COVID-19 Data Mining Final Report

CMPT 459 - Martin Ester - Spring 2021

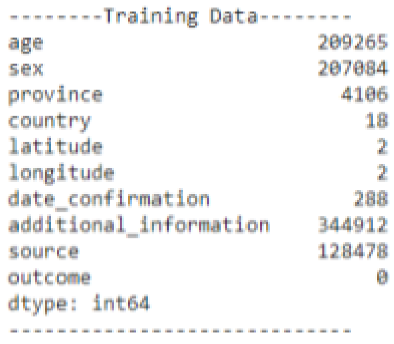
Saqib Aziz Dhuka - 301338669
Cole Stankov - 301295209
Richard Dong - 301317247

1. Problem Statement

Since the first outbreak of COVID-19, millions of people around the world have been diagnosed with the illness. From this there have been many different agencies that have been collecting data on the disease throughout its lifespan. The three datasets that we analyzed are open source COVID-19 datasets that are operated by The John Hopkins University Center for Systems Science and Engineering. The information that is contained within these datasets is a wide set of attributes that are related to countries and people who are affected by the pandemic. The purpose of this study is to use the datasets to predict if a given patient has an outcome of being hospitalized, non-hospitalized, recovered or deceased.

2. Dataset Description and Exploratory Data Analysis

For the first dataset "cases_train.csv", the file contained individual COVID-19 cases for each row, with 367,637 entries in total. The attributes that were contained within the dataset were "age", "sex", "province", "country", "latitude", "longitutde", "date_confirmation", "additional_information", "source" and "outcome". The missing values that were found in each feature can be seen in figure 2.1.



```
--------Training Data--------
age                       209265
sex                       207084
province                    4106
country                       18
latitude                       2
longitude                      2
date_confirmation            288
additional_information    344912
source                    128478
outcome                        0
dtype: int64
-----------------------------
```

Figure: 2.1

After plotting and analyzing the dataset we found that the "age" attribute was not in a consistent format (Ex. 35, 0.666, 20-30, etc.). Which can be referred to in 'Age_cases.png' located in the plots directory. "sex" contained mostly NAN values with 207,084. As for "country" and "provinces" there were 131 and 1106 unique values respectively. The most frequent values for province and country are Maharashtra with 74894 which makes up 20% and India with 212411 which makes up 57% of the data. Columns "province", "country", "latitude" and "longitude" values describe the location of the COVID-19 case. The format of the "date_confirmation" is in day.month.year format. The "outcome" contained only four unique values which are: "hospitalized", "non-hospitalized", "recovered" or "deceased". Lastly, both the "source" and "additional_information" attributes had more than 95% missing values for each.

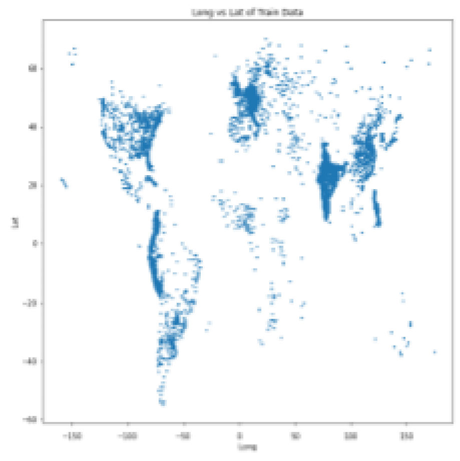


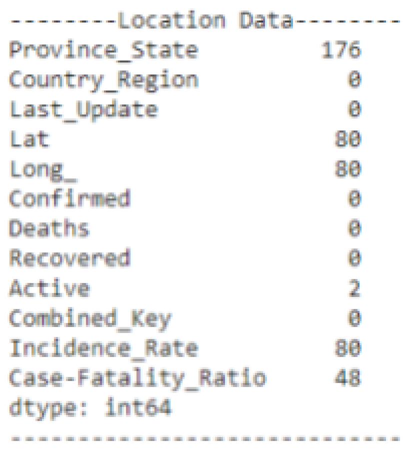
Fig. 2.3

The second dataset we used was "cases_test.csv" which had the same attributes as the first. However, the dataset was much smaller in size, containing 46,500 entries. This dataset was used to test the quality of our models so all of the outcome attributes were removed.

