{ConfirmedCases} * 100$$

The missing values in the 'incidence rate' column for a particular case were replaced with the average of the incidence rate for the country and province of that particular case. We imputed the missing 'sex' values according to the sex ratio in the corresponding province of that particular case. Last but not the least, data in the 'outcome' column was transformed into a binary column.

C. Dealing with Outliers

Next, we moved onto detecting outliers in the data. We made use of the concepts learned in class by using isolation forest, k-nearest neighbors mean, Mahalanobis distance, and Robust-Kernel-based Outlier Factor. Mathematically, we used the help of the inter-quartile range to help us identify outliers that were beyond the IQR range. Also, we used box plots to visualize the extreme values and outliers for the location dataset. Z-score was calculated and all the records with Z-score ¿ 3 were identified as outliers. On top of this, we calculated the largest difference between values and the sample mean as a means of detecting outliers and plotted a scatter

plot graph using ggplot to visually identify and confirm outliers in the data.

D. Transformation

We first subset the records for country US in both data sets. We look for any unusual values just for rectification purposes. Then we take the second dataset and transform by grouping it by 'provinces'. We then used the merge or the left join function to merge both data frames together where the common attribute was 'Province' and all the desired columns were aggregated on their mean to get the resultant dataframe.

E. Joining the Cases and the Location Dataset

We now had to create a single merged dataset to use in the next parts of the project, so we used the 'province, country' as a common feature for merging. We first grouped the second dataset by 'Province' and 'Country'. Then we used the 'merge or left join' function in R to merge both the datasets. We were aware that a simple join might lead to many of the rows being empty hence, wherever provinces were empty, we merged using just the country. All the desired columns were then aggregated on their mean which gave us the merged dataset that was adjusted for consistency in matching provinces and countries

F. Categorical Variable Handling

We used a Label Encoding process on the data to handle all categorical variables. This process takes categorical variables such as "outcomes" column which is our target column and converts it into numerical representation without an arbitrary ordering. The categorical variables 'city' and 'province' had little predictive value as there was significant colinearity between features, thus extensively analysis was performed to remove a majority of these elements from our final predictive feature list.

G. Cross Validation

Cross-validation is an organized way to do repeated data splitting, it produces estimates of metrics with less bias than a traditional train-test split. In this module, we randomize and split the training dataset so that 75-80 percentage of the data we used for model training and validation. Each iteration of this splitting is called a "fold". We used Stratified K-Folds cross-validator, in which the folds are made by preserving the percentage of samples for each class. In our cross validation, we usually specified five folds. Each fold served as a testing set one time and training set for predicting each of the other four folds. This algorithm is repeated for each fold and at the end we obtained the average metric.

V. Classification Models

Prior to training any machine learning algorithm, careful preprocessing is We were tasked with building three classification models and training/testing them using the merged dataset obtained in the last section. We started off by splitting the merged into two parts by using a random sample generator. The first 80 percent of the dataset was used for training and the last 20 percent was used for testing our classification models.

We decided to go with XGBoost, KNN, and Random Forests as our classification models. XGBoost which stands for eXtreme Gradient Boosting is commonly known for its execution speed and model performance by reducing overfitting. In addition to this, we also observed that it performed faster than AdaBoost during execution since the parallelization used in the code reduces runtime while modeling. Amongst the provided choices for classifiers in milestone 2, SVM uses a kernel method to find out the classes. Since our COVID dataset is quite large, we observed that it took more time to train the model and is likely to overfit. Moreover, since we had both numeric and categorical features in the dataset and KNN is a distance-based algorithm, we had the choice to this for modeling. Random Forests classifier, which is based on decision trees, outperforms independent decision trees where the dataset is large and overfitting occurs, so we decided to go with Random Forests. Other than being large, our COVID dataset was observed to be highly correlated so there was no way we could use Naive Bayes since it assumes the features in the dataset to be independent.

