CMPT459

Project Report

Mohammad Raad Sarar 301326232
Kazi Dhrubo 301325924
Ahmed Irteza Haque 301325980

COVID Outcome Predictor

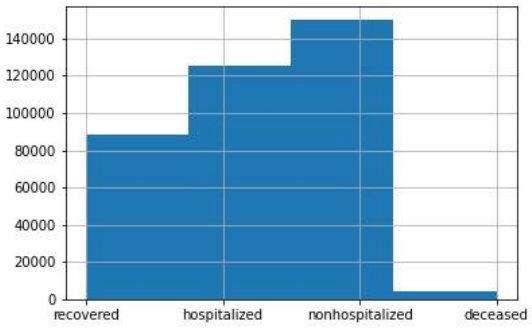
Github Repository: https://github.com/zubayerkader/CMPT459_Project.git

Problem Statement

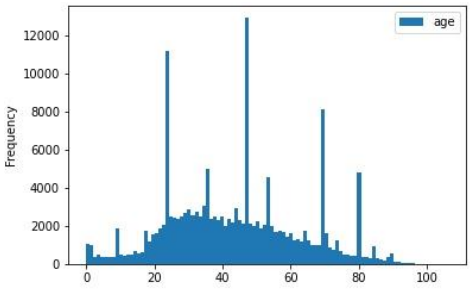
The goal of this project is to predict the fate of an individual after they contract the Coronavirus, given their age, sex, location, date of confirmation. The program uses of publicly available Covid dataset and machine learning models from python sklearn library to predict a possible outcome from: *recovered, hospitalized, non-hospitalized, deceased.*

Data Description & Exploratory Data Analysis

All the attributes in *cases_train.csv* and *location.csv* were read and each attribute was analyzed independently of each other (except for latitude and longitude). The dataset contained a lot of missing values. Additionally, for each numerical attribute, count, mean, standard deviation, min, max and the interquartile ranges were explored along with histograms to understand their distribution. Box and whisker diagrams were also generated to find outliers. For latitude and longitude values, scatterplots were printed and overlayed over an image of the world map to make sure the values are accurate. For each categorical attribute, histograms were created to understand the distribution of the unique values.



Distribution of Outcomes



Distribution of Age

Note: The distribution of outcomes was skewed as the number of deceased entries was much smaller than the other outcomes. The age distribution represented somewhat of a bell curve with mean around 35.

Columns in location.csv (case fatality ratio, recovered, active, incidence rate, deaths) were very skewed and could contain outliers. The active column also contained negative numbers. Rows with negative numbers were corrected first and then statistics and visualizations were created.

Data preparation

Data cleaning and Imputing missing values

We removed only 2 rows from the *cases_train.csv* where latitude and longitude values were not present. Columns *additional_information* and *source* were dropped as we believe the text data would not be of much help during the classification phase as a lot of data was missing. Cleaning and imputing methods are described below:

Age: The values in the form of (*x-y*) were found by python regular expression and the average of x and y was assigned to the age. The values like *z-months* were found by python regular expression and replaced by *z/12*. The values like *a+* or *a-* were also found similarly and replaced by *a*. To **fill in missing values,** a normal distribution was created of the values that were not missing and age was assigned according to the probability determined by the mean and standard deviation of the normal distribution *(np.random.normal(age_mean, age_std, 1))*.

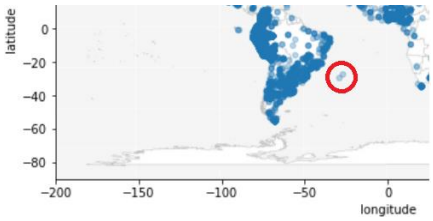
Sex: Before imputing the sex values, we figured out how many male and female values that were not missing, we created a probability of male and female according to the counts and imputed the values according to that probability.

