CMPT 459 Milestone 3: Predicting COVID-19 Case Outcomes

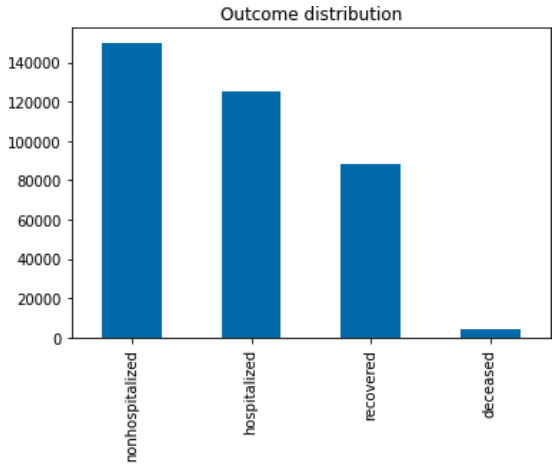
Problem Statement

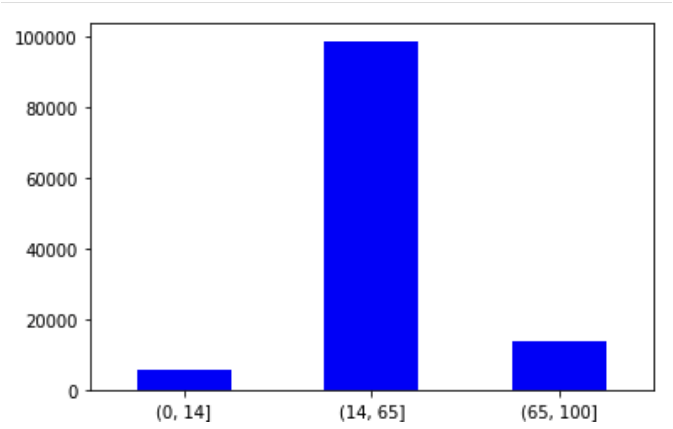
In this project, we are given datasets containing various information on specific COVID-19 cases, and COVID-19 data over geographic regions.
The goal of this project is to train a machine learning model that is able to correctly predict the outcome of a COVID-19 case. Most importantly, we want to be able to correctly predict the ‘deceased’ outcome. This is because in a health care setting, false negatives for a case which is likely to end up as a ‘deceased’ outcome could lead to a patient who is on the brink of death not getting adequate treatment. Ultimately, mispredicting a ‘deceased’ outcome could lead to unnecessary loss of life, and so it is essential that our model not have too many false negatives for ‘deceased’. However, predicting too many false positives for ‘deceased’ could also have disastrous consequences, since overpredicting which cases are likely to end up fatal could overwhelm ICU beds and critical care capacity. Thus, a fine balance between false positives and false negatives on the ‘deceased’ outcome is paramount for this task.

Dataset description and EDA

To get a better understanding of the datasets, we have performed analysis for all features.

Outcome
The outcome distribution shows that deceased cases account for a very small portion of the dataset compared to the others. This will make a lot more challenging to train the model to perform well on predicting deceased cases, which is our main goal.



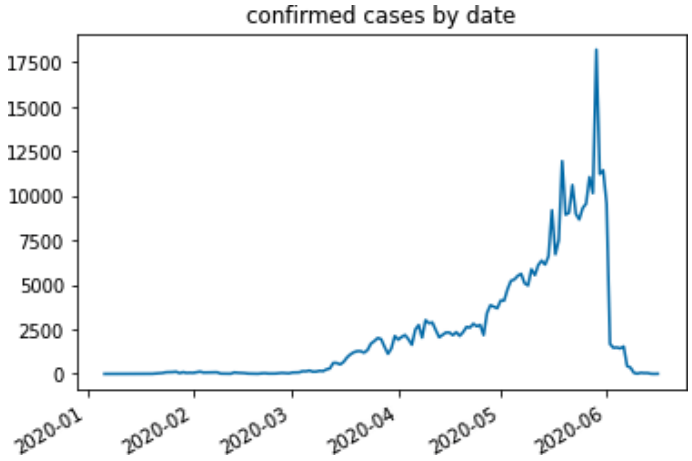


Age Distribution

We found that a lot of the data on age columns are missing. Since COVID mortality rate is higher among elderly, for our demonstration we have divided the age group into 0-14, 14-65, 65-100. The plot shows the majority of rows fall between the 14-65 age range.