For our third dataset "location.csv", the file contained COVID-19 cases based on location, with 3,955 entries in total. The attributes that were contained in this dataset are as follows: "Province_State", "Country_Region", "Last_Update", "Lat", "Long_", "Confirmed", "Deaths", "Recovered", "Active", "Combined_Key", "Incidence_Rate" and

"Case-Fatality_Ratio". The missing values that were found in each feature can be seen in Figure 2.2.

After plotting and analyzing the dataset we found that "Province_State", "Country_Region" , "Lat" and "Long_" (fig. 2.3) were similar to the attributes within the first dataset where they represent the province or state of the given country and the latitude and longitude of the region. "Province_State" and "Country_Region" have 561 and 188 unique values respectively. The "Last_Update" attribute is the date and time when the records were last updated, however, all except three were identical values. "Confirmed" showed the number of people who have been diagnosed with COVID-19, "Deaths" Show the number of people who have died, "Recovered" signifies the amount of people who have recovered and "Active" is the number of active COVID-19 cases. "Incidence_Rate" is the number of people who have contracted the disease out of 100,000 people. The Formula for incidence rate is given by: *Incidence Rate = Number of Confirmed Cases / 100,000.* Lastly the "Case-Fatality_ratio" is the proportion of deaths from COVID-19 compared to the total number of people diagnosed. The Formula for case fatality ratio is given by: *Case Fatality Ratio (%) = Number of Confirmed Cases / Deaths*. Additional plots of the data were created and can be seen within the 'plots' directory.



```
--------Location Data--------
Province_State        176
Country_Region          0
Last_Update             0
Lat                    80
Long_                  80
Confirmed               0
Deaths                  0
Recovered               0
Active                  2
Combined_Key            0
Incidence_Rate         80
Case-Fatality_Ratio    48
dtype: int64
-----------------------------
```

Figure: 2.2

3. Data Preparation

In the 'age' column, we parsed the string and changed all ages in range format to a single float number by taking the mean of the range. Ages with characters such as "80+" were stripped of the character and only set as "80". Ages that were NaN, were set to the mean age of the country that they were in. Any ages that were still NaN were set to the average age of the entire dataset.

The column sex, rows with NaN in the sex column were replaced with unknown. This is done because we believe there was no merit in setting NaN values in sex to be the average sex of the country. After all, this will severely skew the data since there was a large portion of sex with NaN values.

In the province and country column, we found that in training data, there were only 16 countries that were of NaN values and 0 in testing data. In training data, all NaN countries had Taiwan as their province. To have no NaN value, we decided to set the country to China. In the provinces column, NaN provinces were imputed based on matching latitude and longitude of other rows that had no NaN province. Provinces that were still NaN were set to unknown.

There were no missing longitude or longitude values in testing data, and only 2 were missing in training data. These 2 rows were handled as an outlier and removed from the dataset. Date confirmation had their format standardized, and ranges were set to the middle data of the range. Additional information, source, and outcome were set to unknown because we believe there is no value in inputting false information into these columns. The outcome column had no missing NaN values and was not altered.

The z-score was then calculated to find the outliers for train data. We separated the data type into numerical data and categorical data. For categorical data, we used frequency to get z-scored and find the outliers. We used those outliers to analyze and decide on whether or not we wanted to remove the rows or take no action since they were still important. Figure 2.3 shows the outlier for each attribute for the train data and location data respectively.

True 3 dtype: int64 sex	True 959 False 148 dtype: int64 province	True 107 False 24 dtype: int64 country	True 139 False 16 dtype: int64 date_confirmation	True 13348 False 776 dtype: int64 additional_information
True 5949 False 1302 dtype: int64 source	True 4 dtype: int64 outcome	location: True 502 False 60 dtype: int64 Province_State	True 166 False 22 dtype: int64 Country_Region	True 3 dtype: int64 Last_Update
True 689 False 52 dtype: int64 Combined_Key	True 275739 False 91895 Name: age, dtype: int64	True 283492 False 84142 Name: latitude, dtype: int64	True 367339 False 295 Name: longitude, dtype: int64	

False 3954 Name: Case-Fatality_Ratio, dtype: int64	True 3408 False 546 Name: Long_, dtype: int64	True 3315 False 639 Name: Confirmed, dtype: int64	True 3345 False 609 Name: Deaths, dtype: int64
True 3531 False 423 Name: Lat, dtype: int64	True 3340 False 614 Name: Recovered, dtype: int64	True 3411 False 543 Name: Active, dtype: int64	True 3797 False 157 Name: Incidence_Rate, dtype: int64