Country: We found that only a few rows (18) with attribute country was missing. So, we printed them all. They all had province as “Taiwan”. So, we assigned “Taiwan” as the country (not sure if Taiwan or China should be set as the country, Google was a little unclear) of the rows that had country value missing but had Taiwan as a province.

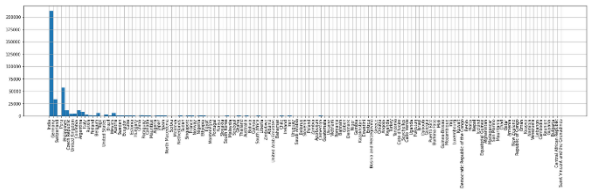
Province: For rows that did not have a province, we checked the country (as after the previous step all rows have a country value), and found the province with the highest occurrence (*mode*) of that country and assigned it as province. Some countries did not have any provinces listed, in that case finding the mode was not possible. So instead of leaving the province field empty or removing the entire row, we decided to replace the Null value with *“unknown”.* eg. (province: unknown, country: Togo).

Date Confirmation: We used a similar approach to the *province attribute* as we assumed countries have release their date of confirmation in a large batch. So, for missing date confirmation, we found the country first and then found the mode of the date confirmed of that country and imputed with that value.

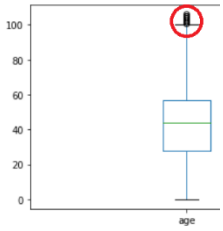
Dealing with outliers



Latitude & longitude: The scatterplot that we generated of the latitude and longitude showed some points located in the sea (image to the right). We filtered those points (using latitude and longitude thresholds) to see what country and provinces they belonged to. We found that the points belonged to Rio Grande do Sul and Santa Catarina, Brazil. So, we found all the data entries with province = Rio Grande do Sul and country = Brazil and took the average latitude and longitude values and replaced them with the average values.



Provinces & Countries:
The data for the countries and provinces and countries were vastly skewed with India having over 200,000 records (over 50% of the dataset). Some countries only had one data entry (row). We could have considered them to be outliers, but we decided not to. This is because we realize that even if we have one row for a country, it significantly increases the chance of classifying a row in the test file if that country shows up.



Age: In our box and whisker plot we saw some age values over 100 marked as outliers (plot on the right). But we decided not to change them as it would give us a more diverse dataset while training the model. It is also feasible to believe that people do live over 100 years and this is an age group whose data cannot be not easily available. So, we decided to keep this group of data without changing the values.

Location.csv: We noticed a lot of attributes with outliers in location.csv. Case fatality ratio, Recovered, Active, Incidence rate, and Deaths were all really skewed where a few locations had most of the data while the other locations had barely any data. The active column also contained negative numbers. Rows with negative numbers were corrected first and then statistics and visualizations were created.

Transformation of Location Dataset

To transform the location dataset, a new dataframe, US, was created where Country_Region == US. The US dataset was then grouped by Province_State and each column was aggregated in the following way:

Latitude & longitude, Incidence_Rate: mean of aggregation
Confirmed, Deaths, Recovered, Active: sum of aggregation
Case-Fatality_Ratio: aggregated Deaths/aggregated Confirmed*100

All the data with Country_Region == US is removed from the original location dataset and the newly created dataset, US, is appended to the original dataset and then sorted by Country_Region to get a transformed location dataset.

Merging Datasets: Cases and Location

Initially we join cases and location using composite keys (province, country) for each dataset. We got a left joined dataset which had a lot of NaNs. This is because the cases dataset was complete but the location dataset had a lot missing provinces. We separated the rows with NaN values and calculated the closest province to that row using the latitude and longitude values. Calculation of closest province was done by comparing distances of each row in cases dataset to all rows in locations dataset to find the smallest distance.

Classification Models

The models chosen are: ***K-nearest Neighbors, AdaBoost,*** *and* ***Random Forest***

Models Not Chosen:

Naive Bayes was not chosen as it assumes that attributes are conditionally independent given a class. But in our dataset many attributes are dependent on each other. For example (longitude and latitude) and (Confirmed, Recovered, Active, Incidence Rate, Case-Fatality Ratio).

