Covid-19 Prediction Models Report

“The Classifiers”
Daniel Bennett 301380010
Jason Cai 301426943
Liam Duncan 301476562

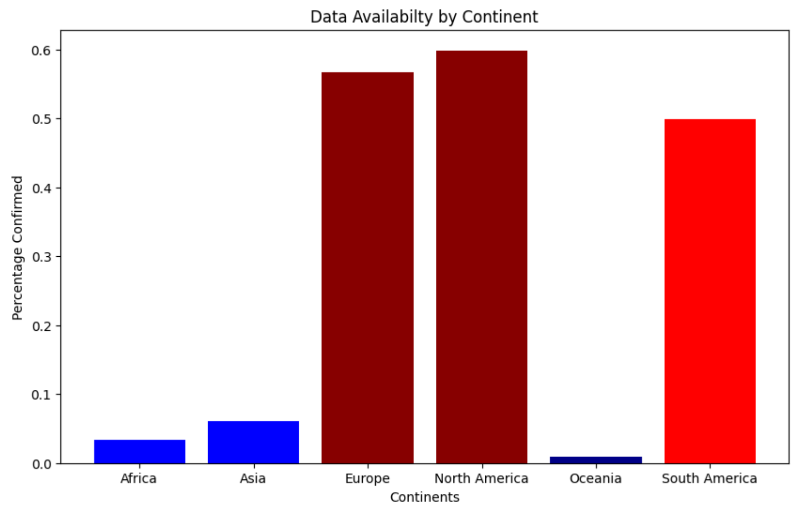
Problem Statement

The task is to develop and evaluate classification models predicting the outcomes (non-hospitalized, hospitalised, and deceased) of COVID-19 cases. To gain insight on the data, visualisations of the geographic information were utilised. The visualisations provided spatial understanding of the data regarding case count in relation to populations. The geographic information assisted with feature selection. Achieving the highest accuracy, macro f1-score, and f1 score for the deceased class is the priority for each of the given models. The project outlines the steps taken to achieve optimal prediction scores and interpretation of the results obtained.

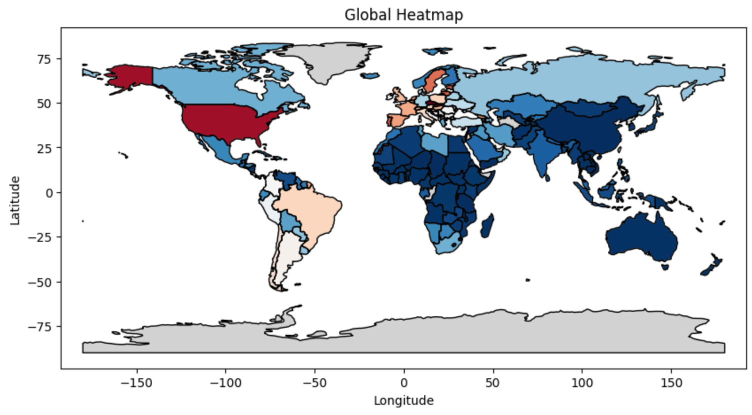
Data Visualization

Task 1

For the Data Visualization, we first created a bar plot (shown below) for data availability based on continents. As per the Professor's suggestion, we used data availability from ONLY the location dataset. We did not use data in the cases dataset for the bar plot or the heatmap as per this suggestion by the Professor. The professor also suggested data availability meant the percentage of confirmed cases divided by the continent's population. The higher the percentage was for the continent, the colour on the bar plot was more red, the lower the percentage was for the continent, the colour on the bar plot was more blue.
Yes, there are a few anomalies that are visible in the bar plot. One thing that we saw as an anomaly was the fact that Asia had a lower percentage confirmed than Europe and North America. Since Asia has a much higher population and population density, logically we would have thought that Asia would yield a higher Percentage confirmed than the other continents. We also felt that since a lot of African countries are less developed countries that would have had a higher Percentage of population confirmed. Comparing this to the overall population of the continent, continents with higher populations did not necessarily mean it had a higher percentage of confirmed cases per population.
After creating the bar plot, we then created a colored heatmap of the world. On this colored heatmap. We colored the heatmap based on the percentage of available data based on the country's population. Once again, as per the Professor's suggestion, we did this using only the location dataset and did not include the cases dataset. The heatmap better showed which countries in the world had a higher percentage of the population that was confirmed. One thing to notice that we would not notice from the bar plot is that the United States of America was the cause of such a high Percentage, as it is more red on the heatmap, while however, Canada and Mexico are more blue, meaning they had a much lower Percentage. The scale of the heatmap is from 0% to 1% with 0 being dark blue and with 1 being darker red