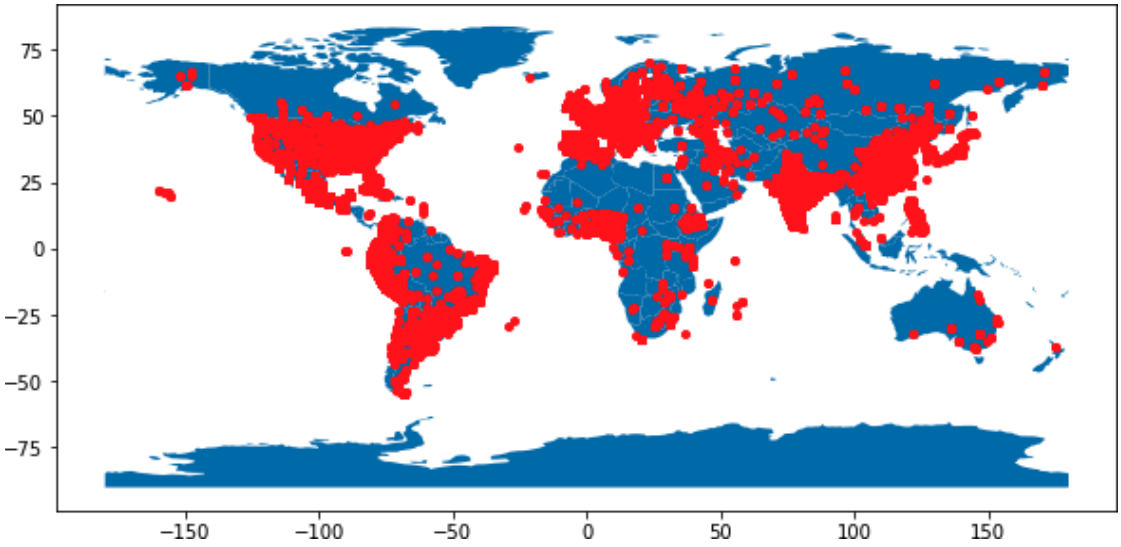
Date Confirmation

To understand the dataset in it's chronological order, we plot the number of daily cases by date and we can see that the trend matches the timeline of COVID-19 pandemic outbreak.



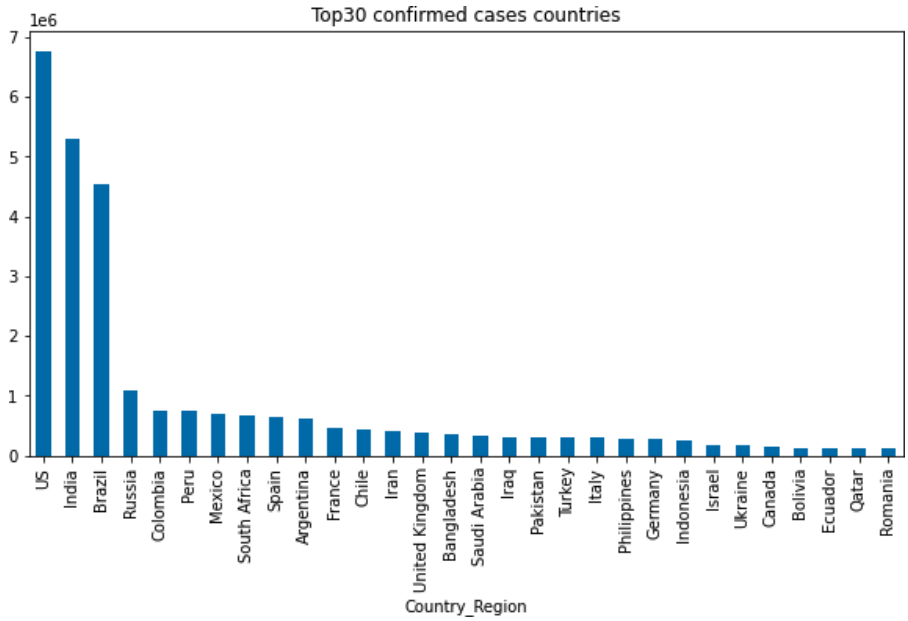
Longitude, Latitude

The coordinates are combined and plot on the world map. Each red dot represents a confirmed case and it shows the geographical distribution of the data.



