CMPT 459 FALL 2020
FINAL REPORT

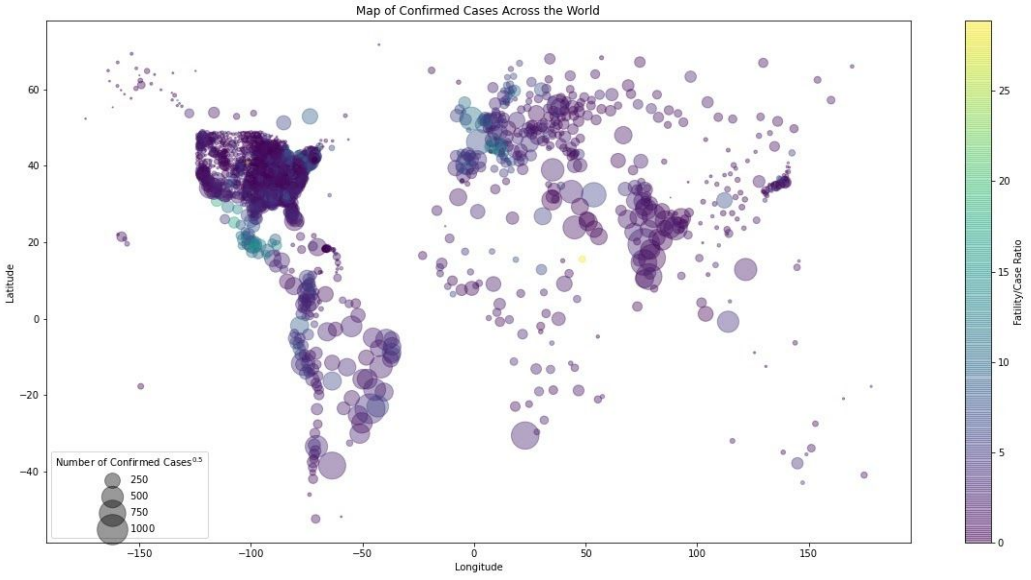
Group 20
Leo Chen 301276105
Gerland Lok 301260310
Parsa Alamzadeh 301316272
Simon Fraser University

Problem Statement

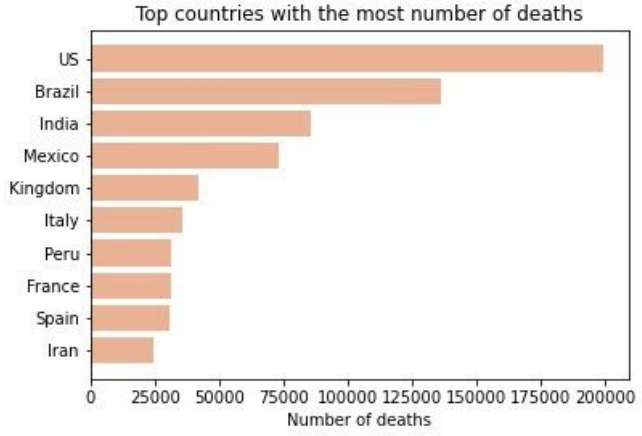
The goal of this project was to train different classifiers we could use to accurately predict the fate of a patient who has tested positive for COVID-19 whether that is future recovery, hospitalization, no hospitalization or ultimately succumbing to the disease. Of the different classifiers we trained and later tuned, we wanted to know which classifiers would perform the best based on a variety of metrics including accuracy and recall. Another goal of this project was to explore methods for preprocessing datasets before training as it's commonplace for records to be missing attribute values. This goal is perhaps the most important if not the least exciting as it directly affects the quality of our classifiers down the line. If we do a subpar job in preprocessing our data, no matter how well or clever we train our classifiers, it'll always produce subpar results. As established data scientists would say: rubbish in, rubbish out.

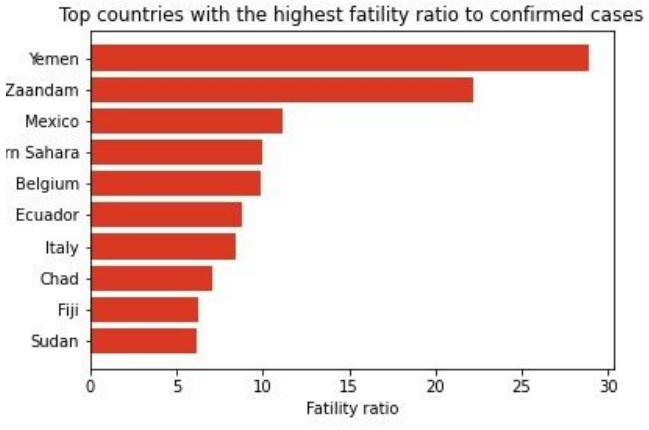
Dataset description and EDA

In our exploratory data analysis, we plotted raw data from our Location dataset to observe interesting trends such as the concentrations of COVID-19 cases around the world.