A. XGBoost Classifier

XGBoost is a decision tree based ensemble algorithm that uses a gradient boosting framework. It is mainly used to handle unstructured non linear data where a neural network would overfit too easily. It utilizes decision trees, bagging and boosting, which is the process of minimizing

the errors of the previous models while boosting the influence of better performing models.[2]

B. Random Forest Classifier

Random Forest is an ensemble learning method for classifications and regression. It fits a number of classifying decision trees or sample subsets of dataset and uses averaging to improve accuracy and control overfitting.[3]

C. KNN Classifier

KNN is a non-parametric (there is no assumption for underlying data distribution) and lazy learning (it does not need any training data points for model generation) algorithm. All training data is used in the testing phase. In KNN, K is the number of nearest neighbors. For finding closest similar points, you find the distance between points using distance measures such as Euclidean distance, Hamming distance, Manhattan distance and Minkowski distance. KNN first calculates the distance, then finds closest neighbors, and finally, votes for labels.[4]

VI. Initial Evaluation and Overfitting

We evaluated our chosen three models using the training and test data by using certain metrics. The metrics used were, Accuracy score, Confusion metric, Precision, Recall, and F1-score. Accuracy gave us the percentage of true classification. Confusion metric told us the count of 'True Positive', 'False Positive', 'True Negative', and 'False Negative' classifications. Precision is the rate of 'True Positive' over 'Predicted Positive' while Recall is the rate of 'True Positive' over 'Actual Positive'. F1-Score is the harmonic mean of Precision and Recall. Note that analysis of all the metrics mentioned here other than accuracy is important for the given problem as it contains multiple classes that are not balanced. Due to the imbalanced nature of the dataset, there is a high probability that the model will be biased toward the higher classes. So, just calculating the accuracy will not give us the true measure. We have created the three following tables containing the results for each classifier model and the metric values so that it can be easier for readers to observe that our models are a good fit.

Based on the accuracy we don't see overfitting but as we have imbalanced classes in the dataset, we will check for the other evaluation metrics. Here we can see

XGBoost with 20 estimators	Accuracy (%)	Precision	Recall	F1-Score	Support
Train	78.27	0.00	0.00	0.00	4080
		0.67	1.00	0.80	157685
		1.00	1.00	1.00	128262
		0.74	0.00	0.00	75383
Test	78.14	0.00	0.00	0.00	1034
		0.67	1.00	0.80	39499
		1.00	1.00	1.00	31870
		0.74	0.00	0.00	18950

TABLE I
BASELINE XGBOOST CLASSIFIER RESULTS ON TRAIN AND TEST

Random Forests with 20 estimators	Accuracy (%)	Precision	Recall	F1-Score	Support
Train	81.11	0.89	0.06	0.11	4080
		0.70	0.98	0.82	157685
		1.00	1.00	1.00	128262
		0.78	0.19	0.30	75383
Test	80.93	0.55	0.04	0.07	1034
		0.70	0.97	0.82	39499
		1.00	1.00	1.00	31870
		0.78	0.19	0.30	18950

TABLE II
BASELINE RANDOM FOREST CLASSIFIER RESULTS ON TRAIN AND TEST

KNN with 20 estimators	Accuracy (%)	Precision	Recall	F1-Score	Support
Train	80.71	0.70	0.03	0.06	4080
		0.71	0.96	0.81	157685
		1.00	1.00	1.00	128262
		0.69	0.21	0.32	75383
Test	80.61	0.61	0.03	0.06	1034
		0.71	0.95	0.81	39499
		1.00	1.00	1.00	31870
		0.69	0.22	0.33	18950

TABLE III
BASELINE KNN CLASSIFIER RESULTS ON TRAIN AND TEST

that metric values for both the train and test are the same in the case of XGBoost and the values are discouraging compared to the other two models. Although the metric values are similar for train and test cases in Random Forest if we see the first class, compared to Random Forest, KNN seems to perform better in test cases.