Countries

From the case count per country in the figure below, we can see that the number of cases in US, India and Brazil are much higher than other countries



Data preparation

After analysis we notice that some of the columns have either missing data or corrupted data that will require some data cleaning.

Data Cleaning:

- if age is in months, convert months to years (or 1 if age < 12 months)
- if age is a range, take the average of the range
- if age is x+, convert age to x (maybe change to average lifespan - x /2)
- **Sex:** impute half missing values as female, half as male (expect a roughly 50:50 ratio of genders so this is reasonable imputation strategy)
- **Country: i**f country missing, impute it using latitude/longitude
- drop rows where country, latitude and longitude are all missing (very rare)
- There were also several inputs in the date_confirmation column that specified a range of dates. The first date presented in these cases was assumed to be the correct date and the data frame was adjusted in accordance to this.

Outlier Detection:

- First outlier that we can find is in the ‘confirmed cases by date plot’, we can see a sudden decline in cases around June 2020, which is unrealistic. What we need to do is to filter out cases that are confirmed after June 2020.
- Some regions have high Case-Fatality Ratio due to the fact that there are a small number of cases in the region but some death. We will calculate the mean and standard deviation of all regions. The outliers would be identified as more than 3

standard deviations away than the mean. And we simply remove them from the merged dataframe.

Merging Datasets:

- We decided to use the combination of country and province as a feature for training our models. In order to join the location and train/test datasets, we synthetically construct a new combined key feature using province/country that corresponds to the combined key in the location dataset. For some cases in test/train however, the location dataset does not contain statistics for the specific province of that case. In such situations, we created a new combined key by joining the string 'Unknown' with the case's country. We then imputed the values for this combined province-country key using the mean values from the location dataset for all the other provinces/states in this country. **Note**, we only created this 'Unknown' region if the country for the case had at least one other 'Province_State' value in the location.csv dataset. Otherwise the combined key for that case consisted only of the country, and there was no need for imputation.
- In any other situation where some value was missing for the location.csv dataset, we would impute the missing value using the means from the same country, and if the country key was a unique value, we would impute the value using the means of all the other rows in the dataset.
- We further split our training dataset into a training set and a test (validation) set with an 80:20 ratio. This was to ensure we were not overfitting during hyperparameter tuning.

Classification models:

For the project we used the XGBoost, Decision Tree, and Random Forest classifiers.

We picked the random forest algorithm because it is robust to outliers due to being an ensemble method and has a lower risk of overfitting data when compared to decision trees. As aresult the random forest model should in theory have low variance compared to other models and thus perform well on unseen data. Since our ultimate goal of the project isto have a high recall rate for the 'deceased' outcome on an unseen dataset, this robustness to variance in data is a desirable quality.

We decided to use decision trees as one of our models to reduce complexity when dealing with a dataset with a mix of continuous and categorical data. Unlike models such as SVM or KNN which use a distance function and require scaling, decision trees allow us to eliminate this complexity and run the model with limited preprocessing.