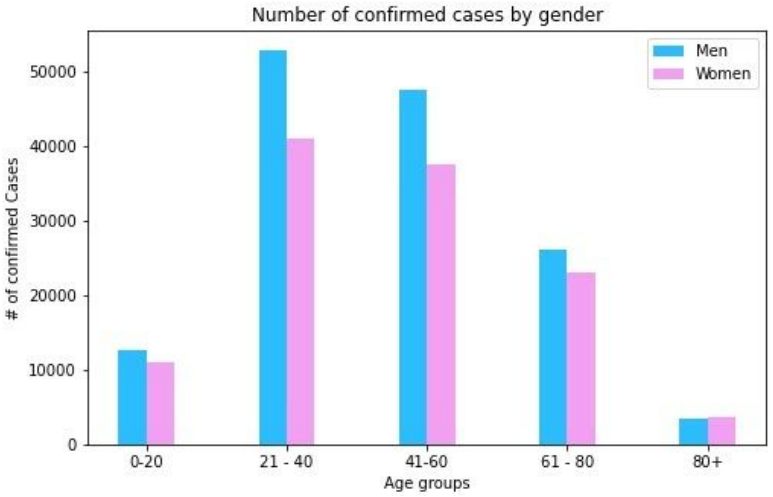


Analysis of the missing values in the datasets was also performed in preparation for how we were going to approach our data preparation and outlier removal. For instance, in the Location dataset, we removed 80 records where longitude and latitude values were missing. This analysis also gave us interesting insights on which





countries have the highest number of deaths and the highest fatality ratio among cases. When looking at the Individual Cases dataset, we plotted the cases based on age range and gender to visualize the impact these two factors had in terms of being infected by COVID-19.



Data preparation

We performed age conversions on the Individual Cases dataset. For age ranges, we used the mean of the upper and lower bound to represent the age of the patient. For ages where we only have a lower bound (Ex. age 80+), we used the base age to represent the patients' age. For patients where their age was represented in months, we converted those into years. For dates, they were represented as "dd.mm.yyyy". Some were represented as "dd.mm.yyyy - dd.mm.yyyy" and only the initial date was kept. The rest of the dates were converted to DateTime objects.

We determined outliers based on insights from our exploratory data analysis. For location data, we dropped records that had empty longitude and latitude data. We found that these rows were usually cruise ships or unknown locations. For Individual Cases, we also dropped records with impossible latitude and longitude values as well as missing country values. For some countries, we also removed records where the province was "Recovered".

After both datasets were cleaned and their outliers were removed, we consolidated data on the Location dataset by aggregating the information for cases in the US to the State level. This was done because only for cases in the United States did we have data with granularity to the county level. Other countries only data that was granular to the state or province level. To do this, we had to develop a way to aggregate the different attributes in a meaningful manner. For the latitude and longitude coordinates, we decided to aggregate these values by using the Weighted Mean Centre formula. This would give us the central longitude and latitude of all confirmed cases of COVID-19 in every state.

In our exploratory data analysis, we discovered that except for cases reported on cruise ships which were removed as outliers, every US county's COVID-19 numbers were last updated on the 20th of September 2020 at 04:22:56. Therefore, for every state, the aggregate Last_Update value was set to 2020-09-20 04:22:56 accordingly. The Combined_Key was aggregated to the form **<State, US>**. The Province_State and Country_Region attributes were kept as-is.

For Confirmed cases, Deaths, Recoveries, and Active cases attributes, these values were summed for each state from their respective counties. For Incidence Rate, we reverse-engineered the formula to re-evaluate these values for the state level. This formula required the use of the population value. As a result, we had to use the incidence rate formula to calculate the population of each county which is as follows:

Population = (Confirmed cases x 100k) / Incidence Rate of County

After summing up the population and the number of confirmed cases from each county to get numbers at the state level, we then used the following formula to find the Incidence Rate at the state level:

State Incidence Rate = (Total Confirmed Cases x 100k) / State population

Case Fatality Ratios were recalculated to the state level using the formula below. In the formula, “Total Deaths” was represented by the total number of deaths across every county in the state and the “Total Confirmed Cases” is represented by the total number of Confirmed Cases found previously. Once this aggregation was complete, we exported the resulting dataset and referred to it later as the US_State_aggregate.

Case Fatality Ratio = (Total Deaths / Total Confirmed Cases) x 100

When it came to merging the Individual Cases and Location datasets, we needed to aggregate the location dataset one more level up before merging. And that level would be the country level because, during our exploratory data analysis, some records in the Individual Cases dataset were missing their state attribute values. We performed for each country, except for the United States, the same aggregation we did for going from US counties to the US State level. Except for this time, we aggregated from the state to the country level. Once we had these aggregations exported to a dataset, we would call country_aggregate. Then we were ready to merge. We started by filtering out records in the Individual Cases dataset that met any of the following conditions:

- If the patient resided in the United States.
- If the patient record was missing the State_Province attribute value.
- If the State_Province and Country_Region the patient resided in was not in the Location Dataset.

We’ll call the resulting filtered out dataset our working dataset. We’ll then begin the first phase of the merge by performing a left join based on the Province_State and Country_Region values where the Individual Cases dataset was the left table and the Location dataset was the right table. Next, we addressed the filtered out records from the Individual Cases dataset where Province_State attribute values were missing or the Province_State and Country_Region combination was not present in the Location dataset. For these records, we performed a left join based solely on Country_Region values where the filtered out dataset was the left table and the location_aggregate we generated earlier was the right table. Finally, for individual cases where the patient was in the United States, we performed a left join for those records based on both Province_State and Country_Region values where the Individual cases dataset was the left table and the US_State_aggregate dataset we generated earlier was the right table.

Once that was all complete, we merged the working dataset and the filtered out dataset together to complete the merge. We think this is an appropriate approach to merge the Individual Cases and Location datasets as each record now has applicable location statistics of the highest granularity possible. Meaning that if a person resides in a State or Province we had statistics for, they will have that data associated with their record. But if they resided in a State or Province we didn’t have

data for, then they would at least still have their Country aggregate statistics. For US cases, since County information was not recorded in the Individual Cases dataset, the highest granular location statistics we had available was to the US State level, which is why for US cases, those statistics were used.

As for imputing the data, we used sklearn's built-in multivariate imputation method, which is a round-robin iterative algorithm. For each step it recognizes a variable as its output and considers the rest of the row as inputs, it then tries to fit a regressor to predict the values of the "output" which in this case are the missing values. Based on our experiments, this method performed way better than the classical single variable imputation (either taking the mode, mean or the mode). The reason for that is the algorithm considers all the other variables when imputing the missing data and since it is an iterative method, it will loop through all the data points and adapt its imputation.

Classification models

The three Classification Models we chose to use were Neural Networks (NN), Adaboost, K Nearest Neighbours (KNN). Our merged dataset would be used for both training, testing, and cross-validation.

We went with Adaboost mainly because it was one of the requirements of Milestone 2. However, notwithstanding this requirement, we still would've chosen this method as Adaboost has been very successful in many other medical field applications and has been proven to be easy to implement while still having low bias and providing great results.

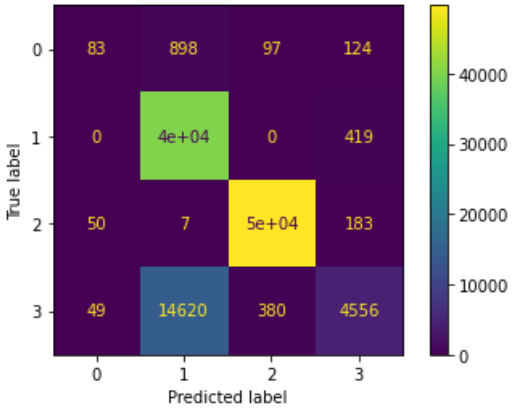
The reason we decided to go with NN was that we assumed that the machine can distinguish between the relevant and irrelevant features while maintaining high accuracy and recall. We also assumed that it is going to be one of the best if not the best model for this task due to its capabilities and how it's been used nowadays in every imaginable task.

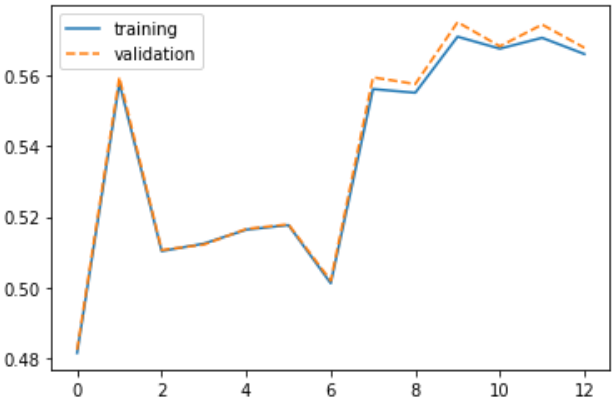
We decided on K-Nearest Neighbors as a third algorithm to see how it would perform considering it does not have a training phase nor assumption of the underlying data. This makes KNN unique and low bias. It is interesting how a generally simple and greedy algorithm would compare to some of the more advanced machine learning models we had.

Initial evaluation and overfitting

Neural Network

The best performing model from the previous milestone had an accuracy of 84.84% on the training dataset and 84.90% accuracy on the validation dataset. We did not notice any overfitting with this model. We kept the epochs very small to avoid this problem. Although this method had an acceptable overall accuracy, it fell short when it came to classes with small occurrences in the dataset (deceased and recovered patients). For those, the model had an F-2 score of 0.083 and 0.272 accordingly. Below is our Confusion matrix and accuracy of training vs validation:

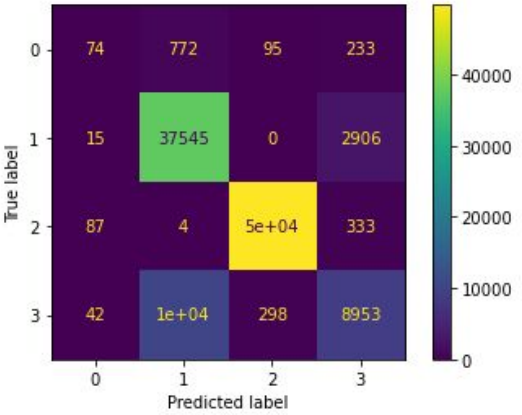


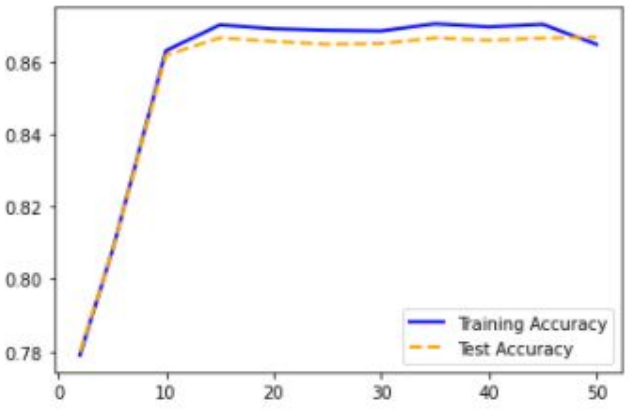


Adaboost

The overall accuracy of the Training set for our original AdaBoost classifier was approximately 86% while the accuracy of the Test set was 87%. When examining the Confusion Matrices for the test set, Deceased patients classification accuracy was only 6% and the accuracy for Recovered patients was roughly 46%.

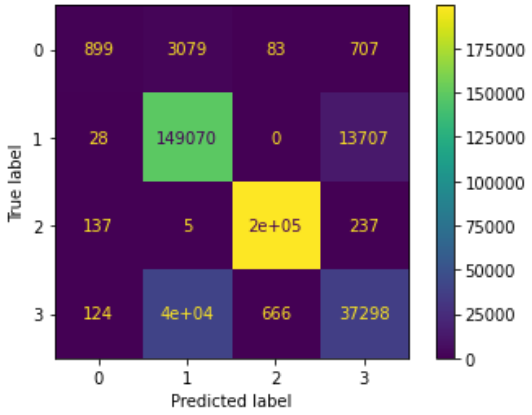
Overfitting was not an issue with our Adaboost Tree as both Training and Test accuracies plateaued and remained constant as the maximum size of the base Decision Tree classifier increased. F2-Score for the Test set was 0.6423 and 0.6384 for the Training set.

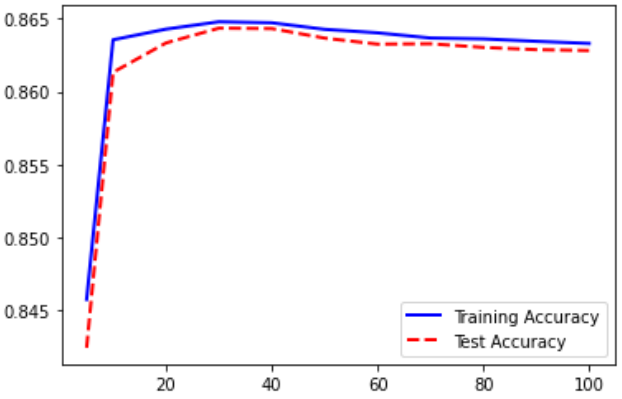




K Nearest Neighbours (KNN)

For KNN, the overall accuracy for training data was around 86.4% while the test accuracy was 86.3%. There is no overfitting but we found the optimal accuracy was best around K=40 with our initial testing from Milestone 2. The F-2 Score for training data was around 0.6493.





Hyperparameter tuning

Neural Network Tuning

Since for implementing the neural network we used PyTorch, we were unable to use any of the sklearn available functions for doing hyperparameters optimization. We decided to use TPE (Tree-structured Parzen Estimator) sampler for our search method, which is a bayesian based optimization method that takes advantage of its tree structure to prune unnecessary searches. The hyperparameters that we decided to tune the first layer size (20 to 250), the second layer size (1 to 200) and the learning rate (1e-5 to 1e-1). As for the k-fold cross-validation, it was done manually and we used StratifiedKFold to create 4 folds and ran our evaluation code on each of the folds. All models were trained for 20 epochs and due to the limited time and lack of hardware resources, we only ran the search for 150 iterations (5+ minutes for each iteration and there was no significant speedup between using CPU and GPU for neural network computation since most of the time the resources had to be transferred from RAM to VRAM for GPU).

Adaboost Tuning

We used GridSearchCV to tune the various hyperparameters for Adaboost. The two main ones we were interested in was the learning rate and the n_estimators. Due to time constraints and the lack of access to powerful hardware to run more extensive tuning, the max_depth of the Decision Tree classifier on which Adaboost is based was not tuned. It was discovered during our initial manual tuning during Milestone 2 that up to a certain depth, further increasing the tree depth past a value such as 10 made no difference to the model. As a result, to conserve resources, we thought opting this hyperparameter out was justified. For future research, however, tuning of the max_depth should be explored in addition to further tuning both the learning rate and n_estimators.

KNN Tuning

For KNN, we used GridSearchCV to do cross-validation and hyperparameter tuning. We found that the best parameters that affected KNN were the number of nearest neighbours and weight (distance, uniform). The distance formula during cross-validation did not have any impact on the overall results compared to Milestone 2. Distance weight means that points closer to the point will have higher weight than those farther away. For the number of nearest neighbours, Milestone 2 told us that the ideal range was between K=38 to K=44 so we were able to narrow down numbers in that range.

Results

Best performing Hyperparameters for classifying Deceased patients highlighted in yellow.

AdaBoost Tuning

Hyperparameters	Accuracy	Overall Recall	Recall on 'Deceased'
{'learning_rate': 5, 'n_estimators': 100}	0.3501563772	0.3501563772	0.3940010007
{'learning_rate': 10, 'n_estimators': 10}	0.1244589969	0.1244589969	0.3438333333
{'learning_rate': 5, 'n_estimators': 50}	0.1244589969	0.1244589969	0.3438333333
{'learning_rate': 5, 'n_estimators': 100}	0.05536392527	0.05536392527	0.2495
{'learning_rate': 5, 'n_estimators': 50}	0.3527805694	0.3527805694	0.2201738937

KNN Tuning

These are the top 5 best performing KNN hyperparameters for recall on 'deceased.' There are 2 parameters tied for 1st place so there are 6 results.

Hyperparameters	Accuracy	Overall Recall	Recall on 'Deceased'
{'metric': 'euclidean', 'n_neighbors': 38, 'weights': 'distance'}	0.851256557734645	0.5896169828197	0.0291723371136313
{'metric': 'euclidean', 'n_neighbors': 40, 'weights': 'distance'}	0.851586687127842	0.5895038196814	0.0290056704469646

{'metric': 'euclidean', 'n_neighbors': 42, 'weights': 'distance'}	0.855083607351424	0.5898029023231	0.02900555926173
{'metric': 'minkowski', 'n_neighbors': 38, 'weights': 'distance'}	0.851256557734645	0.5896169828197	0.0291723371136313
{'metric': 'minkowski', 'n_neighbors': 40, 'weights': 'distance'}	0.851586687127842	0.5895038196814	0.0290056704469646
{'metric': 'minkowski', 'n_neighbors': 42, 'weights': 'distance'}	0.851278082821694	0.5898029023231	0.02900555926173

Neural Network Tuning

Below are the top-5 performing (based on deceased recall) architectures. The full table is available at nn_trials.csv.