The models trained in this case seem to be a good fit model and neither overfitted nor under fitted. To find out whether the model is overfitted or not, we check

the metric value for the test dataset and compare them with the evaluation scores of the training dataset. If the performance of the model is very good on train data but it performs poorly on test data, then the model is said to be an overfitted model. As in our cases, all the models are equally performing well on both the train and test datasets, we can say that the trained models are a good fit.

VII. Hyperparameter Tuning

By training the initial model, we were able to fit the model parameters and obtain certain baseline results. These results were not the best results the model could perform on. That is why hyperparameter tuning was used to obtain the best result from a model. Parameters which define the model architecture are referred to as hyperparameters and thus this process of searching for the ideal model architecture is referred to as hyperparameter tuning [5] [6].

Models can have many number of hyperparameters and finding the best combination of these hyperparameters is a problem that can be solved within the domains of "search problems". In this project, we used the two most common and best strategies of Hyper parameter tuning[7]:

- **GridSearchCV:** Grid search is an approach to hyperparameter tuning that will methodically build and evaluate a model for each combination of algorithm parameters specified in a grid. That is, we simply build a model for each possible combination of all of the hyperparameter values provided, evaluating each model, and selecting the architecture which produces the best results. We used Stratified K-Folds cross-validator, in which the folds are made by preserving the percentage of samples for each class. In our cross validation, we usually specified five folds. Also we wanted the best model to be the one which has the highest recall on class "deceased", therefore, we refitted the model according to recall on class "deceased". For the scorer parameter of the grid search, we used accuracy score, precision, recall and recall on class "deceased". We also set the n_jobs=-1 so that we use all the thread. In the param_grid parameter, we added a parameter dictionary which had all the hyperparamerters and their range that were chosen.
- **RandomizedSearchCV:** RandomizedSearchCV is different from GridSearchCV. It moves within the grid in random fashion to find the best set hyperparameters. We can define a grid of hyperparameter ranges, and randomly sample from the grid using this strategy. This approach reduces unnecessary computation. This strategy is better than GridSearchCV in two ways:
  1) A budget can be chosen independent of the number of parameters and possible values.
  2) Adding parameters that do not influence the performance and does not decrease efficiency.

  As we did in GridSearchCV, we used Stratified K-Folds cross-validator, in which the folds are made by preserving the percentage of samples for each class. In our cross validation, we usually specified five folds. Also we wanted the best model to be the one which has the highest recall on class "deceased", therefore, we refitted the model according to recall on class "deceased". For the scorer parameter of the grid search, we used accuracy score, precision, recall and recall on class "deceased". We also set the n_jobs=-1 so that we use all the thread. In the param_grid parameter, we added a parameter dictionary which had all the hyperparamerters and their range that were chosen. The only difference in parameters was the additional parameter n_iter which defines the number of iterations we want the RandomizedSearchCV to run. In our project we have set this parameter to 30.

Now that we have discussed the strategies used for hyperparameter tuning, lets discuss the hyperparameters used in each model.

A. XGBoost Classifier

We used GridSearchCV as the strategy for hyperparameter tuning of this model. The run time is really long (approx. 5-6 hrs) with this search strategy. The hyperparameters used were min_child_weight, max_depth, n_estimators. We provided a range of values for these hyperparametes. There was no specific reason to choose these parameter, they were chosen at random but learnt a great deal about them. Apart from these hyperparameters, we generalized few parameters such as learning_rate and booster that we wanted to keep same for each model.

The learning_rate is the shrinkage you do at every step you are making and booster which is the type of learner model we want to use. We set the learning_rate=0.01 and booster="gbtree".