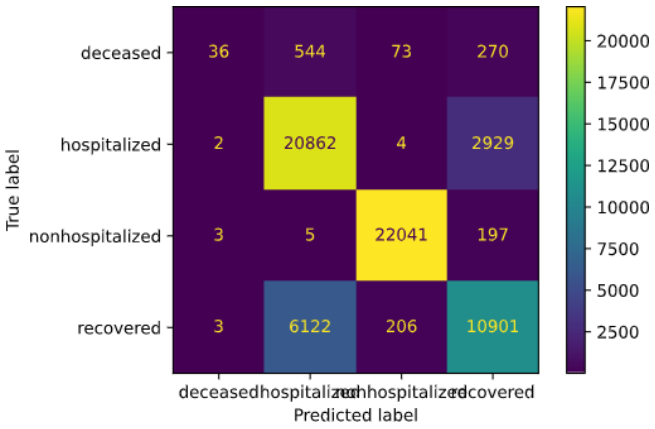
For the boosting tree algorithm we chose XGboost. XGBoost has the advantage of reducing both bias and variance, by using gradient boosting, where classification errors are used to correct for bias in each round of boosting.

In addition, XGboost has more hyperparameters you can tune than other boosting methods, so it’s more flexible. XGboost also provides an option to do regularization to reduce overfitting. However, it does come at the cost that results are highly dependent on proper hyperparameter tuning.

Initial evaluation and overfitting:

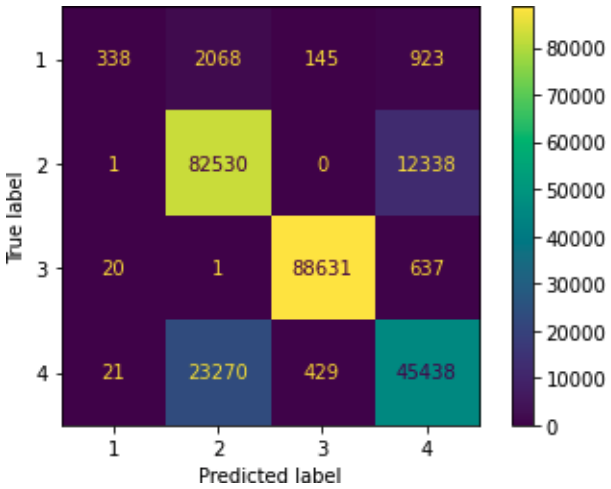
Random Forest:

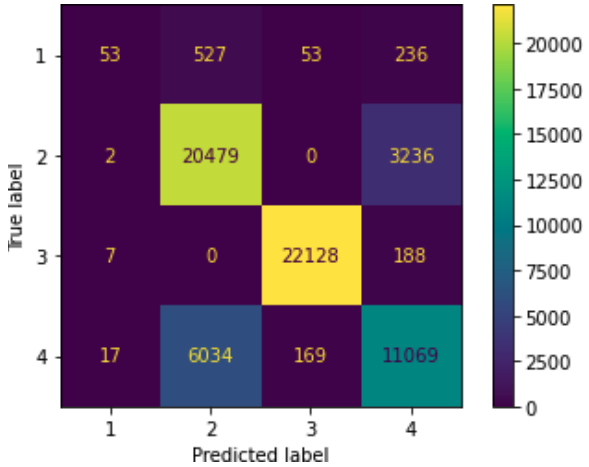
During initial training without any hyperparameter tuning, the confusion matrix produced for random forest on the validation set is shown below. As we can see there is a dismal false negative rate for the ‘deceased’ class label. This is due to overfitting on the other class labels, since Random Forest can achieve a high accuracy by just ignoring the ‘deceased’ class, which is not what we want at all as it contradicts the purpose of the project.Thus the focus in the hyperparameter/model tuning should be to improve the f1 score on ‘deceased’.



XGBoost:

During initial training without any hyperparameter tuning we noticed that in the confusion matrix, the model did a much better job of predicting hospitalized and nonhospitalized cases than deceased/ recovered cases. Specifically on deceased cases with recall of 35.9% and precision of 68%