Hyperparameters	**Accuracy (4-fold cross-validation)**	**Overall Mean Recall**	**Recall on 'Deceased'**
First layer = 48 Second layer = 182 Lr = 0.000848	0.65327	0.447990	0.031207
First layer = 25 Second layer = 189 Lr = 0.000673	0.61685	0.385607	0.023834
First layer = 31 Second layer = 168 Lr = 0.002349	0.59907	0.382504	0.023534
First layer = 37 Second layer = 171 Lr = 0.000522	0.67862	0.411697	0.022636
First layer = 20 Second layer = 169 Lr = 0.000411	0.70008	0.448781	0.021669

Comparative study

As you can see, the accuracies have dropped significantly for both Neural network and Adaboost, while KNN maintained a similar accuracy. As for the f2-scores (which we used primarily to evaluate the models) we saw a 40% drop in f2-score for neural network from 0.65 to 0.42; for knn the score stayed very similar and for Adaboost we saw a similar drop in the performance (0.6 to 0.32). This shows that if we prioritize the recall on deceased cases, our accuracies and f2-scores would suffer as a result for Adaboost and NN. Adaboost performed the best when it comes to recalling deceased cases but overall accuracy suffered, while NN failed to improve its deceased recall, its accuracy also dropped. On the other hand, KNN did not improve its recall on deceased cases by much but its accuracy stayed steady. For overall accuracy we recommend Adaboost if we are focused on accuracy, but it also can be optimized to perform better for deceased cases, while the other two failed to make any significant improvements for both.

Conclusion

Based on our results, we concluded that the Adaboost model performed the best when it comes to maximizing the recall for deceased cases. This high recall value for deceased classifications comes with some sacrifices, one of which is the huge loss in overall accuracy. While KNN yielded a lower recall compared to other methods, its overall accuracy did not suffer as much. In contrast, the neural network performed better compared to KNN when it came to the recall for deceased cases but its accuracy took a big hit and the model did not seem to be viable with such a low overall mean recall and accuracy when cross-validation was performed. We reckon that based on our results, that the best way to predict the outcome of a case is to use two separate models based on Adaboost; where one is for predicting whether or not a patient will be hospitalized or not, and another for predicting if the patient will recover or not. This is because it appears there is an inverse relationship between high recall for the “deceased" classification and overall model accuracy. So for future research, a two-tier classifier that predicts if a patient will be hospitalized and if a patient will recover could prove to be superior to the models we have demonstrated here. And such a model would still be valid as being hospitalized and a patients’ outcome in their fight against a COVID-19 infection are disjoint. Not all COVID-19 patients who recover or ultimately succumb to the virus are hospitalized. Many COVID-19 patients who recover do so not in a hospital bed, but in their own homes.

Lessons learnt and future work

Contrary to popular belief, the classical (statistical methods) are just as effective as newer ML-based models if not more. Adaboost as an example is very fast to train and easy to tune; whereas a simple neural network can have infinitely many hyperparameters especially when it comes to the structure of the layers and how many neurons each could take. We also noticed that performance-wise, they are not even close. Adaboost had better overall accuracy and in our last milestone, the

f-score for all outcomes was higher compared to the best performing neural network. We saw a similar pattern with KNN as well. KNN had better overall accuracy and better f-score compared to the NN. We also learned that data pre-processing is an integral part of our data mining and some of our data attributes (such as country-region key and days since the first confirmed case) threw off our models due to the magnitude of the numbers. We also should have evaluated the mutual information for each of the attributes compared to the outcomes and should have eliminated attributes that did not yield high mutual information. Another improvement we could have made was testing the effects of increasing the layers in the model rather than changing the size of each hidden layer.

Contributions

Both Leo and Parsa contributed to the exploratory data analysis, data cleansing, preprocessing, imputing data and outlier removal in both the Individual Cases and Location datasets. Gerland contributed to the Country and US State aggregation on the Location Dataset as well as the merging of the Individual and Location datasets. He was also responsible for the creation and tuning of the Adaboost Models used. Leo was responsible for the creation and tuning of the KNN Models used. Parsa was responsible for the creation and tuning of the Neural Networks used.