Alvin Ho
Kenrick Lam
Steven Tran

CMPT 459 Course Project Milestone 3

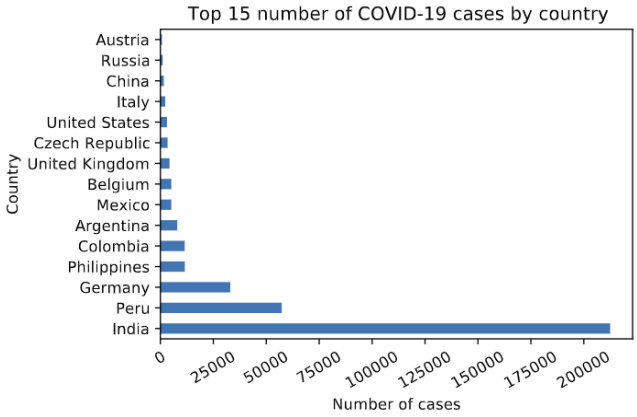
Problem Statement

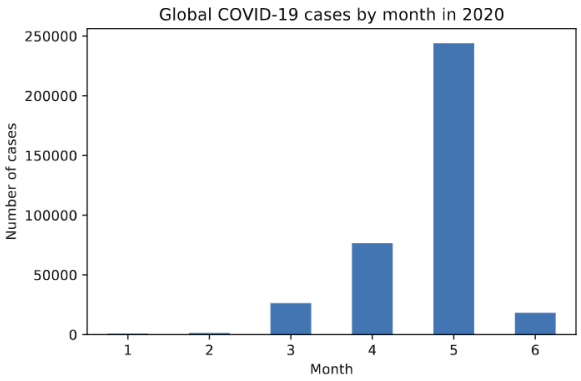
The goal of this project was to get first-hand experience working with data. We looked at data collected from COVID-19 cases around the world and were tasked to clean this data and then use classification models on our training data to predict the outcome of a patient from the test data.

Dataset Description and EDA

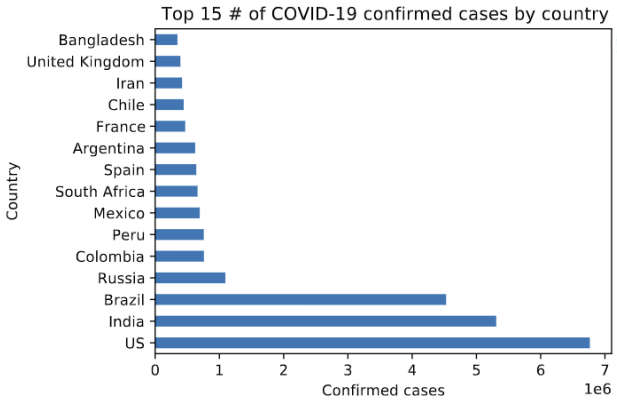
When analyzing the dataset using visualizations, we found the following plots to be the most insightful.

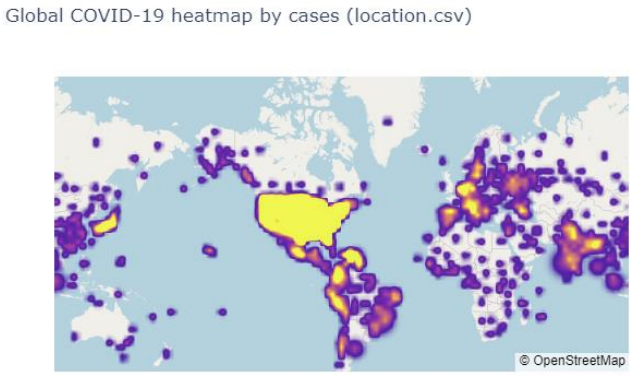
The following two plots were generated using data from cases_train.csv. These two plots tell us that there were a lot of individual cases collected from India between the months of March and June, but most heavily in May.





To contrast the data above, the next plots were generated using data from location.csv, which contained the number of cases for each location instead of individual cases. We observed many confirmed cases in United States which surprised us considering how few cases there were in cases_train.csv. This difference tells us that either there were many inaccuracies in data collection, or that either dataset was sampled by convenience since a simple random sample of global COVID-19 statistics should provide similar results.





Alvin Ho
Kenrick Lam
Steven Tran

CMPT 459 Course Project Milestone 3

Data Preparation

When cleaning data, we looked to replace blank rows with “Unknown” values, and manually cleaned the noisiest column of ages by looking through all unique values and replacing age ranges and dates with valid ages. We also took a manual approach when looking for initial outliers, looking through unique values of all the columns for strange values. We found that rows of data taken from the province of Lima had seemingly erroneous data, with almost all their ages being under 1 or over 100, so we decided to remove those rows.