Heatmap
Percentage of available data based on the country's population



Data Preprocessing

Task 2

Handle missing values.
In the datasets we were given there were many missing values in the datasets. Part of Preprocessing is handling these missing values. There are many different ways to handle missing values. The best way that we found to handle missing values in the dataset was by replacing them with N/A. For missing values in the cases dataset we decided that the values would be replaced with N/A. For missing values in the location dataset, we decided that the value would also be replaced with N/A.
As part of the Preprocessing process, we were also required to join the location and case data. The way we joined the location and case data is on the country and the province combined as a primary key. We also added a feature called Expected_Mortality_Rate.

Data Preparation

Task 3 - Feature Selection

There were some features that we removed completely and others that we created. First, we removed the country and province labels. We did this because it would be extremely difficult to effectively map these categorical features to numeric while maintaining some sort of relevancy. Instead of using country and province, we created 2 new attributes based on the average statistics for each location, the average_rate and average_ratio. These attributes were created by averaging the rates and ratios between the different unique province entries as these statistics were values that described the overall situation inside each province. Our rationale for using these values was to provide another factor for the model to keep into consideration, allowing the model to keep the country's overall conditions in consideration such as development, healthcare, and disease management. The rest of the attributes we decided to keep. Features such as age, sex, and chronic disease indicate the patient's overall health. The longitude and latitude were kept as they can be used by the model to generalize certain trends for different areas on the planet. The date confirmed was kept because the date played an important role in how the virus was treated. This includes vaccinations, lockdowns, health care availability, knowledge of the virus, and a multitude of others. The incident rate was kept as an indicator of the case count of the surrounding population as this may have an impact on things such as healthcare availability. The case fatality rate was kept as it had a direct correlation to one of the 3 class labels. As a group, the features selected for the reasons stated would result in the highest correlation between data points of the same classes.

Task 4 - Mapping the features
To assist with training the model, we chose to convert a few attributes from categorical data to numerical data. In particular; "sex", "chronic_disease_binary", and "outcome_group", were converted into simple integer values (based on the unique possible values for this attribute). We also chose to convert the "date_confirmation" attribute to a numeric value and normalized it based on the time range outlined by the data, this was to make it so that the model could easily identify numerical thresholds for different periods during the COVID pandemic, potentially allowing it to better classify a data entry's outcome group.

Task 5 - Balancing the classes in the training dataset
To balance the data, we tried 3 different methods. The first was over-sampling. In this method SMOTE from imblearn.over_sampling was used to generate synthetic data representing the minority classes. By creating synthetic data from the minority class, we can make the classes balanced without using copies of the minority data and therefore have redundancies. SMOTE is an effective method for oversampling. However, solely over-sampling using SMOTE can create noisy data that may negatively impact the model.

The second method was undersampling. This method was mostly conducted manually using the Pandas and Numpy library. We started by first identifying the attributes that are important to keep in consideration in terms of minority values, which were: "chronic_disease_binary", "region", "age" and "date_confirmation". We then iterated through values and determined a threshold for the minority values inclusion and we extracted all the entries that weren't of the "deceased" "outcome_group". During the process of setting the threshold, we had to strike a balance of including as many "minority" values as possible while also leaving decent amounts of space to fill in with the "non-minority" values. Afterward, we took count of the remaining space to fill for the other "outcome_group"s and filled in the remaining space with random data sampled in the respective classes.

The third and final method combines both oversampling and undersampling into one hybrid balancing method. In this method SMOTEENN from imblearn.combine was used. The idea behind SMOTEENN is to create synthetic data representing the minority class but also remove nearest neighbour misclassification. To generate the synthetic data, the SMOTE method is used as described above. To remove data ENN is used to undersample. The nearest neighbour of the majority classes is estimated, if that specific nearest neighbour misclassifies that data, then it is removed from the data point. Ultimately we decided to use the hybrid class balancing technique as we were able to capitalize on the positives from both methods and minimize the negatives.

