CMPT 459 - Course Project Report

Problem Statement

Predict the outcome group (i.e., hospitalized, nonhospitalized, deceased, recovered) of individuals that test positive for COVID. Perform any necessary data preprocessing and feature engineering. Various classification models will be trained and tested on their performance by comparison of their F1-score. We will conduct hyperparameter tuning and cross validation to improve their performance.

Data Preparation

1.1 Feature Selection

Through training and performance analysis of our models, we chose ***age***, ***country***, ***chronic disease binary***, ***deaths***, ***active***, and ***case fatality ratio*** from the original set of features as the most impactful. Choosing these features resulted in better performance on our models to achieve the desired mean macro F1 scores. To improve the performance even further, we derived the ***month*** feature from date. These features and their importance was also tested later on during our model selection process.

1.2 Mapping Features

Categorical features and the outcome group were mapped to numeric values using `pandas.Categorical()`. This method encodes categories by taking a look at the dtype of the column/feature and makes a decision in a lexicographic order, which follows the mapping requirement of the **outcome groups**.

1.3 Balancing Classes

We have chosen to balance the classes using RandomOverSampler from the imblearn library. The RandomOverSampler is used to over-sample the minority classes by selecting samples with replacement. This is in contrast to the RandomUnderSampler which randomly removes examples of the majority class. We chose to over sample due to the disproportionately low number of the minority outcome groups and the limited data in the dataset. To under-sample such that each predicted class (hospitalized, nonhospitalized, and deceased) have the same number of observations as the minority class would be to severely reduce the number of observations used to train each model. Over-sampling can cause models to overfit to training data, however, we use a validation set and plot evaluation metrics to check for generalization errors.

We used Stratified K-Fold Cross-Validation which makes sure we have a proportional amount of each class in each k folds. In our case, because we applied over-sampling before Stratified K-fold Cross-Validation, we will have exactly the same number of observations for all three outcome classes.

Classification Models

1.4 Building Models and Hyperparameter Tuning

Generally, we fit different models with default hyperparameters on data that was a simple train-test split in a ratio of 80:20. This allowed us to get a *sense* of how good each model is

without any hyperparameter tuning. Generally, selecting optimal hyperparameters improved the model scores; however, the improvement was relatively minor. So, an algorithm that performed poorly—for example perceptrons, naive bayes, and KNN—were found to not reach scores of even the default well-performing algorithms. Such well-performing algorithms were found to be ensembles, namely Random Forest, Extra-Trees, and AdaBoost.

The metrics we were optimizing were the macro F1-scores, the F1-score for the “deceased” category, and the overall accuracy. Due to the skew of the data, a model could achieve a high accuracy by simply predicting the majority class. So, we focused mostly on the macro and “deceased” F1 scores for determining the “best” model. With our metric of success clearly defined, we saw Random Forest, Extra-Trees, and AdaBoost perform the best on the train-test split data. To further improve the models, we tuned hyperparameters and performed k-fold cross validation.

We used k-fold cross validation with 10 folds. We tested our models incrementing k from 2 to 20 and found that after around 10 folds the score metrics stopped improving significantly. Because cross validation is often used with 5 or 10 folds and increasing k greater than 10 did not result in significantly better scores, we used 10-fold cross validation.

To tune hyperparameters we started with GridSearchCV; however, we quickly realized that for many combinations of hyperparameters, grid search is extremely slow. So, we used RandomSearchCV to randomly try hyperparameter combinations and settle on the best known combination. The random forest and extra-trees classifiers had many identical hyperparameters and therefore were tested with the same ranges. The hyperparameters tuned and their respective ranges are as follows.

- criterion: [gini, entropy]
- max_depth: [2, 8, 21, 100, None]
- max_features: ['sqrt', 2, 5, None]

The optimal combination of features found for the random forest classifier were criterion=entropy, max_depth=100, and max_features=5. The optimal combination for extra-trees was criterion=gini, max_depth=None, and max_features=5. These combinations resulted in high scores; however, the best combination found across all the experimentation done—in and out of automated hyperparameter searching—were found to be criterion=entropy, max_depth=7, and max_features=sqrt. So, these hyperparameters were used for our final models.

AdaBoost has much fewer hyperparameters than the tree-based models. We determined through experimenting with different hyperparameter tunings that the learning_rate had the greatest impact on our scores. So, we tested AdaBoost models with learning rates 0.0001, 0.001, 0.01, 0.1, 0.5, and 1.0 where the best learning rate was found to be 0.1 as reflected below.

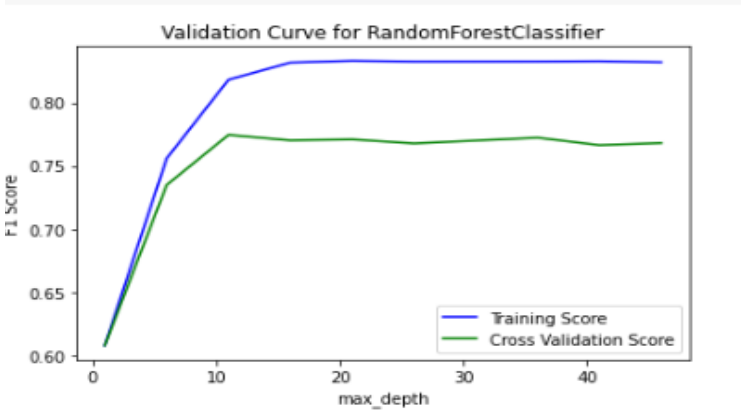
Model	Hyperparameters	Mean macro F1-score across the validation sets	Mean F1-score on 'deceased' across the validation sets	Mean overall accuracy across the validation sets
Random Forest	criterion=entropy, max_depth=7, max_features=sqrt, All else default.	0.7845	0.5271	0.9289
Extra-Trees	criterion=entropy, max_depth=7, max_features=sqrt All else default.	0.7803	0.4987	0.9339
AdaBoost	learning_rate=0.1 All else default.	0.7504	0.4726	0.9133

1.5 Overfitting

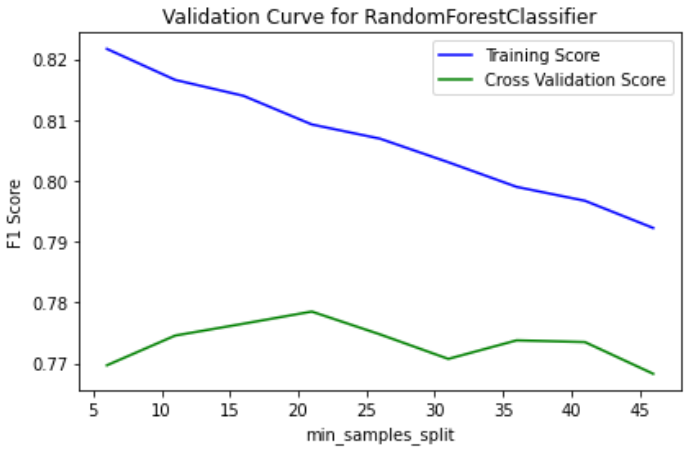
A simple, yet effective way to check for overfitting is to plot the validation curve of a model's performance using sklearn.model_selection's validation_curve feature. This validation curve can be used to measure the influence of a single hyperparameter.

For **RandomForest** ;

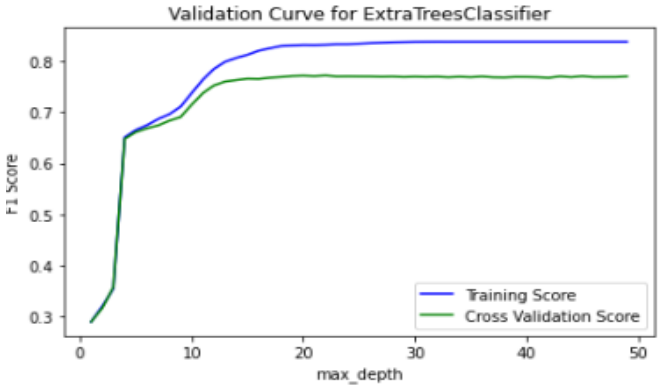
- varying **max_depths** on shows diminishing returns at max_depths > 11. The training increasing and deviating from the validation score is a result of overfitting.