Figure 2.3

The number of elements that is not within the 25-75 quantile of its column. (False = outlier, True = inlier)

For 'age' we got 91895 outliers, but we decided to take no action for age because it was within a valid age and we figured that age would be useful for training and predicting an outcome. Furthermore, for 'latitude' and 'longitude', we got 84142 and 295 outliers respectively, but since latitude and longitude were quite important to determine the location and since they will also be useful for training and testing, we decided to take no action. However, we did notice that the train data had two rows where the country, province, latitude, and longitude were missing and we decided to remove those rows since these rows were not of any use.

For categorical data, we used frequency on the unique values of the columns to find outliers. Since the column, 'additional_information' and 'source' were expected to have outliers and since they were not that important, we decided to not take any actions for the outliers in those columns. The columns 'outcome' and 'sex' had no outliers. Furthermore, the columns 'country' and 'province' had 24 and 148 outliers respectively, but since they were an important part of the dataset, we decided to take no action for those outliers. Lastly, 'date_confirmation' only had 16 outliers which were negligible and it also did not make sense to exclude those rows.

The location dataset was then transformed in order to make merging the datasets easier. We first changed the numerical data types to floats and then filled the missing provinces with 'unknown' and corrected the combined keys for each row. Train data had "United States", location data had "US", so we changed the location data to match the train data. For each unique combined key's "Lat" and "Long_" the mean was saved. "Confirmed", "Deaths" and "Recovered", the sums were saved. We then dropped the duplicate rows and replaced the values for each unique row with the mean for that "Combined_Key": latitude and longitude or the sum for that "Combined_Key": "Confirmed", "Deaths" and "Recovered". After that we recalculated the active cases with the formula (“Active” = “Confirmed” - “Deaths” - “Recovered”), the incidence rate with the formula (“Incidence_Rate” = “Confirmed” / 100,000) and “Case-Fatality_Ratio” with the formula (“Case-Fatality_Ratio” (%) = “Confirmed” / “Deaths”).

Cases and location datasets were joined based on the province and country. This decision was that longitude and latitude values were too unique to be used as the key features. We did not use a range either because, with country borders, it is not guaranteed that a line between any two points of the country will be within the border. We believe that this will invalidate province and country legitimacy so we chose province and country (‘Combined_Key’). The merging method we chose is left join because cases_train contains the labeling and this is how classification models are trained. An inner join was not done because we lost too much case data, which is vital for training. The strategy was to have a new column that contains the province and country that we called “combined key” and join it under this column. For the missing values after joining the dataset, we decided to replace the values with ‘-1.0’ to indicate a default value. We were not sure if this was the best way to go since we have not decided on a training strategy yet and it would not be fair to remove or perform preprocessing steps on these missing values.

Lastly, we split the processed train dataset and used the ‘sklearn.model_selection.train_test_split’ function to split the data into a 80-20 ratio. 80 for training our models and 20 for testing them.

4. Classification Models

LightGBM Classification Model:

For the boosting tree algorithm, we chose Python's LightGBM Classifier. The main reason for doing this was that the LGBM classifier is known to efficiently handle a large amount of dataset with less memory usage and our dataset contains more than 300,000 lines/rows [3][4]. Moreover, the LGBM classifier is also known to be quite fast and powerful since it uses gradient boosting. Moreover, the gradient boosting used has the advantage of minimizing loss function by using the additive method to add up the weak learners.

Random Forest Model:

The decision to choose Random Forest over other models is because of its properties as an ensemble classifier and its ease of use. Decision Trees alone are of low bias and high variance, but as an ensemble the variance is reduced greatly. This reduces the amount of overfitting while enhancing accuracy [7]. Another big emphasis on random forest rather than Neural Networks is that we can see the variables. It offers insight on understanding the variables, which may be important in understanding the effect of Covid19 [5] . It is also scalable (based on Tree partitions) so there are less parameters to tune compared to SVM, making it easy to use.