While transforming data from the location dataset, we aggregated the data of individual counties into their respective states for the US, and aggregated individual provinces into their respective countries for the rest of the world. We then joined this data based on province and country with the training and test data, and only included columns that we felt would give us meaningful knowledge. This included total incidence rate, provincial incidence percentage, total case-fatality ratio, and provincial case-fatality ratio.

Classification Models

The three classification models that we chose to use were XGBoost, K-Nearest Neighbours, and Random Forests. As an overview, we chose two ensemble methods: XGBoost and Random Forests, because they are robust to outliers and non-linear data, and they produce results with high classification accuracy. As for KNN, we chose it because we felt the results were very simple to interpret and it is robust regarding search space since classes do not have to be linearly separable like in SVM.

First let us discuss our decision to choose XGBoost as our boosting tree variant versus AdaBoost. XGBoost is much faster than AdaBoost and has a multitude of hyperparameters that can be tuned to increase performance, whereas AdaBoost only supports tuning for max depth, learning rate, and number of iterations [1]. XGBoost also has a regularization parameter that reduces variance which AdaBoost lacks.

Next, we will compare Random Forests versus Decision Trees. The problem with Decision Trees is that single decision trees are prone to overfitting (high variance), especially when a tree is grown deep. The classical way to combat this issue is to set a max depth but that increases the error due to bias [2]. Random Forests essentially minimizes the error due to variance and bias by using a collection of decision trees whose results are aggregated into a single result.

Finally, we will compare KNN to SVM. As previously mentioned, we mainly chose KNN since its results are easily interpretable. However, it is important to note that SVM handles outliers better than KNN does [3]. Although in terms of performance, KNN outperforms SVM when the number of training data objects if far greater than the number of features, which in our case is true.

Alvin Ho
Kenrick Lam
Steven Tran

CMPT 459 Course Project Milestone 3

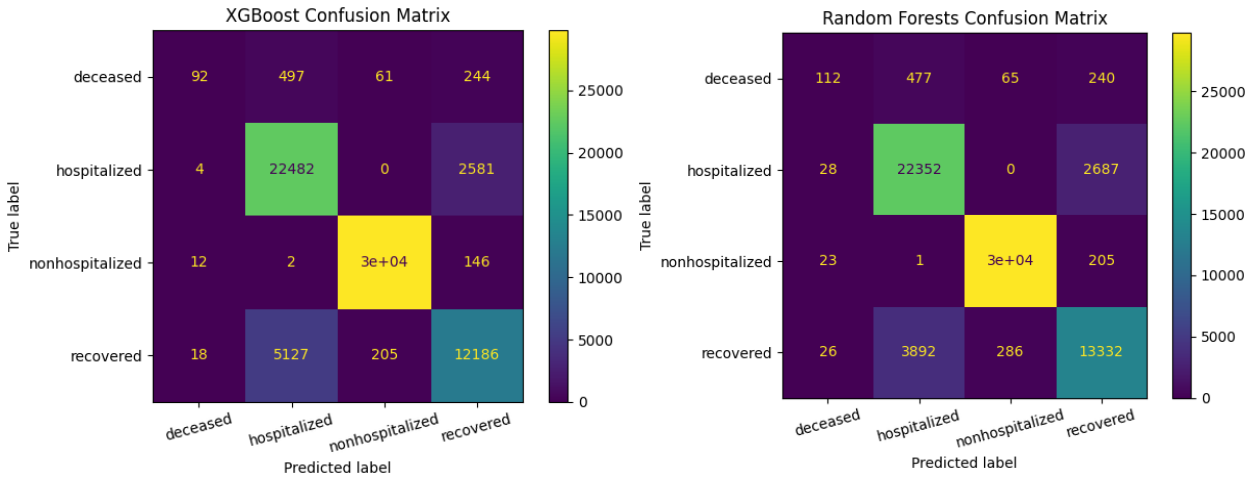
Initial Evaluation and Overfitting

Initially, we considered prediction accuracy as our main metric for evaluating the three classification models. However, we realized that prediction accuracy was not the best metric to use since the COVID-19 dataset was highly imbalanced and therefore, we would have gotten incorrect insights. We decided to use the recall on the 'Deceased' outcome label as our main metric. Using scikit-learn's classification report, we got the recall on the 'Deceased' outcome label for each of our models on both the training and validation datasets.