Classification Models

Task 6 - Building models and hyperparameter tuning

XGBoost

XGBoost uses decision trees as the base classification algorithm. XGBoost tends to outperform other classification models. One good thing about XGBoost is it does not overfit as much as some other models. Also even if it does overfit a slight change in the value of the hyperparameter. When tuning the model I initially used RandomSearchCV to hypertune the classification model. However, as I got more specific in my tuning it started to become very inconsistent in my search. Since I was getting these inconsistencies, I decided to switch from RandomSearch to GridSearch to see if I got better performance and consistency. Another good thing about XGBoost is it is similar to Random Forest, however XGBoost is more efficient and more accurate than Random Forest. For the K-fold, I decided to use K=5. I found that with K=5 I still got great results because of the K-fold cross-validation without significantly increasing the training time. In my hyperparameter tuning, I used the hyperparameters min_child_weight and max_depth. After multiple iterations, I was able to see that the best values for the hyperparameters were (min_child_weight: 0.42, max_depth: 19, n_estimators: 200). Before I trained the model, I split the data 80:20 ratio of training and validation.

Random Forest

Random forest holds a variety of benefits that make it stand out from other models. Typically random forests have a high accuracy rate due to multiple decision trees working together to produce 1 classification. It also has a lower chance of overfitting as each tree is trained on a random subset of the data instead of being trained on the same data. This reduces the likelihood that the tree has overfitted. Not only does the random subset of training data reduce the odds of overfitting, but it also reduces the odds of being affected by noisy data. Instead, it can pick up the complex patterns of the training data. Another bonus to using random forests is that I had experience using it; which enabled me to tune the model better based on my prior knowledge of the classifier. I ultimately decided to choose K = 5. The main reason that I chose 5 was because it provided the benefits of k-fold cross-validation

without drastically increasing the training time. The other reason for choosing k somewhat low is due to the already randomness of the random forest. Since it already takes a random subset of the data it will lower the chances of overfitting and keep reducing the odds of the model knowing the data instead of the patterns of the data. Therefore, a higher k value was not necessary. To hyperparameter tune I opted to use GridSearchCV due to its exhaustive capabilities. Using my prior random forest knowledge I was able to have a rough idea of what values to put in the grid. Ultimately the best values were 'max_depth': 40, 'min_samples_split': 10, 'n_estimators': 300 as reported in the .txt file. These values were later changed due to overfitting detection. The final values are 'max_depth': 19, 'min_samples_split': 10, 'n_estimators': 300. Resulting in the final scores of:
Before training the model the data was split into training and validation with a ratio of 80:20. With the given training data, I then performed StratifiedKFold cross-validation with k = 5. StratifiedKFold was used as it selects data maintaining the class imbalances. In the hybrid data, the classes were not perfectly balanced which is why I chose to preserve the minor difference when creating the k folds.

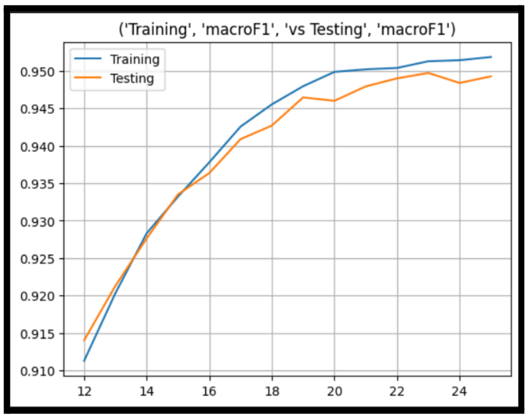
<u>Support Vector Machine</u>
SVMs work well with datasets with a large number of features and non-linear relationships between features and classes. While our dataset didn't have too many features, the relationship features and classes were non-linear. SVMs are also tolerant to irrelevant features, which would provide additional security from potential incorrect classifications due to irrelevant features. Though I wasn't as experienced with SVMs as with other models that were available to use, I was curious as to whether SVMs might be able to generate an adequate hyperplane that can accurately classify data points from the attribute values provided to it (despite the relatively low dimensionality of the dataset). In terms of K-fold cross-validation, there was a shared sentiment that K=5 would be an adequate value considering the size to unique values ratio of the dataset. For hyperparameter tuning, I utilized the GridSearchCV to cover all possible hyperparameter values; checking all 3 types of kernels, C values, and each of their parameters. However, after waiting for 2+ hours for the hyperparameter tuning, I realized how time-consuming the model training was for SVMs, and ultimately made the decision to reduce the hyperparameters to check. Taking the robustness and time consumption into consideration for model development, the "RBF" kernel provided the best value for both categories as it provided better robustness than the "linear" kernel while also taking relatively less time than the "poly" kernel. Conducting another round of hyperparameter tuning after choosing the "RBF" kernel, the results yielded a C value of 1000 (from a range of 0.0001 to 1000)and a gamma value of 0.1(from a range of 0.001 to 0.1).