B. Random Forest Classifier

We used RandomSearchCV as the strategy for hyperparameter tuning of this model. The run time is much faster (approx. 20-50 min) with this search strategy compared to GridSearchCV. The hyperparameters used were n_estimators, max_depth, max_features and criterion. We provided a range of values for these hyperparametes. There was no specific reason to choose these parameter, they were chosen at random but learnt a great deal about them. Apart from these hyperparameters, we generalized few parameters such as bootstrap, warm_start, and random_state that we wanted to keep same for each model. Bootstrap is a general procedure that can be used to reduce the variance for random forest model, warm_start approach enables us to start training from a better initial point on the loss surface and often learn better models and random_state is basically used for reproducing your problem the same every time it is run [8]. We set the Bootstrap=TRUE, warm_start=TRUE and random_state=0.

C. KNN Classifier

We used RandomSearchCV as the strategy for hyperparameter tuning of this model. The run time is much slower (approx. 2-3 hrs) with this search strategy compared to Random Forest model. The hyperparameters used were n_neighbors, weights, and algorithm. We provided a range of values for these hyperparametes. There was no specific reason to choose these parameter, they were chosen at random but learnt a great deal about them. There were no generalizes parameters set for this model.

VIII. Results

We obtained results for accuracy, precision, f1-score, overall recall, and recall on deceased for each of the three built classifier models. Just a reminder that accuracy gives us the percentage of true classification while precision is the rate of True Positive over Predicted Positive and recall is rate of True Positive over Actual Positive. To better analyse our results, we calculated the harmonic mean of precision and recall (aka. F1-Score). We then calculated the recall on just the class 'deceased' and reported that as well. Moreover, for each model, we sorted the results and reported the top three results for the best accuracy and then best recall on deceased. In most cases, increasing recall on deceased reduced accuracy and vice versa. We suspect that this might be due to the fact that as recall goes up for our dataset, we get more True Positives but we also get higher True Negative, False Positive, and False Negative hence, this results in lower accuracy considering True Negative and False Positive does not affect recall but it does affect accuracy.

A. Random Forest Classifier

Random Forest - Best Results For Recall On Class 'deceased'						
	Hyperparameters	Accuracy	Precision	f1-score	Overaall Recall	Recall on 'deceased'
1	n_estimators=20, max_features='sqrt', max_depth=50, criterion='entropy'	0.8092401026	0.8159618516	0.7652178066	0.8092401026	0.03073620602
2	n_estimators=20, max_features='auto', max_depth=20, criterion='gini'	0.8094765492	0.817018713	0.7652977258	0.8094765492	0.03020640469
3	n_estimators=100, max_features='log2', max_depth=20, criterion='gini'	0.809377299	0.8168109922	0.7652183144	0.809377299	0.0299411527
Random Forest - Best Results For Accuracy						
	Hyperparameters	Accuracy	Precision	f1-score	Overall Recall	Recall on 'deceased'
1	n_estimators=50, max_features='sqrt', max_depth=20, criterion='gini'	0.8095261736	0.8173356891	0.7652454894	0.8095261736	0.02967625204
2	n_estimators=200, max_features='auto', max_depth=20, criterion='entropy'	0.8094969818	0.8173043174	0.7652440479	0.8094969818	0.02914574806
3	n_estimators=100, max_features='auto', max_depth=20, criterion='entropy'	0.8094882243	0.8173724425	0.7652046051	0.8094882243	0.02994080138

TABLE IV
HYPERPARAMETER TUNING: RANDOM FOREST CLASSIFIER

It can be observed that when we set the hyperparameters for RF model to n-estimators=20, max-features='sqrt', max-depth=50, criterion='entropy', we get the best result for recall on deceased. Similarly, setting hyperparameters to n-estimators=20 and 100, max-features='auto' and 'log2', max-depth=20, criterion='gini' we got the second and third best results for recall on deceased respectively. In addition to this, when setting hyperparemeters to n-estimators=50, max-features='sqrt', max-depth=20, criterion='gini', we obtained the best accuracy for our RF model. For second and third most accurate RF model, we set the hyperparameters to n-estimators=200 and 100, max-features='sqrt', max-depth=50, criterion='entropy' respectively.