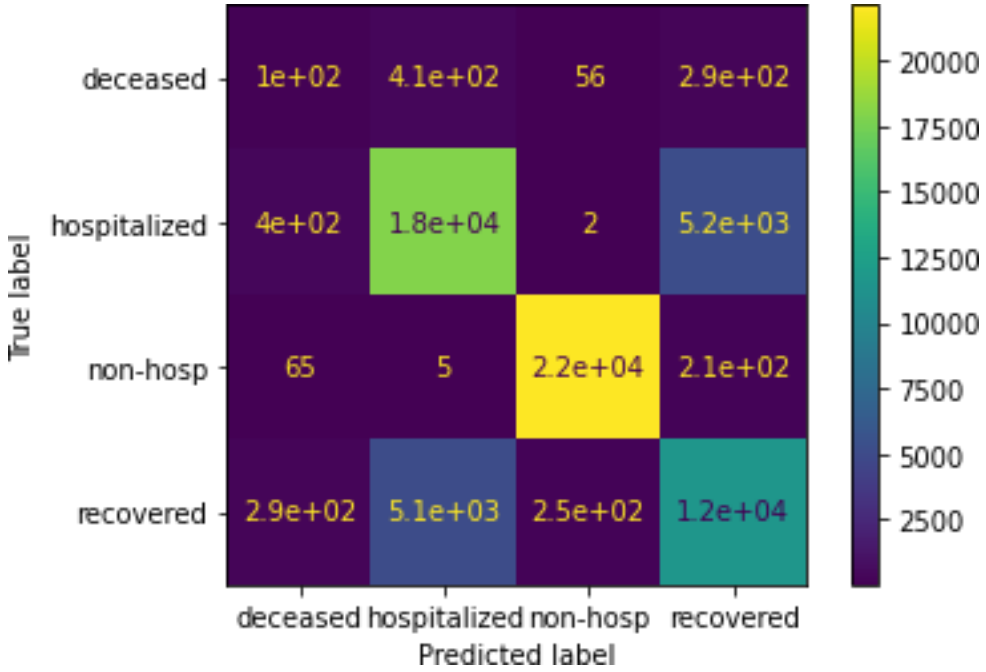


Part of the problem is that the class labels of our dataset are highly imbalanced, where the majority of the data with outcome either hospitalized or nonhospitalized. We can see this from the value counts on the outcome column of the dataset:

hospitalized	non-hospitalized	recovered	deceased
118586	111612	86447	4343

To address this issue we use the **sample_weight** parameter to improve our model training, which assigns class_weight for every instance by computing an array. We can see significant improvement on recall (averaging over 70%) in the next section after **sample_weight** is applied.

Decision Tree



Above is the confusion matrix for the decision tree without tuning. There is understandable overlap between hospitalized and recovered, however the most glaring error is in the deceased label. This because there are far less deseased cases than any other one, so the model overfits to the other, more popular, labels. We will do further discussion on decision tree overfitting in the next section.

Hyperparameter tuning:

Random Forest:

For addressing the massive class imbalance and our priority of correctly classifying the minority label, ‘deceased’, we combined the two approaches discussed in the paper by

Chen et al. (2004). These strategies are weighted random forest and balanced random forest. In balanced Random Forest, we construct a new dataset from our original training data which contains a nearly balanced proportion of the different outcomes. This is achieved by randomly oversampling with replacement from the minority 'deceased' class. Rather than naively duplicating these 'deceased' data points however, we used a library called imbalanced learn which leverages a more sophisticated approach known as synthetic minority oversampling technique (SMOTE). This approach creates synthetic data points using the distribution of features in the minority class, with the goal of alleviating the overfitting introduced by duplicating data points. The resulting distribution after applying SMOTE to the training data can be seen in 'balance.png'. Weighted random forest assigns a weight to each class, which is inversely proportional to the frequency of that class label. Thus, classes that occur more frequently are 'penalized' by the weighting system, and inversely classes that occur infrequently are amplified by the weights. We used a further improvement suggested by this blog to weight the classes for each bootstrap subsample, as opposed to the overall dataset. This is helpful because random forest uses multiple subsamples for training different trees in the ensemble, and so it is more precise to use the class distribution of the bootstrap sample which could vary drastically from the overall dataset when computing class weights.

We tested both these methods separately and together using GridSearchCV and found that the optimal f1 score on ‘deceased’ on the validation dataset was achieved using a combination of the two strategies.

The technique used for tuning the **max_depth**, **max_features** and **n_estimators** hyperparameters (number of trees in the ensemble) was to use GridSearchCV with a refit scoring parameter of f1_weighted. F1_weighted was selected as the refit performance metric due to the fact that the goal of the project is improving the f1_score on ‘deceased’. We chose the weighted average for f1_score as opposed to the f1_score on ‘deceased’ after empirically testing out the two and finding that weighted f1_score performed better on the validation set. This is likely due to overfitting that occurs on the training dataset when f1_score on ‘deceased’ is used as opposed to the overall average f1_score.

XGBoost:

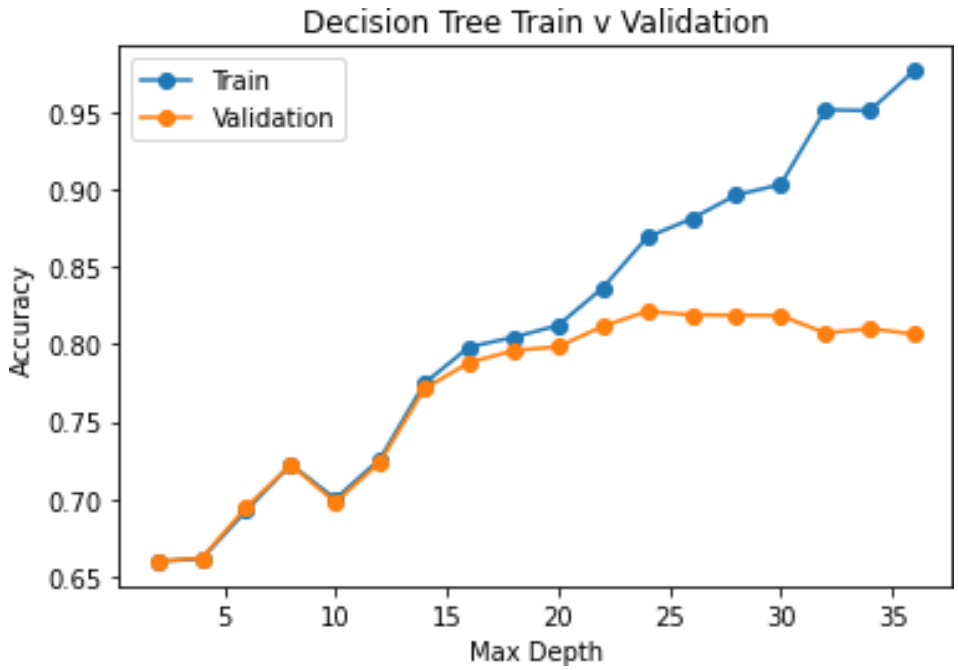
XGBoost has many hyperparameters to tune such as **max_depth, learning_rate, gamma, reg_lambda.** Due to the long training time, we leave out the **reg_lambda** as **0,** which is an L2 regularization term on weights. Increasing this value will make models more conservative. **GridSearchCV** is used for our hyperparameter tuning because it will exhaust all combinations from the different parameters provided. The metrics that we choose to evaluate the performance of each set of hyperparameters is **f1_score,** which gives us a better measurement on how well the model is performing overall. In addition, we use 3-fold cross validation to get better results. We picked 3 options for each parameter and see if the middle value gives the best results.

For example, we picked [3, 4, 5] for max_depth in the round 1. If the best result is given by 3 or 5, then we explore further. If the best result is given by 4, we can consider it as the optimal parameter for max_depth.

Final round hyperparameter combination results are recorded in **xgboost_tuning.csv** file in the results folder.

Decision Tree

The most important hyperparameter to tune when using a decision tree is the max_depth. This is because of decision trees’ feature of strongly overfitting to the training data. By minimizing the max_depth, we limit the tree’s ability to overfit.



As you can see from the diagram above, the optimal max_depth lies around 25. However, as you can see from **DecisionTree_tuning.csv** the optimal F1 score is found when max_depth = 27 not 25. Because, as stated in the description file, our goal is to tune our models with respect to the F1 score, we decided that we would set max_depth = 27 for our optimal decision tree model.

We also chose to tune the hyperparameters max_features and criterion as we believed that they would have the biggest impact on the quality of our model.

The optimal tunings were found by running CVSearch with a cross-validation = 3 using a variety of variables for each hyperparameter.

Results

The full reports for the hyper parameters that were used to tune XGBoost, Random Forest and Decision Tree are recorded in 'xgboost_tuning.csv', 'DecisionTree_tuning.csv', and 'randomforest_tuning.csv'.

For XGboost model, the hyperparameters that gives best F1_score on 'deceased' are:

gamma	learning_rate	max_depth	reg_lambda

0	0.1	7	0

The model trained was able to achieve the following results on the training dataset:

F1_deceased	Recall_deceased	Accuracy	Recall
0.1081	0.0591	0.7338	0.7338

For Random Forest model, the hyperparameters that gives best F1_score on 'deceased' are:

n_estimators	max_depth	max_features
100	30	sqrt

The model was able to achieve the following results on the **testing/validation** dataset:

F1_deceased	Recall_deceased	Accuracy	Recall
0.1306	0.0992	0.7988	0.7988

For the Decision Tree the hyperparameters that give the best F1_score for the 'deceased' label were:

criterion	max_depth	max_features
'gini'	27	sqrt

On the validation dataset, the following results were observed:

F1_deceased	Recall_deceased	Accuracy	Recall
0.0757	0.1413	0.7993	0.7993

Conclusion

As previously mentioned, the highest f1 score for Random Forest achieved on the 'deceased' label using a validation set was 0.1306. The highest recall on 'deceased' and it's overall weighted average was Decision Tree with 0.1413. The highest f1 score achieved on 'deceased' using XGBoost on the *training* dataset was 0.1081.. The most accurate model was Decision Tree with an accuracy of 79.93%. In the end, we picked Random Forest dueto its higher performing f1 score on 'deceased'. Since f1 score is a blended metric that balances false positive and false negative frequencies, looking at the confusion matrix can show us how well we are performing on classifying 'deceased' labels. As we can

see from the figure below, we are still not performing very well on the validation set on identifying ‘deceased’ outcomes. It is comforting to note two things however: from the confusion matrix in the previous section showing Random Forest predictions prior to tuning, we see that our false positive/negative count on ‘deceased’ has been reduced significantly. Furthermore, at least most of the false positives/negatives for the ‘deceased’ class are being misclassified as the ‘hospitalized’ class, which is the second most severe outcome of a covid case. Thus the model seems to be sensitive to the severity of covid cases, which is also important.

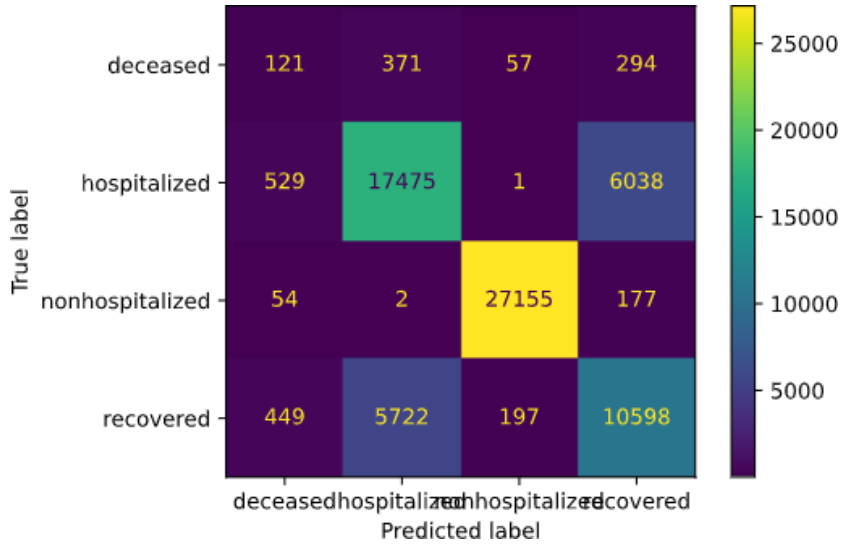


Figure: Random Forest Confusion Matrix on Test Data Post Tuning

Finally, if we look at the confusion matrix on the testing dataset, we see how the oversampling strategy has led to significant overfitting on the ‘deceased’ class. Unfortunately this seems to be a drawback of SMOTE which we must pay in exchange for a lowered false negative count on the minority class. It also means that the f1 score for ‘deceased’ on the training dataset is not a good metric for Random Forest that has been trained using SMOTE, which is why we reported the values using the validation set instead.