Model	**Hyperparams**	**Mean macro F1 over validation**	**Mean deceased F1 over validation**	**Mean overall accuracy over validation**
XGB	Objective: "multi:softmax" Min_child_weight: 0.42 Max_depth: 19 N_estimators: 200	0.97082	0.95526	0.97133

Random Forest	Max_depth: 40 Min_samples_split: 10 N_estimators: 300	0.95015	0.92293	0.95115
SVM	C: 1000 Gamma: 0.1	0.90851	0.85924	0.91093

Task 7 - Overfitting

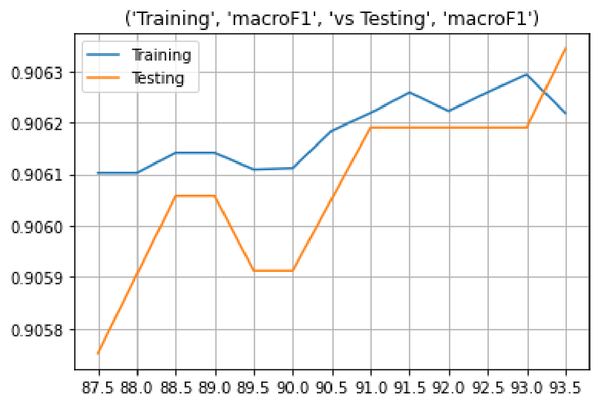
After the results from our hyperparameter tuning, we created multiple models on the training data with varying values for hyperparameters. To start we used the hyperparameter values that our tuning methods returned. In the case of the Random Forests, we trained models with varying values starting from 5 up to 40 for max_depth. The training scores and test scores were calculated for each value and the results were then plotted. When max_depth was set to 19, it appeared to give the best scores before the overfitting. At 20, the training score starts to deviate away from the testing score; which is a telling sign of overfitting. Below is the graph for the described testing.



With the new max_depth value, the same procedure was used to find where the min_sample_split began to overfit. The value the grid-search returned, 10, was used as a starting point and values were tested up to 200. The lower sample split, indicated no signs of overfitting as the test scores followed the same path as the training scores. Due to the line similarity, it was evident that the grid-search had returned the best possible value for min_sample_split without any overfitting occurring. Finally, the same procedure was done with n_estimators. The different values did not show any signs of overfitting and therefore the value returned from grid-search was used.

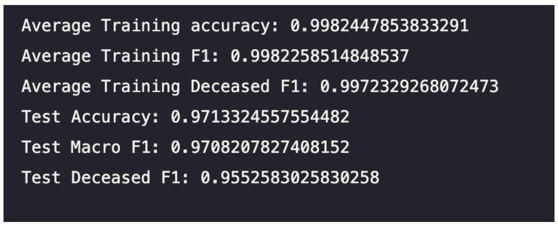
By plotting the different hyperparameter values and seeing where the training scores increased when the test scores decreased we were able to see overfitting. With the given results, we then picked the best possible values for each hyperparameter that showed no signs of overfitting. All 3 of the different models followed this same idea; use the hyperparameter tuning as a baseline, try values close to the suggested hyperparameter value, plot the resulting graph, look for signs of overfitting, and then adjust the model to use the best hyperparameters without overfitting.

Similar to Random Forests, different SVM models were also plotted with variations of the hyperparameters gamma and C to determine the overfitting of the model. The variation in gamma was set between 0.0001 to 0.1 and the plot showed that the change in the hyperparameter did not affect how the model performed, shifting the focus to the C value. Plotting the C value resulted in more visible differences and as a result; we were able to determine that 91 was the best value for C for the best result and low overfitting possibility.