Training Dataset	'Deceased' Recall	'Deceased' Precision	'Deceased' F1-Score
XGBoost	0.1781	0.9386	0.2994
Random Forests	0.3393	0.9730	0.5031
K-Nearest Neighbours	0.3628	0.8679	0.5117

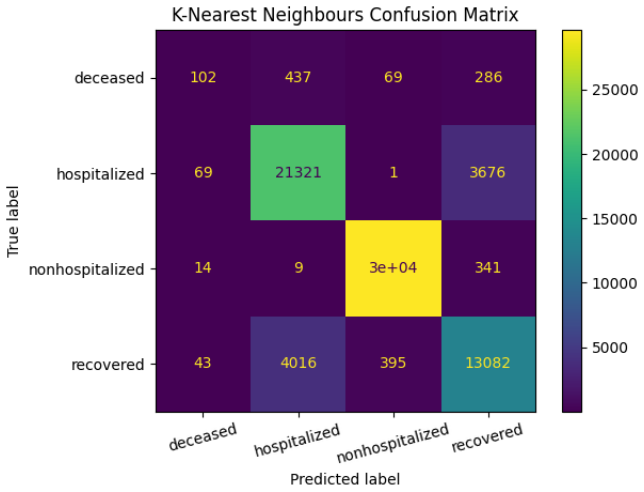
Validation Dataset	'Deceased' Recall	'Deceased' Precision	'Deceased' F1-Score
XGBoost	0.1029	0.7302	0.1804
Random Forests	0.1253	0.5926	0.2068
K-Nearest Neighbours	0.1141	0.4474	0.1818

We also used scikit-learn's confusion matrix on the validation dataset for each of the models. For each confusion matrix C below, we define C_{ij} to be the number of data objects known to be in group *i* (true label) but were predicted to be in group *j* (predicted label). From these results we can derive insights like how each model is incorrectly predicting data objects that are deceased to be hospitalized. This implies that there are more false negatives than true positives for data objects with 'deceased' outcome label.

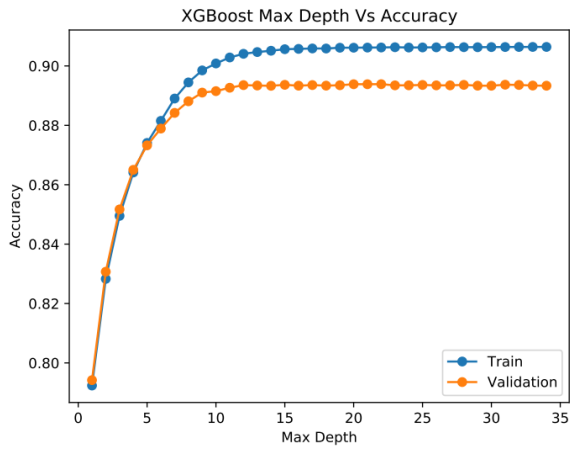


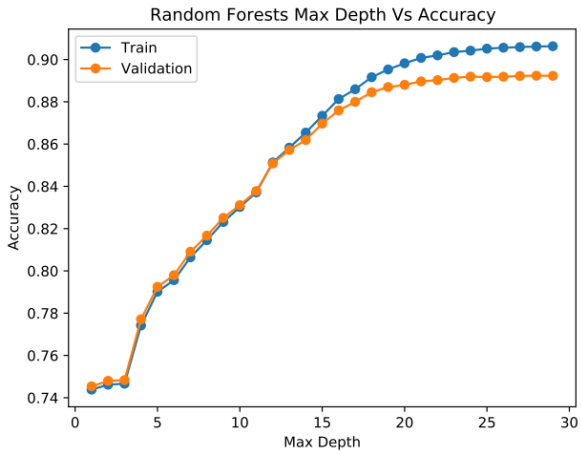
Alvin Ho
Kenrick Lam
Steven Tran

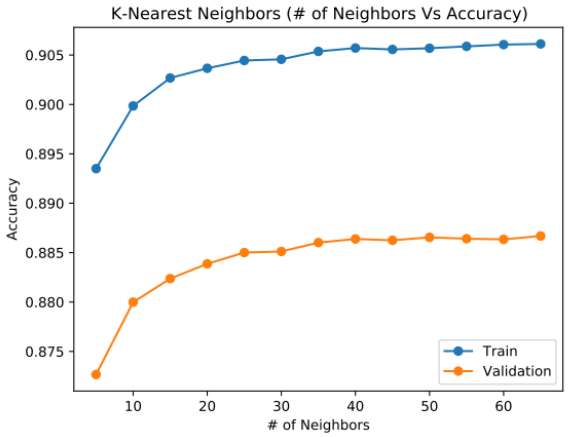
CMPT 459 Course Project Milestone 3



For the classification models we trained, we did not observe any overfitting. We noticed that the accuracy tended to stagnate for both the training and validation data after further increase of the hyperparameters, as can be seen by the plots below. We tested different values for each model's respective hyperparameter to find the most optimal values for them and take precautions towards overfitting.