Linear SVC Model:

The last model chosen is a linear support vector machine model. The reason for choosing a SVM model is for its ease of use as well as it is very effective on datasets with multiple features. In comparison to other models such as Naive Bayse, SVM is seen to outperform it in most cases [6]. The one downside to an SVM is that it is not suitable for large datasets normally which the processed data is. However, this is why the SciKit Learn Linear SVC model was used since it scales better to larger datasets in comparison to a default SMV model [1]. The linear SVC model also contains multiclass support with a one-vs-the-rest scheme which is mandatory for our dataset.

5. Initial Evaluation and Overfitting

Evaluation:

For the evaluation of each classification model, we used Sklearn's confusion matrix to extract the true positive (TP), true negative (TN), false positive (FP) and false negative (FN) values for each unique outcome from the training dataset. Since the models were trained without fine tuning any hyperparameters, there were some zeroes and NaNs because the default model returned zeroes for some or all values of TP, TN, FP or FN. The output of each model is shown below. For this classification problem, we believe that accuracy and recall are the most important evaluation metrics. This is because it is important to see how often our classifier is correct, which is accuracy. As for recall, we believe it is important since the nature of the dataset is predicting if a patient with COVID-19 is hospitalized, deceased, etc… False negatives in this case are a lot more costly than false positives, which is why we chose recall over precision.

LightGBM Classification Model:

It can be seen in the graph below that precision and recall are quite low which indicate that false positives and false negatives were a lot more than true positives (as per the formulae). Moreover, the overall accuracy is also not that good which shows that the model did not perform that well on the validation dataset. Overall, this shows that there were a lot of false results as compared to true results since the numbers indicate that FP and FN overpowered TP and TN in terms of quantity.

Deceased	Hospitalized	Non-Hospitalized	Recovered
Accuracy: 0.9878684020835884 Precision: 0.0 Recall: 0.0 F_One: 0.0	Accuracy: 0.6533790308322113 Precision: 0.00347826086956521 Recall: 8.027292795504716e-05 F_One: 0.0001569242840329541	Accuracy: 0.4124335278197125 Precision: 0.408176039391874 Recall: 0.9781992733091103 F_One: 0.5760020413771445	Accuracy: 0.7464332830116828 Precision: 0.06427221172022685 Recall: 0.0038370387089493286 F_One: 0.007241746538871139

Random Forest Model:

The results obtained are subpar. Both deceased and hospitalized labels have NaN and zero values. The model does not predict Hospitalized or Deceased labels for the Testing data, thus we obtain NaN Values. For Non-hospitalized and recovered, Precision=1.0. This means that the False Positives are very low. But the recall, being very close to zero means the false negatives are high, and this is further support in how low the accuracy is.

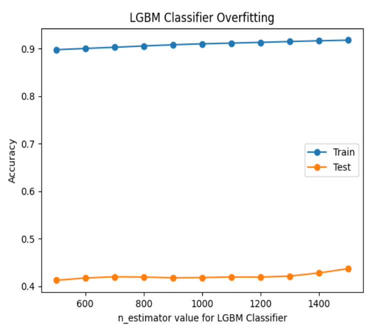
Deceased	Hospitalized	Non-Hospitalized	Recovered
Accuracy: 0.0 Precision: NaN Recall: 0.0	Accuracy: 0.0 Precision: NaN Recall: 0.0 F_One: NaN	Accuracy: 0.0015039604291300424 Precision: 1.0 Recall: 0.0015039604291300424 F_One: 0.0030034038577053993	Accuracy: 0.0028386510730101055 Precision: 1.0 Recall: 0.0028386510730101055 F_One: 0.00566123188405797

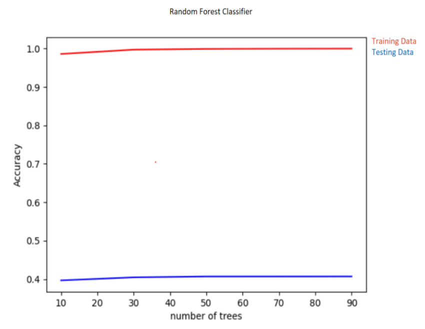
Linear SVC Model:

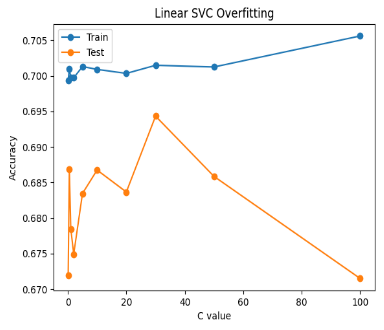
Again the confusion matrix below tells us the result of our model. From the matrix you can see from the output table that the evaluation metric values for Non-hospitalized are extremely high and hospitalized and Recovered are both subpar or average. As for deceased it has a high accuracy rating at roughly 98%, though the other metrics are either nan or zero. As mentioned above, this is due to our model not predicting any deceased labels correctly making the true positive and false positive all zero. We know this by the formula for precision since it is a nan value. Since f-score relies on both precision and recall we see a zero as well.