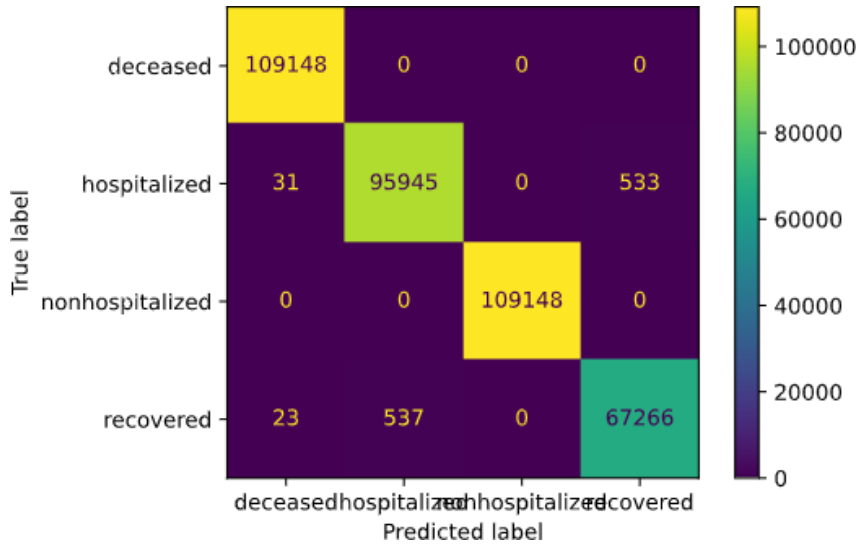


Figure: Random Forest Confusion Matrix on Training Data Post Tuning

Prediction on test dataset

To predict the outcome labels on the test dataset, we selected the model with the highest f1 score on deceased, Random Forest, and the optimal parameters from that model that were determined by GridSearch. We trained one last Random Forest model using the entire train dataset to minimize variance as much as possible. Finally we called the predict function on the test dataset to generate the outcome label predictions.

Lessons learnt and future work

The most important lesson learned from this project was that accuracy is not always the most valuable metric. Rather, one needs an understanding of the purpose behind the classification task in order to choose the optimal metric, of which there are many. In our case f1 score on 'deceased' was the most appropriate metric, since we care the most about correctly classifying 'deceased' outcomes, as that is the most critical outcome in a healthcare setting. Another key lesson learnt during the project was that classification on imbalanced datasets using supervised learning models is a difficult task. This is due to the difficulty in correctly predicting when an unlikely/infrequent minority outcome may occur. Even after applying the suggested techniques from the current data science literature of random oversampling and class weighting, we were still not able to achieve a great f1 score for the 'deceased' outcome. Some further suggestions for improving our recall/f1 score on the 'deceased' label would be to explore other machine learning models, such as a Gaussian Process Classifier. Finally, some in the data science community have mentioned that generative models may outperform discriminative models on moderately sized datasets. Though the evidence for this is limited, there are some recent papers discussing the topic, and it would be worth investigating after other options have been exhausted

References

Using Random Forest to Learn Imbalanced Data (using rf on imbalance data)
https://stats.stackexchange.com/questions/274807/how-to-improve-f1-score-with-skewed-classes improve f1 score with skewed data
https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/ oversampling with rf
https://stats.stackexchange.com/questions/340854/random-forest-for-imbalanced-data generative models
https://openaccess.thecvf.com/content_CVPR_2020/papers/Wang_Deep_Generative_Model_for_Robust_Imbalance_Classification_CVPR_2020_paper.pdf generative models paper
https://machinelearningmastery.com/bagging-and-random-forest-for-imbalanced-classification/ subsample weighting
https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/ SMOTE