Alvin Ho
Kenrick Lam
Steven Tran

CMPT 459 Course Project Milestone 3

Hyperparameter Tuning

For each of our models, we tuned their respective hyperparameters using GridSearchCV. We choose GridSearchCV versus other techniques such as RandomSearchCV because GridSearchCV allows us to define the combinations of parameters we wish to use while RandomSearchCV selects these combinations randomly. In addition, the RandomizedSearchCV does not try out all parameter values, but rather only a fixed number of iterations. Therefore, GridSearchCV should theoretically provide more precise results which may be crucial when tuning >= 3 hyperparameters simultaneously. These are both effective ways for tuning the models, but we felt GridSearchCV provided slightly better results than RandomSearchCV from our testing.

Our approach for generating our tuned models was to first run a cross validation on each model using 3-4 hyperparameters that we felt were most important to that model each with a wide range. We also utilized 4 evaluation metrics for scoring simultaneously: F1-Score on 'deceased', recall on 'deceased', overall accuracy, and overall weighted recall. We then tried to refit the model to F1-Score on 'deceased' to optimize it towards that evaluation metric. Afterwards, we could compare the results of GridSearchCV's best_params_ and best_estimator_ attributes to obtain of baseline of where we can start tuning our hyperparameters. The process from there was to just slightly alter every hyperparameter for each model and observe any increases or decreases to F1-Score on 'deceased', which we considered our primary criterion for judging our model's performance.

Results

Following our approach in 'Hyperparameter Tuning' we generated the following results for each model. The optimal hyperparameters that gave us the highest F1-Score on 'deceased' value is highlighted in green for each model.

XGBoost Hyperparameters	F1-Score on 'deceased'	Recall on 'deceased'	Overall Accuracy	Overall Weighted Recall
learning_rate = 0.6, max_depth = 18, n_estimators = 100	0.2095	0.1286	0.8936	0.8936
learning_rate = 0.6, max_depth = 18, n_estimators = 105	0.2121	0.1309	0.8936	0.8936
learning_rate = 0.6, max_depth = 18, n_estimators = 110	0.2111	0.1298	0.8936	0.8936
learning_rate = 0.6, max_depth = 19, n_estimators = 105	0.2125	0.1309	0.8937	0.8937

learning_rate = 0.6, max_depth = 20, n_estimators = 100	0.2093	0.1286	0.8937	0.8937
learning_rate = 0.65, max_depth = 19, n_estimators = 105	0.2114	0.1309	0.8937	0.8937
learning_rate = 0.55, max_depth = 19, n_estimators = 105	0.2156	0.1331	0.8938	0.8938
learning_rate = 0.5, max_depth = 19, n_estimators = 105	0.2079	0.1264	0.8941	0.8941

Random Forests Hyperparameters	F1-Score on 'deceased'	Recall on 'deceased'	Overall Accuracy	Overall Weighted Recall
max_depth = 25, max_features = 0.9, min_samples_leaf = 1	0.2070	0.1253	0.8931	0.8931
max_depth = 25, max_features = 0.8, min_samples_leaf = 1	0.2045	0.1230	0.8927	0.8927
max_depth = 25, max_features = 1.0, min_samples_leaf = 1	0.2055	0.1253	0.8927	0.8927
max_depth = 25, max_features = 0.9, min_samples_leaf = 2	0.1950	0.1141	0.8926	0.8926
max_depth = 30, max_features = 0.9, min_samples_leaf = 1	0.2048	0.1298	0.8929	0.8929
max_depth = 20, max_features = 0.9, min_samples_leaf = 1	0.2035	0.1197	0.8911	0.8911

K-Nearest Neighbors Hyperparameters	F1-Score on 'deceased'	Recall on 'deceased'	Overall Accuracy	Overall Weighted Recall
n_neighbors = 60, p = 1 (Manhattan), weights = 'distance'	0.1703	0.0996	0.8868	0.8868

n_neighbors = 69, p = 1 (Manhattan), weights = 'distance'	0.1703	0.0996	0.8871	0.8871
n_neighbors = 60, p = 2 (Euclidean), weights = 'distance'	0.1687	0.0984	0.8863	0.8863
n_neighbors = 69, p = 2 (Euclidean), weights = 'distance'	0.1687	0.0984	0.8868	0.8868

From our reports, we can observe that XGBoost had the best overall results. We can observe that the tuned Random Forests model outperforms the tuned K-Nearest Neighbors model in every criterion and then the XGBoost model outperforms the Random Forests model in every criterion.