Deceased	Hospitalized	Non-Hospitalized	Recovered
Accuracy: 0.9872699824554245 Precision: nan Recall: 0.0 F_One: nan	Accuracy: 0.7463380798890203 Precision: 0.573383227346817 Recall: 0.9972847787893308 F_One: 0.7281314228240747	Accuracy: 0.969779808777728 Precision: 0.9658045011756802 Recall: 0.95980771798638 F_One: 0.9627967719251247	Accuracy: 0.757177635426442 Precision: 0.0 Recall: 0.0 F_One: 0.0

Overfitting:







LightGBM Classification Model:

By varying the 'n_estimator' model and plotting the accuracy of the train and validation datasets, we got to see how much the default model overfits for our dataset. It can be clearly noticed in the plot (named 'LGBM_overfitting_graph' inside the plot folder) that the test accuracy with increasing n_estimator parameter was around 41% while the train was around 91%. This huge difference shows us that there was definite overfitting.

Random Forest:

The hyperparameter chosen is the number of Trees in the forest. There is perfect accuracy when running the model on the training data. However, when running on the testing data, our accuracy falls sharply down to 40%. This goes to show that there is very strong overfitting, especially in how the numbers have no impact on the accuracy. The standard parameters set does not do any form of pruning or regularization that is important in managing overfitting.

Linear SVC Model:

The hyperparameter chosen is the C parameter. The reason for choosing the C parameter is that the parameter is looking to minimize error. With the tuning of this parameter there is a bias-variance trade-off. When C is set to a lower value the classifier allows only a small bit of misclassification making the classifier have low bias but high variance due to its generalization to be poor. Converse, if C is large the number of misclassifications allowed are increased. This swaps the values of bias and variance as well. From the C value plots we see that there is a slight gap indicating some variance between the train and test data. However, the gap differs by roughly 3%. This leads us to believe that there is a little bit of overfitting occurring but not enough to make a substantial difference.

6. Hyperparameter Tuning

For hyperparameter tuning our models we decided to use GridsearchCV. Since the models only accept numeric values, before running GridSearchCV or any kind of training of the model, we first converted the categorical data to integers using the sha256 hash function and we also converted the outcomes to their respective integers. An exhaustive search was conducted with the hyperparameters we chose to find the best that increased f1-score and recall on the deceased column while maintaining a high overall accuracy. Moreover, the GridSearchCV was performed with a 5-fold cross-validation (cv = 5) and refit was set to 'False'. The follow are the parameters that were run with GridsearchCV:

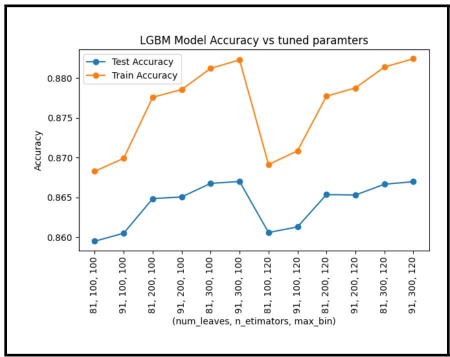
- **LightGBM**: The three parameters that were tuned were num_leaves (81 and 91), max_bin (100 and 120) and n_estimators (100, 200, and 300). Moreover, the learning rate was kept as 0.1 and the boosting type was kept as 'gdbt'.
- **Random Forest**: Three parameters were tuned, max_depth, n_estimators and min_samples_split. For param max_depth = [10, 15, 20, 25], for param n_estimator = [50, 100, 150] and for min_samples_split = [2, 4].
- **Linear SVC**: Three parameters were tuned C, tol and class_weight. For param C = [0.01, 0.1, 1] , for param tol = [0.00001, 0.0001, 0.001] and for class_weight either 'balanced' or 'None'. More parameters were tested to see if any change would occur. It was found that max_itr and intercept_scaling seemed to have no effect on the output evaluations; however, it was noticed that if the penalty was changed from its default 'l2' to 'l1' the evaluation metrics tripled. Therefore, the penalty was set to 'l1' for the hypertunning of the other parameters.