Decision Tree was not chosen as Random forest is a better estimator of the decision tree.

SVM assumes a linear plane separating the datapoints. We believe that a dataset of 367635 datapoints with 14 attributes would be quite difficult to separate using a linear decision surface unless the added complexity of the kernel method is implemented which would reduce the interpretability of our model.

Models Chosen:

K-nearest Neighbors: Since KNN uses a distance function to calculate other data points that are closest to a given point we believe data point that are similar will have the same outcome. Similar locations, age group and sex are likely to have similar outcomes. This hypothesis was also the reason why we assigned weights that are inverse to the distance, i.e. distant objects will have less weight.

Since KNN depends on distances, we scaled the data so that all attributes contribute equally to the distance calculations. Columns province and country were dropped as label encoder would cause inconsistency in distance calculations. This will not be a

problem because latitude and longitude columns will serve the purpose of locating geographically nearby points. Also, the sex column is one hot encoded.

Random Forest: Decision tree split attributes that have the highest Information Gain/Gini Index which result in more accurate and sophisticated decision surface. These are desirable properties however, decision trees have high variance and are likely to overfit. Therefore, we use Random Forest Classifier which is a voting-based classifier that reduces the variance and results in more comprehensive classifications.

AdaBoost: We know that decision trees have some desirable properties as mentioned above. AdaBoost uses Decision Trees as its base classifier and fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases. https://scikit-learn.org/

Pre model fitting Transformation:

For all classifiers, the date_confirmed column was converted to hours from January 1st, 2020 so that the column contained numerical data.

For Random forest and AdaBoost, categorical columns (Sex, Province and Country) were labeled using sklearn's label encoder as these classifiers do not rely on distances.

Initial Evaluation and Overfitting:

Evaluation was done while parameter tuning to detect overfitting.

We varied only one hyperparameters for each base classifier and calculated training and validation accuracies. Then we plotted a graph of train/validation accuracies against the hyperparameter values. For different hyperparameter values we also recorded the precision, recall and confusion matrix for train and validation datasets.

n_neighbors	training_score	validation_score	training_confusion_matrix	training_precision	training_recall	validation_confusion_matrix	validation_precision	validation_recall
1	0.911348591	0.769404436	[[2376 748 12 493]	0.835756485	0.838710815	[[64 443 68 295]	0.574420504	0.575425938
2	0.912079617	0.773552573	[[2888 608 1 132]	0.823500236	0.865082802	[[86 489 68 227]	0.577895807	0.577550357
3	0.921980776	0.779196758	[[2660 667 2 300]	0.837466771	0.864389707	[[67 456 64 283]	0.582930396	0.582450081
4	0.924160255	0.784732139	[[2507 793 2 327]	0.877362447	0.85299694	[[57 499 67 247]	0.590969839	0.581425736
5	0.926594743	0.787017014	[[2507 828 2 292]	0.87589169	0.85733971	[[51 483 68 268]	0.591664289	0.583363393

Figure 1: scores for knn using k=1-5

Using this data for each classifier we plotted the following graphs:

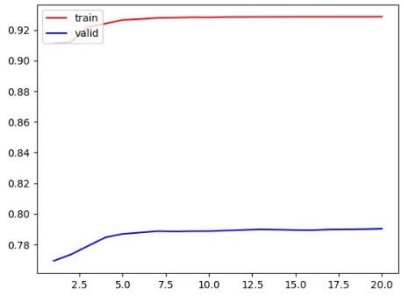


Figure 2: AdaBoost with varying max_depth

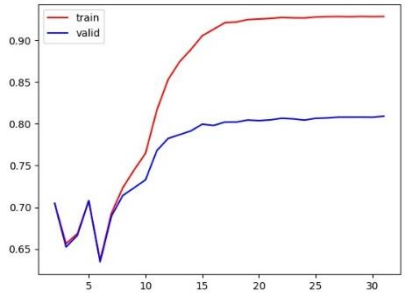


Figure 3: KNN with varying k

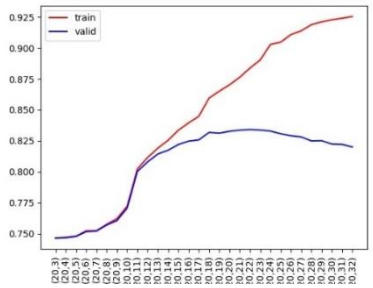


Figure 4: Random Forest with varying max_depth

In our training and validation graphs we looked for points where the validation data stopped increasing or started decreasing to determine the best initial hyper-parameter. For overfitting detection, we looked for points in the graph where the validation score no longer increases but starts decreasing. For the random forest we observed slight overfitting after max_depth = 19 (figure 4). For KNN we did not observe overfitting (curve flattening after k=7 in figure 3) because we used the inverse of the distance as the weights which means the more distant a neighbor is the lower its impact on the classification of a point. AdaBoost also did not show overfitting as the curve just flattened after max_depth =16 (Figure 2) the initial spikes are probably caused as the max_depth is too low to classify correctly.

Evaluation Metrics of the best Hyperparameter for each base classifier:

Classifier	Best Hyperparameter	Training Accuracy	Validation Accuracy	Validation Precision	Validation Recall
Random Forest	Max_depth =19	87%	83%	71%	61%
KNN	K= 7	93%	79%	60%	59%
AdaBoost	Max_depth =16	91%	80%	61%	59%

Hyperparameter Tuning using Grid Search

The strategy adopted by our team is Grid Search as it is an exhaustive method. The advantage of using grid search is that it considers all different hyperparameter combinations to ensure the best one. The disadvantage of this method is it takes a long time to complete as models need to be fitted for each hyperparameter combination. The tuning part of our program takes about 8 hours to run.

It is crucial to maximize the recall for class deceased as predicting deceased is a sensitive matter. To re-enforce this idea, the evaluation criteria to find the best model is set to be f1 score of deceased which considers the recall and precision score of deceased.

Scoring metrics

- Overall accuracy
- Overall recall
- Overall precision
- Recall of deceased class label
- F1 score of deceased class label

Hyperparameter range selection

K-Nearest-Neighbors:

- n_neighbors: from 2 to 15 with increment of 1
- weights: distance and uniform

- p: Manhattan (1) and Euclidian (2)

Random Forest:

- n_estimators: from 2 to 30 with an increment of 2
- max_depth: from 2 to 30 with an increment of 2

ADA boosting:

- base estimator: Decision tree classifier
- base estimator max depth: from 2 to 30 with an increment of 2
- base estimator criterion: gini and entropy
- ADA boost n_estimator: from 10 to 60 with an increment of 10

Results

After training with all possible hyperparameters for each classification algorithm using grid search, very large tables were formed. In order to eliminate unnecessary information, the best 3 rows of each scoring metric (test_accuracy, test_recall_overall, test_f1_deceased, test_recall_deceased, test_precision) were selected and put into a dataframe dropping any duplicates. These table are shown below for each classifier:

Random Forest

params	test_accuracy	test_recall_overall	test_f1_deceased	test_recall_deceased	test_precison
{'max_depth': 24, 'n_estimators': 2}	0.812860071	0.812860071	0.098712552	0.081128167	0.807782568
{'max_depth': 28, 'n_estimators': 2}	0.806584809	0.806584809	0.098477183	0.09268743	0.802704452
{'max_depth': 22, 'n_estimators': 2}	0.814494863	0.814494863	0.097633958	0.072682487	0.808519461
{'max_depth': 26, 'n_estimators': 2}	0.80955244	0.80955244	0.095393518	0.084682734	0.805212228
{'max_depth': 22, 'n_estimators': 22}	0.834231882	0.834231882	0.074286432	0.041563466	0.828535355
{'max_depth': 22, 'n_estimators': 26}	0.833856506	0.833856506	0.074018496	0.041563713	0.827987423
{'max_depth': 22, 'n_estimators': 28}	0.83441413	0.83441413	0.069467549	0.038896057	0.828429148
{'max_depth': 20, 'n_estimators': 22}	0.833402252	0.833402252	0.064315272	0.034894821	0.828399571

While the accuracy was found to me the highest (0.8344) for the parameter {‘max_depth’: 22, ‘n_estimator’: 28}, the highest f1 score was given by the parameter {‘max_depth’: 24, ‘n_estimator’: 2}. We also notice that as the number of n_estimators increase, the accuracy increases but f1 on deceased decreases.

Overall recall score was high but recall score for deceased label was low because the model predicted a lot of false negatives when calculating recall for deceased.

Highest recorded f1 score for deceased is 0.0987

KNN

params	test_accuracy	test_recall_overall	test_f1_deceased	test_recall_deceased	test_precison
{'n_neighbors': 2, 'p': 1, 'weights': 'uniform'}	0.774123182	0.774123182	0.089070381	0.131583488	0.7782137
{'n_neighbors': 2, 'p': 2, 'weights': 'uniform'}	0.771694134	0.771694134	0.086833661	0.127805463	0.775266443
{'n_neighbors': 4, 'p': 1, 'weights': 'uniform'}	0.789113633	0.789113633	0.083207419	0.054454579	0.780660894
{'n_neighbors': 2, 'p': 2, 'weights': 'distance'}	0.773426836	0.773426836	0.074785024	0.09224348	0.772502725
{'n_neighbors': 12, 'p': 1, 'weights': 'distance'}	0.79351475	0.79351475	0.073281456	0.055566926	0.787207266
{'n_neighbors': 13, 'p': 1, 'weights': 'distance'}	0.793626274	0.793626274	0.072356009	0.054678037	0.787267618
{'n_neighbors': 14, 'p': 1, 'weights': 'distance'}	0.793838441	0.793838441	0.071970812	0.054233346	0.787395293

With n_neighbors=2, we get the highest f1 and recall score of deceased. Whereas, the highest accuracy is with n_neighbors=14. Although n_neighbors=2 gives the highest f1 score we believe it might be risky to use only 2 neighbors to make a prediction. As n_neighbors increase, the model can predict other outcomes correctly, but it becomes more difficult to predict the deceased.

Highest recorded f1 score for deceased is 0.0891

Highest recorded recall score for deceased is 0.1316

ADA Boost

params	test_accuracy	test_recall_overall	test_f1_deceased	test_recall_deceased	test_precison
{'base_estimator__criterion': 'entropy', 'base_estimator__max_depth': 16, 'n_estimators': 30}	0.800752937	0.800752937	0.083367936	0.058234087	0.795351026
{'base_estimator__criterion': 'gini', 'base_estimator__max_depth': 18, 'n_estimators': 50}	0.803723274	0.803723274	0.082500861	0.058678285	0.798186841
{'base_estimator__criterion': 'entropy', 'base_estimator__max_depth': 18, 'n_estimators': 50}	0.804384256	0.804384256	0.081869515	0.057788654	0.798878822
{'base_estimator__criterion': 'gini', 'base_estimator__max_depth': 20, 'n_estimators': 50}	0.804990839	0.804990839	0.078529534	0.056233593	0.79950014
{'base_estimator__criterion': 'entropy', 'base_estimator__max_depth': 20, 'n_estimators': 60}	0.805186695	0.805186695	0.078387969	0.055790384	0.799594822
{'base_estimator__criterion': 'entropy', 'base_estimator__max_depth': 20, 'n_estimators': 30}	0.80520301	0.80520301	0.077182458	0.054675565	0.799370549
{'base_estimator__criterion': 'gini', 'base_estimator__max_depth': 8, 'n_estimators': 30}	0.69629303	0.69629303	0.054529792	0.226731183	0.739043774
{'base_estimator__criterion': 'entropy', 'base_estimator__max_depth': 8, 'n_estimators': 50}	0.69866498	0.69866498	0.052873019	0.210053393	0.744099055
{'base_estimator__criterion': 'entropy', 'base_estimator__max_depth': 8, 'n_estimators': 30}	0.699796557	0.699796557	0.051994171	0.210047213	0.742609854

As overall accuracy decreases recall on deceased increases. When max_depth=8, we get the highest recall and the lowest accuracy. The best f1 score was found at {base_estimator__criterion: entropy, base_estimator__max_depth: 16, n_estimators: 30 } however recall deceased was very low.

Conclusion

Pick the best classifier and params from the above tables

Compare performance of the models using standard metrics and report the best fit

Best F1 scores of each model

Classifier	F1 score on deceased	Recall score on deceased	Overall accuracy	hyperparameters
Random Forest	0.0987	0.0811	0.8129	{max_depth: 24, n_estimator: 2}
KNN	0.0891	0.1316	0.7741	{n_neighbors: 2, p: 1, weight: uniform}
ADA Boost	0.0834	0.0582	0.8008	{base_estimator__criterion: entropy, base_estimator__max_depth: 16, n_estimators: 30 }

KNN was not chosen because we did not trust the predictions based on only 2 neighbors. ADA boost was not chosen it had the lowest recall score.

Best Model selected based on f1 score of deceased: Random Forest

Best Hyperparameters: {max_depth: 24, n_estimator: 2}

Prediction on test dataset

Using Random Forest and with hyperparameters {max_depth: 24, n_estimator: 2} found from above analysis, we trained a model from scratch (model.fit(x,y)) using the entire training dataset. We preprocessed the test dataset according to the needs of the classifier as mentioned earlier and predicted the outcomes (model.predict(test_data)) and saved the predictions to a txt file in the required format called predicitions.txt.

Lessons Learnt

- Large datasets are difficult to work with and requires attention to keep it consistent.
- Recall score of minority class labels is usually low when data is skewed
- Data cleaning and imputation requires more work than machine learning
- It is important to perform exploratory data analysis to understand data distribution and identify outliers
- Accuracy score is not always the best measure of performance of a model, it is important to understand the goal of the data analysis and pick appropriate scoring metrics.
- It is important to select hyperparameters such that overfitting does not occur
- Baseline models can be improved upon by hyperparameter tuning

Future Work

Improve f1 score for deceased by:

- Trying more classification algorithm and tune them
- Trying different tuning methods like RandomSearchCV or BayesianOptimization
- Trying more sophisticated data imputation methods

The runtime of our program was a major limitation (8 hours) in experimenting with different parameter of the models using grid search.

References

https://pandas.pydata.org/docs/reference/index.html#api

https://scikit-learn.org/stable/modules/classes.html

https://stackoverflow.com/

https://www.google.com/

Contributions

Members	Milestone 1	Milestone 2	Milestone 3
Mohammad Sarar	- Data Cleaning - Data Imputation - Detecting outliers - Contributed to report writing	- Model choosing brainstorm - KNN model creation, training, overfitting and plotting - Contributed to report writing	- Debugging of dataset joining - KNN Hyperparameter tuning using grid search - Results and conclusion analysis - Contributed to report writing
Kazi Dhrubo	- Transformation location - Joining cases and location - Contributed to report writing	- Model choosing brainstorm - ADA boost model creation, training, overfitting and plotting - Contributed to report writing	- Debugging of dataset joining - ADA boost Hyperparameter tuning using grid search - Results and conclusion analysis - Contributed to report writing
Ahmed Haque	- Data visualization - Contributed to report writing	- Model choosing brainstorm - Random forest model creation, training, overfitting and plotting - Contributed to report writing	- Random forest Hyperparameter tuning using grid search - Results and conclusion analysis - Contributed to report writing