B. XGBoost

It can be observed that when we set the hyperparameters for XGB model to max-depth=10, min-child-depth=5, and n-estimators=200, we get the best result

XGB - Best Results For Recall On Class 'deceased'						
	Hyperparameters	Accuracy	Precision	f1-score	Overaall Recall	Recall on 'deceased'
1	max_depth=10, min_child_depth=5, n_estimators=200	0.808358531	0.822859913	0.76090395	0.808358531	0.033867235
2	max_depth=10, min_child_depth=0.5, n_estimators=200	0.808507405	0.823142435	0.761109635	0.808507405	0.03386689
3	max_depth=10, min_child_depth=1, n_estimators=20	0.806837685	0.822077947	0.757733512	0.806837685	0.033866545
XGB - Best Results For Accuracy						
	Hyperparameters	Accuracy	Precision	f1-score	Overall Recall	Recall on 'deceased'
1	max_depth=10, min_child_depth=0.5, n_estimators=200	0.808507405	0.823142435	0.761109635	0.808507405	0.03386689
2	max_depth=10, min_child_depth=5, n_estimators=200	0.808358531	0.822859913	0.76090395	0.808358531	0.033867235
3	max_depth=10, min_child_depth=0.5, n_estimators=20	0.806858118	0.82237341	0.757745314	0.806858118	0.033079144

TABLE V
HYPERPARAMETER TUNING: XGBOOST CLASSIFIER

for recall on deceased. Similarly, setting hyperparameters to max-depth=10, min-child-depth=0.5 and 1, and n-estimators=200 and 20, we got the second and third best results for recall on deceased respectively. In addition to this, when setting hyperparemeters to max-depth=10, min-child-depth=0.5, and n-estimators=200, we obtained the best accuracy for our XGB model. For second and third most accurate XGB model, we set the hyperparameters to max-depth=10, min-child-depth=5 and 0.5, and n-estimators=200 and 20, respectively.

C. KNN

It can be observed that when we set the hyperparameters for KNN model to n-neighbors=3, algorithm='brute', and weight='distance', we get the best result for recall on deceased. Similarly, setting hyperparameters to n-neighbors=3 and 5, algorithm='brute', and weight='uniform' and 'distance', we got the second and third best results for recall on deceased respectively.

KNN - Best Results For Recall On Class 'deceased'						
	Hyperparameters	Accuracy	Precision	f1-score	Overaall Recall	Recall on 'deceased'
1	n_neighbors=3, algorithm='brute', weights='distance'	0.750277361	0.751390112	0.747994017	0.750277361	0.043632144
2	n_neighbors=3, algorithm='brute', weights='uniform'	0.750146002	0.751088408	0.747776548	0.750146002	0.040235158
3	n_neighbors=5, algorithm='brute', weights='distance'	0.791889021	0.771654708	0.764322685	0.791889021	0.039189747
KNN - Best Results For Accuracy						
	Hyperparameters	Accuracy	Precision	f1-score	Overall Recall	Recall on 'deceased'
1	n_neighbors=100, algorithm='auto', weights='distance'	0.807698819	0.809391168	0.765524889	0.807698819	0.030047953
2	n_neighbors=50, algorithm='ball_tree', weights='distance'	0.80708289	0.805502859	0.766422039	0.80708289	0.030831584
3	n_neighbors=50, algorithm='brute', weights='distance'	0.806674213	0.802884442	0.767428326	0.806674213	0.030831584

TABLE VI
HYPERPARAMETER TUNING: KNN CLASSIFIER

In addition to this, when setting hyperparemeters to n-neighbors=100, algorithm='auto', and weight='distance' we obtained the best accuracy for our KNN model. For second and third most accurate KNN model, we set the hyperparameters to n-neighbors=50, algorithm='ball-tree' and 'brute', and weight='distance' respectively.

IX. CONCLUSION

The final parameters chosen for each model are those that gave high recall on "deceased". The performance for Random Forest and XGBoost Classifiers increased after hyperparameter tuning whereas the performance for KNN model decreased significantly. For XGBoost, the recall on class "deceased" increased significantly from the baseline results. However the recall on class "deceased" is still very low for all the models. Based on

	Model	Accuracy
1	XGB	81.11%
2	RF	80.79%
3	KNN	73.87%

TABLE VII
FINAL MODEL ACCURACIES

accuracy we don't see overfitting but as we have imbalanced classes in the dataset we will check for the other evaluation metrics. Here we can see that metric values for test are increased in case of XGBoost, approximately same as train in the case of Random Forest Classifier and decreased in the case of KNN. The models trained in this case seem to be a good fit model and neither overfitted nor underfitted. To find out whether the model is overfitted or not, we check the metric value for the test dataset and compare them with evaluation scores of the train dataset. Overall the best model performance is by XGBoost model as its accuracy is highest and recall on call "deceased" is similar to other values. Here are the results obtained:

	precision	recall	f1-score	support
0	0.67	0.03	0.05	1305
1	0.70	0.98	0.82	49244
2	1.00	1.00	1.00	40253
3	0.80	0.17	0.29	23389
accuracy			0.81	114191
macro avg	0.79	0.55	0.54	114191
weighted avg	0.83	0.81	0.76	114191

TABLE VIII
XGBOOST CLASSIFIER TEST RESULTS ON BEST HYPERPARAMETERS

We can see from the above results that, Random Forest and XGBoost have the same accuracy of 81 percent while KNN has a lower accuracy of 74 percent. RF has lower percision but higher recall compared to XGBoost. As expected, KNN has lower percision and recall compared to the other two classifiers. Moreover, RF has a higher

	precision	recall	f1-score	support
0	0.59	0.04	0.08	1340
1	0.70	0.97	0.81	49232
2	1.00	1.00	1.00	39923
3	0.77	0.18	0.30	23696
accuracy			0.81	114191
macro avg	0.76	0.55	0.55	114191
weighted avg	0.82	0.81	0.76	114191

TABLE IX
RANDOM FOREST CLASSIFIER TEST RESULTS ON BEST HYPERPARAMETERS

	precision	recall	f1-score	support
0	0.10	0.05	0.07	1287
1	0.72	0.67	0.70	49272
2	1.00	1.00	1.00	40042
3	0.41	0.48	0.44	23590
accuracy			0.74	114191
macro avg	0.56	0.55	0.55	114191
weighted avg	0.75	0.74	0.74	114191

TABLE X
KNN CLASSIFIER TEST RESULTS ON BEST HYPERPARAMETERS

f1-score than both KNN and XGBoost. Moving on to whether or not we observe overfitting, we can see that for XGBoost the accuracy of the model with testing data is 81 percent and with training data it is 80.8 percent which rounds to 81 percent. For Random Forest, we observe a similar pattern of having accuracy of 81 percent in both training and test data. Furthermore, KNN has a drop of accuracy from 81 to 74 percent when testing data is used instead of training data. So considering that RF and XGB have the same accuracies but RF performs better than compare models to each other choose best fit model

X. FURTHER IMPROVEMENTS

There were certain thing that needed to be taken into consideration we were predicting results. First, the dataset is not perfect as it had a lot of missing data that needed to be treated thorougly. Second, alot of important features like sex were dropped for making predictions as a lot of values were missing from that feature. Third, the imbalanced nature of the classes lead to lower metric scores. Class “deceased” has significantly less data available hence had the lowest recall. To treat the imbalance nature of the dataset, one could oversample the minority classes to make good predictions. There are two techniques one can use:

- **Upsampling or Interpolation**
- **Downsampling or Decimation**

We tried to use SMOTE (Synthetic Minority Oversampling Technique), a type upsampling technique, which increased the minority class significantly but decreased the overall accuracy to 65%. Recall on the minority class increased significantly (closer to 1). We decided not to use this model as it represented a bias. However it could be used to train models with imbalanced datasets.