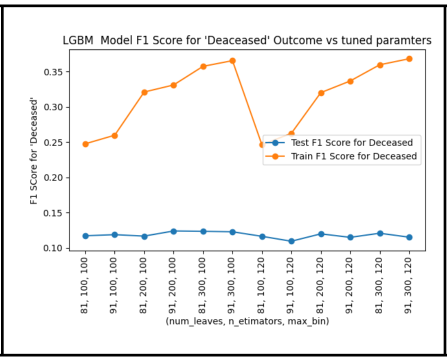
7. Results

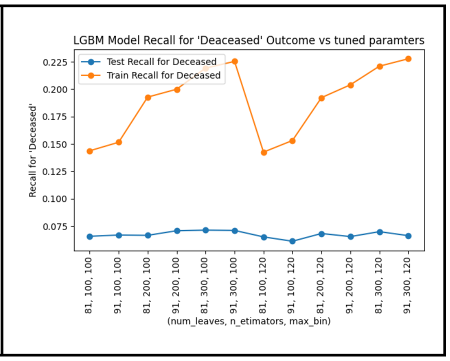
Here is a table of 3 significant scores for each model. These are not the 3 best, but 3 variations of hyperparameters that we thought showed significant change. A full table for each model can be found inside the results folder. After analyzing the evaluation metrics we believe with this type of data that Recall is an important, if not the most important evaluation metric, primarily on the deceased outcome. Recall shows how many correct results were found in comparison to how many correct results should have been returned. This is valuable due to the nature of this project since we want to be able to correctly predict as many deaths as possible in order for medical staff to spend more time treating that patient. With this being said an tuning should be in a direction to maximize the overall recall as well as the recall for deceased. In conjunction to this since F1-score takes both the false positives and false negatives into account by taking a weighted average of both precision and recall it should also be a primary metric to help tune our models. Lastly, the overall Accuracy also needs to be quite high since we want our model to be predicting correctly as much as possible.

LightGBM

Hyperparameters	F1-score on ‘deceased’	Recall on ‘deceased’	Overall Accuracy	Overall Recall
max_bin = 100, n_estimators = 200, num_leaves = 81	Train: 32.08% Test: 11.65%	Train: 19.27% Test: 6.65%	Train: 87.76% Test: 86.49%	Train: 69.33% Test: 65.20%
max_bin = 100, n_estimators = 300, num_leaves = 91	Train: 36.54% Test: 12.26%	Train: 22.53% Test: 7.10%	Train: 88.23% Test: 86.70%	Train: 70.59% Test: 65.56%
max_bin = 120, n_estimators = 100, num_leaves = 91	Train: 26.23% Test: 10.93%	Train: 15.31% Test: 6.11%	Train: 86.93% Test: 86.13%	Train: 67.68% Test: 64.64%