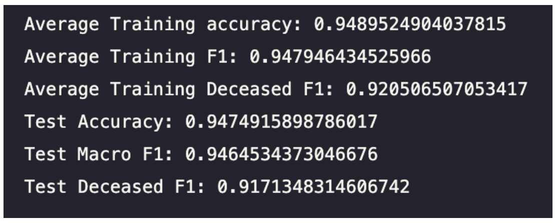


Task 8 - Comparative Study

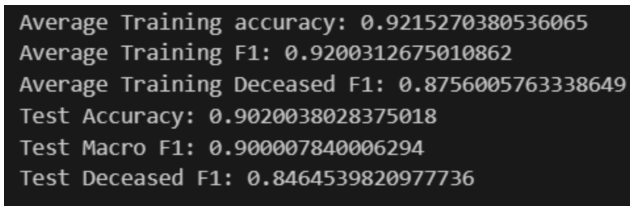
The XGBoost appeared to perform the best across the board. One of the main advantages to the XGBoost is its efficiency this is important when training multiple models and comparing results. It also uses a self pruning technique which reduces the likelihood for overfitting. Finally it is widely used in competitions for high prediction results which is appealing. One major disadvantage of XGBoost is the complexity making it hard to understand. However, due to the amount of time for the project we were able to understand the model and effectively tune the hyper parameters. These are the results for XGBoost:


```
Average Training accuracy: 0.9982447853833291
Average Training F1: 0.9982258514848537
Average Training Deceased F1: 0.9972329268072473
Test Accuracy: 0.9713324557554482
Test Macro F1: 0.9708207827408152
Test Deceased F1: 0.9552583025830258
```

The random forest model also performed relatively well. Random forests are an effective classification model due to its randomness and reduced variance while keeping the bias low as well. The random selection of training data is an effective way to combat the normal increase in bias when bagging. Another advantage is that it's hard to overfit a random forest. The other major advantage of the random forest is that we already had experience working with them. One major disadvantage of the random forest is computation time. This was noticeable when trying grid search. Another disadvantage is that they are sensitive to hyper parameter values, making them tough to perfect. These are the results for the random forest:


```
Average Training accuracy: 0.9489524904037815
Average Training F1: 0.947946434525966
Average Training Deceased F1: 0.920506507053417
Test Accuracy: 0.9474915898786017
Test Macro F1: 0.9464534373046676
Test Deceased F1: 0.9171348314606742
```

The SVM model performed the worst of the 3 models. SVM's are especially effective when there is a clear margin between the classes. This is because it is creating a hyperplane. It is also effective in high dimensional data spaces. The kernel trick allows the data to effectively map the data into different dimensional spaces making it an appealing model. A major disadvantage of SVM is that it is very susceptible to noise as it can be hard to create a distinct hyperplane. Another disadvantage to this model is that there is no probabilistic explanation for the results. This is because the predictions are based strictly on where the data lies in relation to the hyperplane. These are the results for SVM:



```
Average Training accuracy: 0.9215270380536065
Average Training F1: 0.9200312675010862
Average Training Deceased F1: 0.8756005763338649
Test Accuracy: 0.9020038028375018
Test Macro F1: 0.900007840006294
Test Deceased F1: 0.8464539820977736
```

Based on these findings XGboost had the best scores across the board. It also had the quickest run time which was appealing. The random forest was close in scores however the XGBoost was slightly better all around. For these reasons we chose XGboost to be our main classification model.

<u>Model Predictions - Task 9</u>

The predicted data followed a similar imbalance to the labelled data. This is a good sign. The hospitalised was the majority class, followed by non-hospitalised, and then deceased as minority.

<u>Conclusion</u>

From task 3-9 we essentially followed a real world example of how to preprocess data, design models, tune models, and compare these models. One interesting thing we found was how different the models were for the same dataset. Although the data was preprocessed and we all used the same data, our results varied. This shows the importance of choosing the correct model and how it can impact the results.

<u>Lessons learned and future work</u>

The biggest lesson we learned was the computational time for training models. Across all 3 models tuning the hyper parameters took extended periods of time. We also learned the importance of hyperparameter tuning. Prior to this we had no experience tuning hyperparameters and therefore didn't realise the effect it would have on the models. To improve this project more time could have been spent tuning the parameters further to ensure we get the best results. We would also spend more time detecting overfitting as this was an issue for all the modes.

References

Scikit Learn: https://scikit-learn.org/stable/user_guide.html
IMB Learn: https://imbalanced-learn.org/stable/user_guide.html
World Map: https://www.naturalearthdata.com/

Contribution

Daniel - Took the lead on the data visualisation (tasks 1 and 2). Trained the XGBoost model and experimented with the random search technique for hyperparameter tuning.
Liam - Assisted Daniel with effectively joining and visualising the given datasets. Designed the oversampling and hybrid class balancing. Trained the random forest model. Designed modular code to test overfitting for the different models. Compared Models.
Jason - Mapped the data and performed feature selection. Designed the undersampling class balancing. Designed the SVM model, and added onto the modular code by Liam to check the overfitting of the other models.