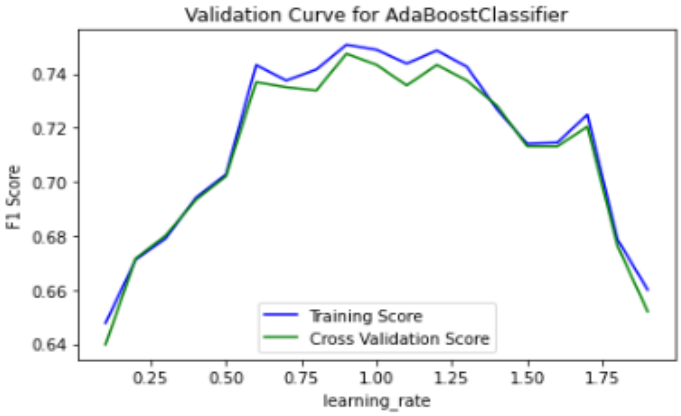
- The plot for varying **min_sample_splits** shows training scores decreasing while cross-validation score stays stagnant. Increasing this hyperparameter's value results in underfitting.



For **ExtraTrees** ; after getting the score of the model from changing the **max_depth** parameter we were able to see overfitting after max_depth >= 10 as the scores start saturating around max_depth = 20



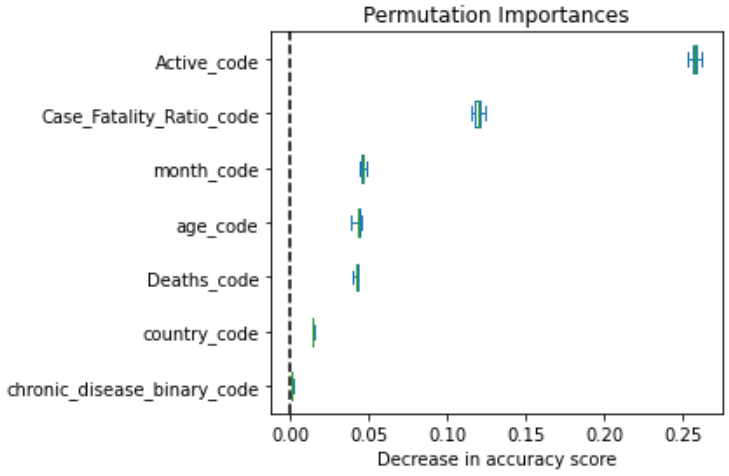
For **AdaBoost ;** altering the **learning_rate** for the Adaboost classifier the score of the algorithm results in a maxima being formed in the range of values for learning_rate = 0.65 to 1 and then learning_rate > 1.0 drastically decreases the score.

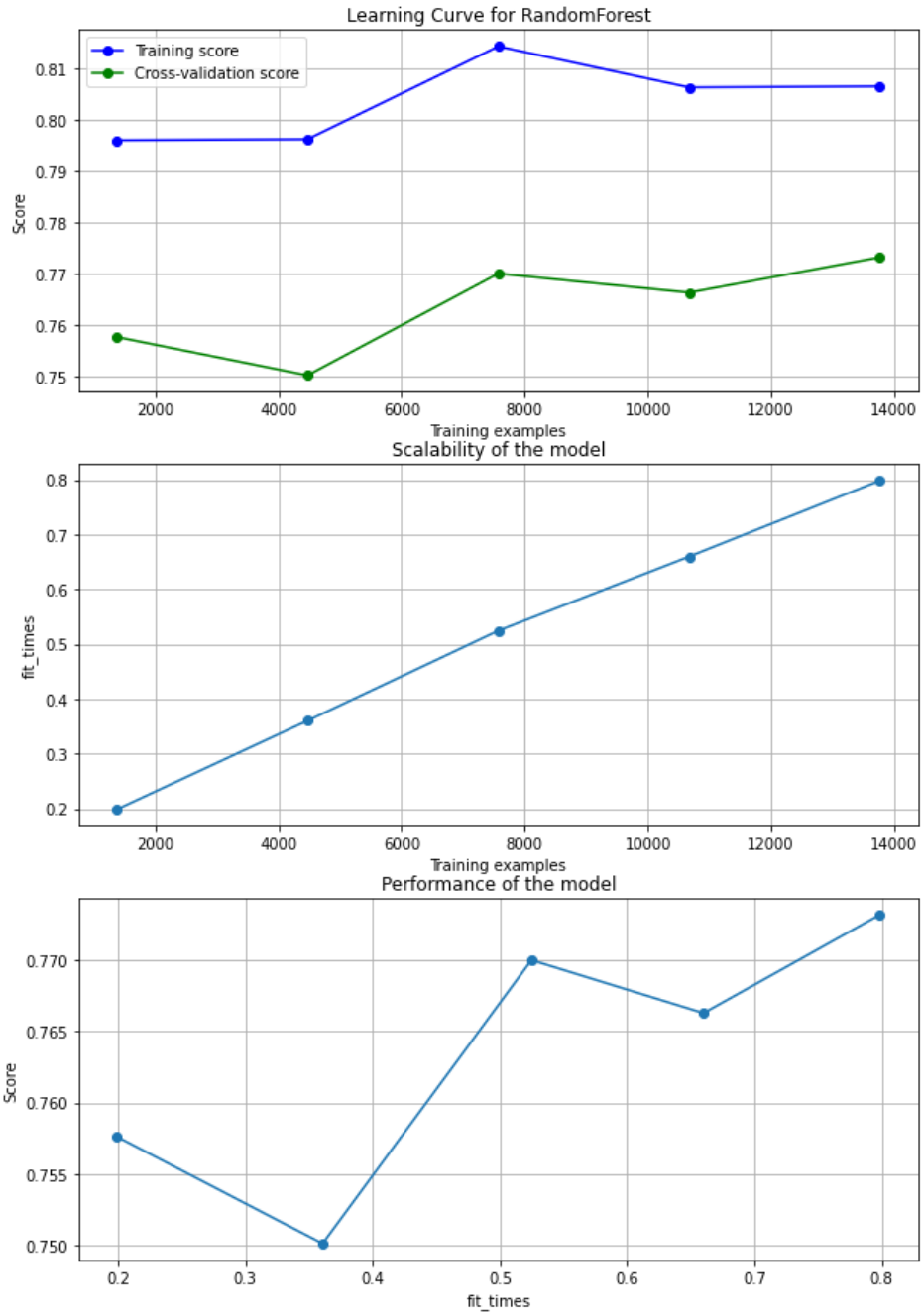


1.6 Comparative Study

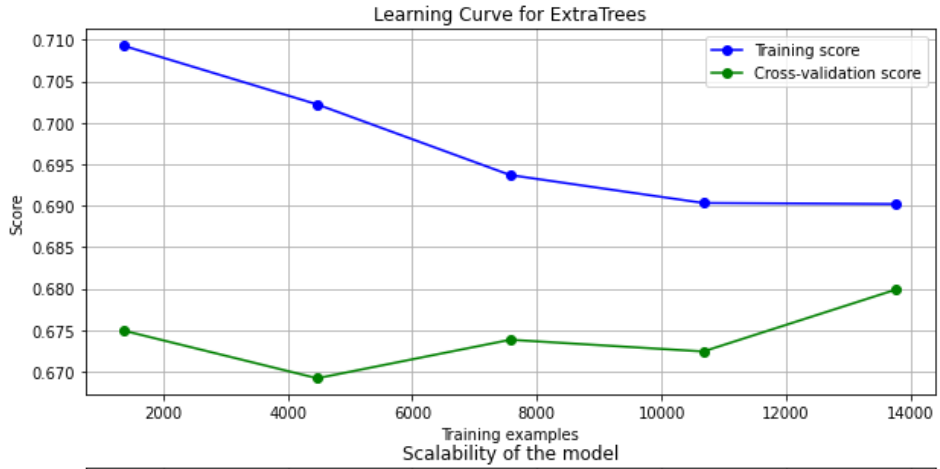
Machine learning models can vary by their learning ability, scalability, and fit times over incremental sizes of training sets. Utilizing learning curves and plotting the results can be beneficial in diagnosing the learning and generalization performance over an increasing training set. To compare the models, we plotted (1) the training and cross validation scores over the number of training examples, (2) the fit time of the model over training examples, and (3) fit times relative to how well they scored. This gave us a general overview on their learning ability and performance. From our results, we found that our RandomForest model is the best overall performer. The advantages of RF are its hyperparameter tuning capabilities coupled with their interoperability and its generalization accuracy for larger datasets. The disadvantage is evident from being the poorest performer in scalability due to the resources required to build numerous trees and voting of the combined outputs.

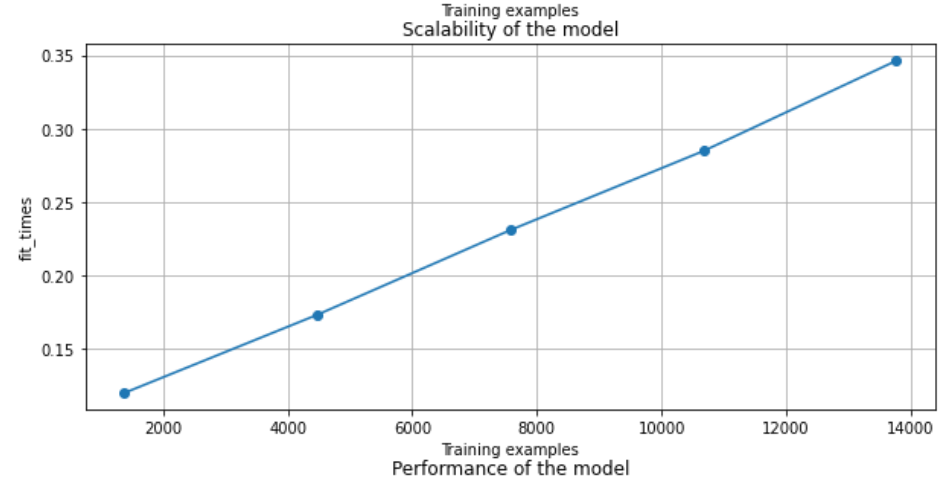
Another important aspect of selecting the best model is its robustness and accuracy to back the real-world bias, so , for the Random Forest we used the `permutation_importance function` from `sklearn.inspection` because feature importance shall not be a part of model selection rather it should be a method to make claims about our selected model. The permutation feature importance is defined to be the decrease in a model score when a single feature value is randomly shuffled. This procedure breaks the relationship between the feature and the target, thus the drop in the model score is indicative of how much the model depends on the feature.Using this the Most Important Features will be number of active cases in the area (province/country) followed by fatality ratio() which can be seen as a measure of how deadly a variant pertaining to a particular area is.
Month is also a major factor because of how much that particular variant spreads during a time period. Thus our choice to pick RandomForest is strongly backed.

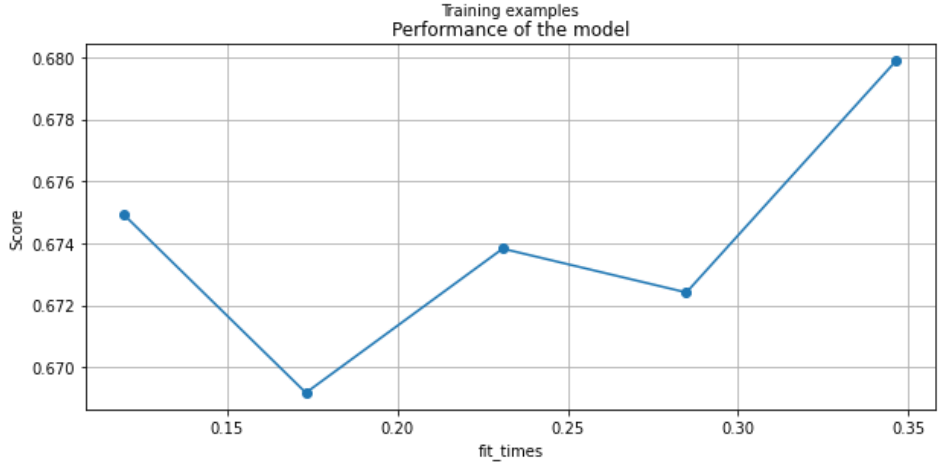




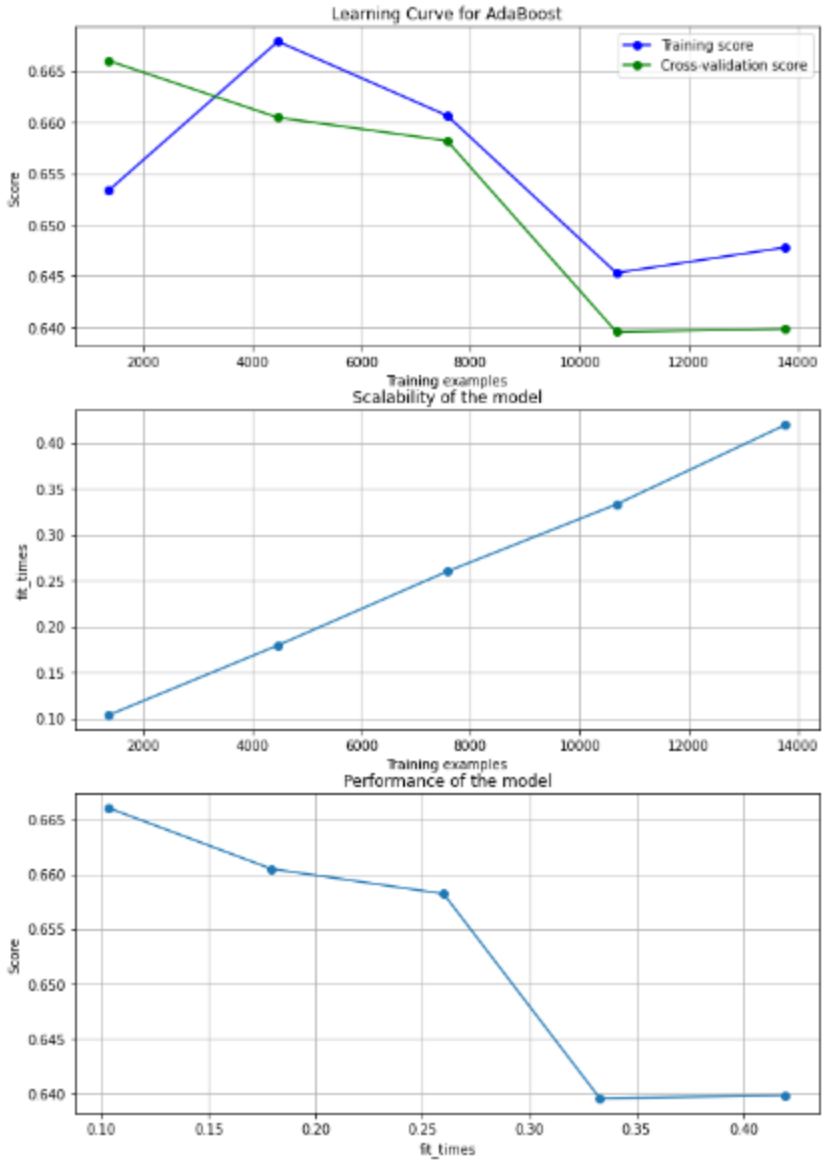
From the first plot, we can see that our Random Forest model is adept at learning from a smaller sample size with impressive training and cross validation scored on the F1 metric. Both scores increase over incremental training sizes and at our given maximum dataset size, we can see that the model can possibly improve with an even larger dataset. The scalability of this model is not as impressive as there is an increased fit time complexity as the dataset increases, however, the last plot indicates that the increased fit time complexity is correlated to yielding better results.







Compared to our RandomForest model, ExtraTrees plot 1 shows that the training and cross-validation score come close to convergence towards the given maximum size of the dataset. ExtraTrees fit times are lower than our RandomForest, however, still suffers from scaling to larger sets.



For Adaboost we clearly see that the model performs very badly with an increase in the number of the training samples as compared to the other two models so this model was directly eliminated from the process of choosing the best model straightaway.

Concluding from all these findings RandomForest is the best choice to move forward.

Model Predictions

Random Forest (Chosen)

```
Mean macro f1 (must be >= 70) 0.782550329457812
Mean deceased f1 (must be >= 40) 0.5195819526950279
Mean accuracy 0.9267377230508422
```

ExtraTrees

```
Mean macro f1 (must be >= 70) 0.7789893623544122
Mean deceased f1 (must be >= 40) 0.4965888076482627
Mean accuracy 0.9335928183719455
```

AdaBoost

```
Mean macro f1 (must be >= 70) 0.750006796425628
Mean deceased f1 (must be >= 40) 0.4699087870256179
Mean accuracy 0.9126771095053856
```

Conclusion

The conclusion from this process is that RandomForest model is the best overall performer as the goal is to predict the outcome (hospitalized, nonhospitalized, deceased) of a case correctly with a small number of false negatives and false positives and RandomForest after our hyperparameter tuning gives the best macro-F1 score.

Lessons Learnt and Future Work

- Doing Exploratory analysis beforehand results in a clearer pathway to build a predictive model.
- Feature extraction and feature importance calculation are 2 completely different things; Feature extraction shall always be done before or during choosing the best model while feature importance calculation shall always be done for the selected model.
- Stratified sampling should mostly be preferred to random sampling.
- Only testing set performance of the model should be considered while model selection.

- Overfitting should be negated by looking at the difference in train and test scores after hyperparameter tuning.
- Mapping data of categorical type to categorical codes helps in making critical decisions and computations on it take less time.

References

https://scikit-learn.org/stable/modules/learning_curve.html#learning-curve
https://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html#sphx-glr-auto-examples-model-selection-plot-learning-curve-py

Contribution

Jason Huang:

- Milestone 1 :
  - 1.1/1.2 Cleaning messy outcome labels
  - 1.4 Data cleaning and imputation
  - 1.5 Histogram portion for finding outliers
- Milestone 2 :
  - 1.2 Mapping categorical features to numerical
  - 1.5 Plots for hyperparameter affecting overfit and underfit of default models
  - 1.6 Plots for performance analysis and comparative study

Alex Blackwell:

- Milestone 1:
  - 1.6 Joining the cases and location datasets
  - 1.7 Feature selection
- Milestone 2:
  - 1.1 Derived month feature.
  - 1.3 Built a function to handle k-fold stratified cross validation with oversampling.
  - 1.4 Built a function to train and test models and return the mean macro F1 and deceased scores as well as the mean accuracy.
  - 1.5 Plot overfitting random forest classifier.
  - 1.7 Integrated given function to generate prediction CSV.

Gursmeep Syan:

- Milestone 1:
  - 1.3 Generating data on plotly maps
  - Report
- Milestone 2:
  - 1.6 Feature importances for seeing robustness
  - 1.5 Overfitting analysis for ExtraTrees and AdaBoost
  - Report