However, each model does have its own advantages and disadvantages. Since tuning hyperparameters requires us to generate the model again each time, it took us a while to finish tuning our K-Nearest Neighbors model. However, we found XGBoost and K-Nearest Neighbors easier to tune than Random Forests because they provided deterministic results. Since Random Forests is pseudo-random, it would provide a different result that would be slightly better or worse with out without altering a hyperparameter. This made it difficult to judge if altering a hyperparameter slightly truly made a difference to the model's results. Other important things to note is that XGBoost and Random Forests had many different hyperparameters that we could alter, however, K-Nearest Neighbours only had a select few, which may have affected how much we could have optimized the model.

Conclusion

From the tables in the 'Results' section we observed that the model that provided us with the best results for F1-Score on 'deceased' was XGBoost using the following hyperparameters: learning_rate = 0.55, max_depth = 19, and n_estimators = 105.

Regarding runtimes, we found the order from fastest to slowest to be: XGBoost, Random Forests, then K-Nearest Neighbors. Generating each of the tuned models above and taking the average of our trials we observed the following results:

Runtime	Trial 1	Trial 2	Trial 3	Average
Tuned XGBoost	11.6206s	11.5397s	11.7806s	11.6469s
Tuned Random Forests	64.8922s	63.2134s	64.9233s	64.3430s
Tuned K-Nearest Neighbors	116.3070s	118.2159s	116.4605s	116.9945s

Therefore, we conclude that XGBoost is the best model for the goal specified in the 'Problem Statement' section since it provides the best results for F1-score on 'deceased' and it has the best runtime of the 3 classification models.

Alvin Ho
Kenrick Lam
Steven Tran

Prediction on Test Dataset

Since our trained model took in as input a dataframe of all the rows in the training dataset excluding the 'outcome' column, we set up the input similarly for our testing dataset. We then used the model's predict() function on the input which returned an integer array of the different outcomes label-encoded as integers. We then converted that array to strings and used the following dictionary: {'0': 'deceased, '1': 'hospitalized', '2': 'nonhospitalized', '3': 'recovered'} for string replacement. We then saved this output as 'predictions.txt' with each row *i* in the file having a single outcome string which is associated with the original data object's 'outcome' result at row *i* from cases_train.csv.

Lessons Learnt and future work

We learned valuable lessons in data exploration, as for all of us this was our first experience working with large datasets and extracting information from them. We learned the necessary requirements for data to be useful, such as ensuring no data is empty and that some columns require conversions to certain data types so that it can be used in later processing tasks. From this we gained experience in cleaning data albeit in a very rudimentary way. In the future, we should look for a more automated way to clean data from columns, as opposed to manually searching for unique values within the rows with invalid formats and replacing them.

We also gained a lot of experience working with classification libraries and finetuning them. We found that adding just one more hyperparameter made the model selection run for much longer. When trying to run GridSearchCV with 3 hyperparameters, it took an extremely long time to run, even when using all computer threads concurrently. In the future, it would possibly be faster to only tune with 2 hyperparameters to begin with, and then manually test the third hyperparameter.

References

[1] The Ultimate Guide to AdaBoost, random forests and XGBoost [online]
Available: https://towardsdatascience.com/the-ultimate-guide-to-adaboost-random-forests-and-xgboost-7f9327061c4f

[2] Decision Trees and Random Forests [online]
Available: https://towardsdatascience.com/decision-trees-and-random-forests-df0c3123f991

[3] Comparative study on the classic machine learning algorithms [online]
Available: https://towardsdatascience.com/comparative-study-on-classic-machine-learning-algorithms-24f9ff6ab222

Alvin Ho
Kenrick Lam
Steven Tran

CMPT 459 Course Project Milestone 3

Contributions

Alvin Ho

Milestone 1: handling outliers, transforming and joining data

Milestone 2: K-Nearest Neighbors, report

Milestone 3: Tuning K-Nearest Neighbors, report

Kenrick Lam

Milestone 1: exploratory data analysis, handling outliers, outcome labels

Milestone 2: Random Forests, accuracy and classification report functions, report

Milestone 3: Tuning random forests, confusion matrices, report

Steven Tran

Milestone 1: imputing data, cleaning ages, transforming and joining data

Milestone 2: XGBoost, accuracy and classification report functions, report

Milestone 3: