Milestone 2 Report

2.1 Splitting Dataset

We used sklearn's train_test_split function to split the dataset into 80% for training data and 20% for validation data.

2.2 Build Models

- XGB was chosen over adaboosting due to the outliers in our data set which adaBoost could chase. And for model learning efficiency utilizing multiple threads.
- Decision Tree was chosen as one of the models because it is resistant to outliers, which was quite prominent in our dataset. Moreover, a decision tree is ideal for predicting a categorical label and it is easy to understand and interpret.
- KNN is a distance based algorithm and as we have both numeric and categorical features in the dataset, we can use this algorithm for modelling.

2.3 Evaluation

XGBoost

XGBoost was chosen over the other boosting models

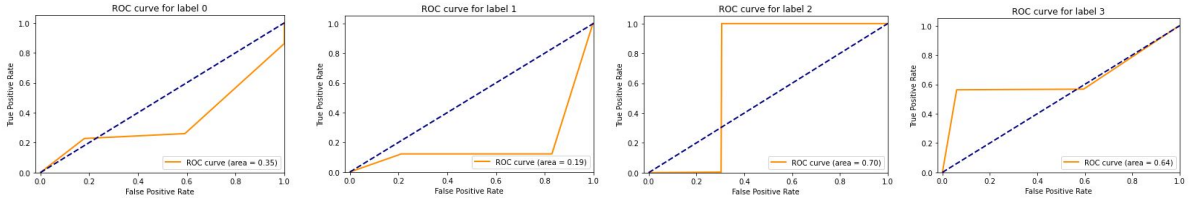
The model against the validation set had an accuracy of 83.7% while the training set had an accuracy of 84.2%.This is a fairly good score.

0col/row= deceased, 1stcol/row= hospitalized, 2ndcol/row = non hospitalized, 3rdcol/row= recovered

The confusion matrix for validation set, And for the training set:

valid:
```
[[   61   581    46   233]
 [    5 21909     6  2992]
 [   19     3 29902   195]
 [   23  7729   174  9650]]
```
train:
```
[[  495  2152   117   814]
 [    0 87896     3 12189]
 [   16     1 119437   427]
 [   14 30507   301 39739]]
```

The confusion matrix for the validation and training set show the same trends. For all labels, they are misclassifying the label quite often, especially for label deceased. This is also supported by the ROC/AUC graphs for each label.



The low ROC curve (area) scores indicate that the XGBoost Model is having a hard time distinguishing the different labels from each other except for non hospitalized.

Decision Tree

Running the decision tree gave an accuracy of 84.4% for the training data and an accuracy of 83.3% for the validation data. This presents a relatively good score, and the relatively closeness of the accuracies indicates that the model is good at fitting to the data, as well as predicting new data.

Decision trees perform a lot of comparison in order to move closer towards an outcome label. Knowing the importance of a feature can help us understand which columns could be removed, in order to speed up the decision tree. Below is the feature importance for the columns of the dataset:

	feature	importance
9	Case-Fatality_Ratio	0.643
2	date_confirmation	0.101
3	combinedKey	0.081
6	Recovered	0.062
1	sex	0.047
5	Deaths	0.026
8	Incidence_Rate	0.017
0	age	0.014
7	Active	0.008
4	Confirmed	0.002

```
[[   77   581    44   219]
 [   39 21922     7  2944]
 [   75    20 29804   220]
 [   53  7830   222  9471]]
              precision    recall  f1-score   support

         0.0       0.32      0.08      0.13       921
         1.0       0.72      0.88      0.79     24912
         2.0       0.99      0.99      0.99     30119
         3.0       0.74      0.54      0.62     17576

    accuracy                           0.83     73528
   macro avg       0.69      0.62      0.63     73528
weighted avg       0.83      0.83      0.82     73528
```

Above, we can see the most important feature is the case-fatality-ratio while the least important feature is confirmed.
Below is the confusion matrix for the validation data and the predicted data in the decision tree model:

KNN

Running the KNN resulted in an accuracy of 82.5% for the training data and an accuracy of 82.3% for the validation data. This shows a fairly good score as the accuracy being close between the two sets of data shows that it is good for fitting the data and predicting new data.