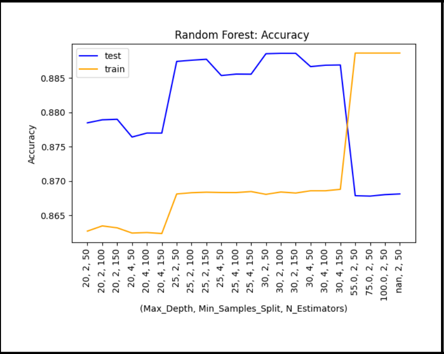


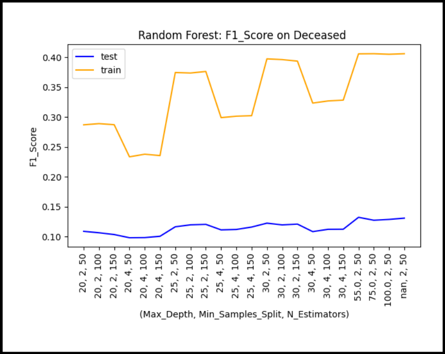


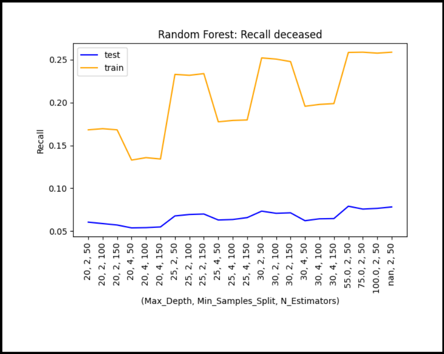
The two plots above, which can also be found inside the plots folder, shows how the scores change with changing parameters. It can be seen that the test scores for f1 score and recall on deceased remain fairly similar while they fluctuate quite a bit for the accuracy graph. The train for all three graphs seems to show the same pattern on both plots. While the graph shows big changes/overfitting between test and train, please note that it is not such a big difference since the y-axis scale is in 2 decimal places.

RandomForest

Hyperparameters	F1-score on ‘deceased’	Recall on ‘deceased’	Overall Accuracy	Overall Recall
Max depth = 20 N_estimator = 50 Min_split = 2	Test: 10.90% Train: 28.71%	Test: 6.05% Train: 16.81%	Test: 86.27% Train: 87.84%	Test: 64.62% Train: 65.55%
Max depth = 25 N_estimator = 150 Min_split = 4	Test: 11.62% Train: 30.23%	Test: 6.58% Train:17.96%	Test:86.84% Train: 88.55%	Test: 60.88% Train: 69.78%
Max depth = 55 N_estimator = 50 Min_split = 2	Test:13.25% Train:40.57%	Test: 7.90% Train:25.84%	Test:85.26% Train: 92.54%	Test:65.90% Train: 72.00%







The plots above show all the sets of parameters that were evaluated. F1-score on deceased increases with greater max_depth, but stagnates at around depth 55. Unexpectedly, the number of trees (n_estimators) had little impact on the F1-score (accuracy only has a +-2% difference so there should not be much overfitting). Min_samples_split had a more noticeable impact on the training data than the testing data, seeing that the chosen evaluation scores dip with increased min_samples_split. (Raw data can be seen in r

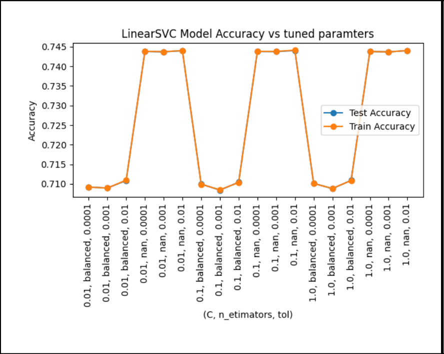
LinearSVC

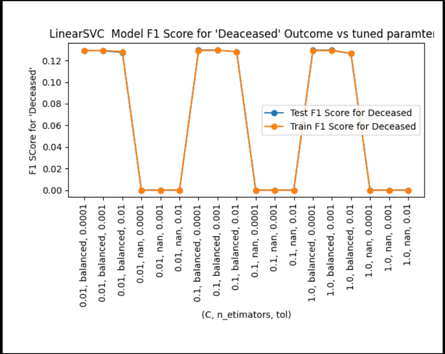
Hyperparameters	F1-score on 'deceased'	Recall on 'deceased'	Overall Accuracy	Overall Recall

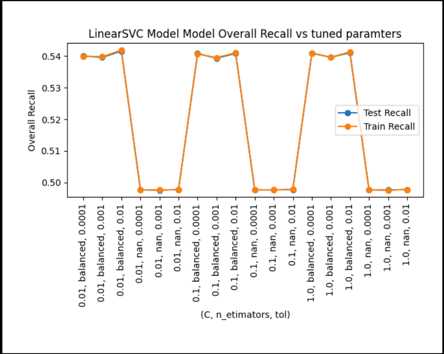
C = 0.01
class_weight = None
tol = 0.00001
	Train: 0%
	Test: 0%
		Train: 0%
		Test: 0%
			Train: 74.38%
			Test: 74.38%
				Train: 49.77%
				Test: 49.77%

C = 0.1
class_weight = Balanced
tol = 0.001
	Train: 12.94%
	Test: 12.96%
		Train: 23.26%
		Test: 23.21%
			Train: 70.84%
			Test: 70.83%
				Train: 53.94%
				Test: 53.92%

C = 1
class_weight = Balanced
tol = 0.001
	Train: 12.67%
	Test: 12.66%
		Train: 23.13%
		Test: 23.03%
			Train: 71.08%
			Test: 71.08%
				Train: 54.13%
				Test: 54.01%







The three plots above are Figures 'LinearSVC_accuracy_plot', 'LinearSVC_F1_score_deceased_plot' and 'LinearSVC_Recall_plot' which can be found in the plots directory. They show that when the classifier's class weight is not set to balanced they receive substandard results for F1 score but slightly better scores for the overall recall.

This 3% increase is negligible compared to the roughly 12% increase to F1-score when balanced. When balanced is set we see that the highest scores for the four metrics above is when tol is set to 0.01, however, the difference is very miniscule. From this we can also see that the C parameter has little to no profound effect on any of the outcomes we analyzed.

8. Conclusion

In terms of F1_Score on Deceased, random forest seems to perform the best with an F1-score of 13.25%. The Recall on Deceased has similar scores to the LGBM model, but it is lower than LinearSVC which has about 23% on test and train. It can be noted that LinearSVC did not overfit whereas the other two models seem to overfit a bit. In terms of overall accuracy, RandomForest and LGBM seem to out-perform the LinearSVC model. RandomForest took quite a long time to perform GridSearchCV compared to other models mentioned above and this was due to the fact that the RandomForest model had a lot of trees to work with. Even though the RandomForest model took the longest amount of time to train, it did end up giving better results for the F1 score on deceased metrics.

9. Prediction on Test Dataset

With F1-Score as the main metric of evaluation, Random Forest with the hyperparameters: n_estimators = 50, min_sample_split = 2 and max_depth = 55, provided us with the best result. This model is used to predict the outcomes of the Test_data. The outcomes are saved into predictions.txt file inside the results folder. In general, we trained the model and called predict on the test attributes (without the outcome column) which gave us predictions as integers and then we mapped the integer back to its respective string form (Deceased, non-hospitalized, hospitalized, recovered).

10. Lessons Learnt and Future Work

The project allowed our team to experience real world data science problems allowing us to see how data mining can be utilized in a very relevant real life circumstance such as the COVID-19 pandemic. It also allowed us to see the steps that should be taken. Starting from preprocessing the data we learned how to first visualize our data and then go to cleaning our data by imputing missing values, removing outliers and joining relevant datasets to help improve our data. It then took us through how to use different models in conjunction with the dataset we preprocessed. Furthermore, it allowed us to better understand how to train, test and tune our models to achieve the optimal evaluation metric we preferred.

As for future work, it is difficult to tell what direction we will all be heading after this project, however, we all do plan to further explore data mining concepts. The relevance and importance that was portrayed throughout this assignment intrigued us to continue with further exploration in data mining diseases/pandemics in particular. It is quite apparent that there is an importance when it comes to data mining for that field.

11. References

1. https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
2. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
3. https://machinelearningmastery.com/gradient-boosting-with-scikit-learn-xgboost-lightgbm-and-catboost/
4. https://www.analyticssteps.com/blogs/what-light-gbm-algorithm-how-use-it
5. https://towardsdatascience.com/3-reasons-to-use-random-forest-over-a-neural-network-comparing-machine-learning-versus-deep-f9d65a154d89
6. https://towardsdatascience.com/support-vector-machines-a-brief-overview-37e018ae310f
7. https://www.ibm.com/cloud/learn/random-forest

12. Contributions

Saqib Aziz Dhuka (sdhuka) - 301338669

Milestone-1: Worked within the group to impute missing values and decide on outliers of columns assigned to me and worked on joining the dataset correctly.

Milestone-2 and Milestone-3: Worked on building and tuning the LGBM Model.

Cole Stankov (cstankov) - 301295209

Milestone-1: We worked as a group to impute missing values, plot the diagrams and decide on outliers of columns. Worked alone on the transformation of the location_data to be merged into the train data.

Milestone-2 and Milestone-3: Worked on building and tuning the Linear SVC Model.

Richard Dong (rcdong) - 301317247

Milestone-1: We worked as a group to impute missing values, plot the diagrams and decide on outliers of columns. Worked alone on the outlier detection for the datasets.

Milestone-2 and Milestone-3: Worked on building and tuning the Random Forest Model.