XI. LESSON LEARNT AND FUTURE WORK

A. Neil Shah

This is not my first machine learning and prediction project. However this project thought me to think outside the normal thinking paradigm. I would never have thought of apply sampling techniques to handle imbalance dataset if I hadn’t learned it in class. I learnt a great deal about different preprocessing techniques to clean the data and remove outliers. I also learned latex to write reports from my teammates. According to me this projects comes with a huge learning curve for new machine learners as it is a hands on exeperience to real life dataset. I learned a lot from my teammates as they were quiet knowledgeable when it came to understanding the dataset. For future work, I am working on my own research in the fields of machine learning and time-series analysis.

B. Murtaza Mushtaq

One of the fundamentals problems I faced was the lack of completeness in the dataset. Many people including full-time data scientist face similar issues everyday. Therefore, learning to build/train/test models on incomplete data by finding interesting/efficient/accurate ways to replace this missing data are skills students can take with them directly to real-world data. Future work would definitely be finding ways to program better/enhancing algorithms so that data imputation can be more accurate which will in most cases result in more accurate models and analysis. Moreover, I had the opportunity to learn about different classifiers such as KNN, RF, XGB, AdaBoost, SVM, etc.

C. Ronit Chawla

Understanding your dataset is a central issue and primary element of progress, cleaning information is one of the main advances in ML. Discovering issues in data collection is significant. Visualizing the dataset can help

to distinguish numerous issues. Does the information contain bunches of NaN values, How to impute them, also if the data needs to be normalized. So, What i realized from this project is data preparation in quite a important task, which cannot be ignored. Also, it takes significant amount of time, which is worthwhile is the longer run. Therefore data understanding and cleaning is critical because. If the data is not clean and properly transformed then the resulting ML model would not be effective at all. For the future work I aim at using this skill at the organization where I work, they have huge data sets which they have been collecting since years. It would be quite interesting to apply the learning's from this course to their stored data and explore some insights and hidden trends from it.

XII. Contribution

A. Neil Shah

- Milestone 1: Data cleaning and imputation, outlier detection, transformation and data joining
- Milestone 2: Developing models, tuning baseline models, report and analysis of results
- Milestone 3: report, model hyperparameter tuning, looping over data preprocessing, trying oversampling data, trying other techniques to improve model results.

B. Murtaza Mushtaq

- Milestone 1: Data cleaning and imputaion, outliers, report
- Milestone 2: Dataset split, classifier modelling and evaluation metrics
- Milestone 3: Data preprocessing, classification models, initial evaluation and overfitting

C. Ronit Chawla

- Milestone 1: Dataset description and ERD, outliers, transformation, report
- Milestone 2: Dataset split, built classifier model and evaluation metrics, overfitting
- Milestone 3: classification models, initial evaluation and overfitting, hyperparameter tuning and report

References

[1] A. Navlani. (), [Online]. Available: https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn. (accessed: 12.14.2020).

[2] J. Brownlee. (), [Online]. Available: https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/. (accessed: 12.14.2020).

[3] S. Patel. (), [Online]. Available: https://medium.com/machine-learning-101/chapter-5-random-forest-classifier-56dc7425c3e1. (accessed: 12.14.2020).

[4] V. Singh. (), [Online]. Available: https://blog.quantinsti.com/machine-learning-k-nearest-neighbors-knn-algorithm-python/. (accessed: 12.14.2020).

[5] J. Jordan. (), [Online]. Available: https://www.jeremyjordan.me/hyperparameter-tuning/. (accessed: 12.15.2020).

[6] GeeksForGeeks. (), [Online]. Available: https://www.geeksforgeeks.org/hyperparameter-tuning/. (accessed: 12.15.2020).

[7] Sciket. (), [Online]. Available: https://scikit-learn.org/stable/modules/grid_search.html. (accessed: 12.15.2020).

[8] ——, (), [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. (accessed: 12.15.2020).