Below is the classification report of the training data and the validation data using the KNN model:

	precision	recall	f1-score	support
0.0	0.62	0.04	0.08	3578
1.0	0.72	0.84	0.78	100088
2.0	0.99	0.99	0.99	119881
3.0	0.69	0.55	0.62	70561
accuracy			0.82	294108
macro avg	0.76	0.61	0.62	294108
weighted avg	0.82	0.82	0.82	294108

	precision	recall	f1-score	support
0.0	0.52	0.03	0.06	921
1.0	0.72	0.84	0.78	24912
2.0	0.98	0.99	0.99	30119
3.0	0.69	0.55	0.61	17576
accuracy			0.82	73528
macro avg	0.73	0.60	0.61	73528
weighted avg	0.82	0.82	0.81	73528

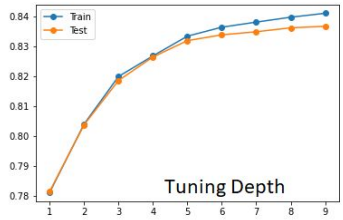
Accuracy Train: 82.47% **Accuracy Validation: 82.29%**

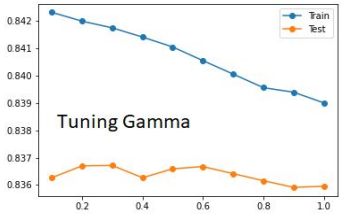
From the above classification reports we see that the model has a precision and recall very close to 1 for label 2.0 which shows that it is good at predicting label 2.0 almost all the time. The model is poor for predicting label 0.0 as the recall for label 0.0 is very close to 0 showing that a lot of the predictions are false negative.

2.4 Overfitting

XGBoost

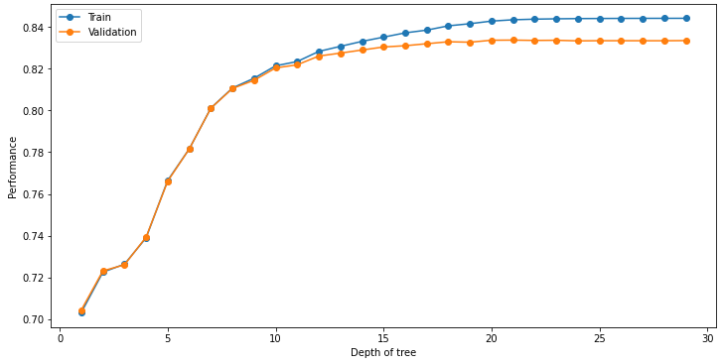
There is slight overfitting on XGBoost when the hyperparameters are set for the model to search deep or do lots of iterations as indicated by these graphs which show the accuracy between training and validation set diverging or converging. For a tuned model, doing 10-foldCV we get an accuracy score of 83.65% with std=0.21% so there is little variance between the scores and low bias and thus no signs of overfitting for the tuned model.





Decision Tree

We observe overfitting in the decision tree when we tune the max depth to be above 12 as show in the graph below:



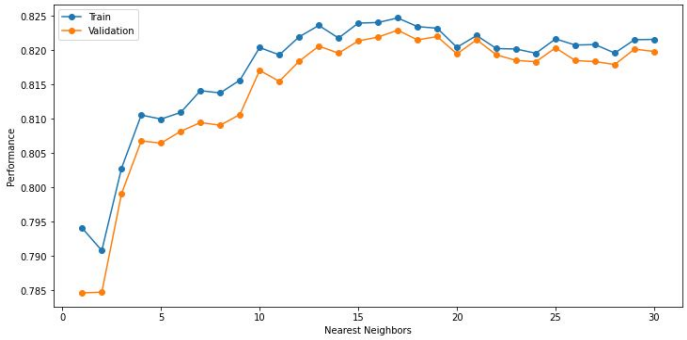
The above is further confirmed by running gridsearchCV on the decision tree model:

```
Tuning hyperparameters for recall

Best parameters set found on development set:

{'max_depth': 24, 'min_samples_split': 6}
```

KNN



The KNN model does not seem to have overfitting as both the training and validation data set seems to have the same performance. The accuracy does not seem increase after n_neighbors = 17, so 17 has the highest performance for the KNN model.