Simon Fraser University

CMPT 459: Data Mining

Data Mining Approach to Analyze Covid-19 Dataset

Project Final Report

Presented By
Andy Wang
Qirui Wu
Liyang Zhou

December 15, 2020

Problem Statement

The outbreak of Coronavirus disease 2019 (Covid-19) started with first cases in December 2019 in Wuhan, China. The high prevalence of Covid-19 has made it a new pandemic across almost all the countries with positive cases. Hence, how to correctly and effectively predict the outcome of Covid-19 throughout the world is not only imperative for researchers to gain a more comprehensive understanding of the disease, but also crucial for the public to realize its potential impact of our life.

In this data mining project, we aim to select some most popular machine learning models, tune different hyperparameters, and use the best tuned models to predict the outcome (hospitalized, non-hospitalized, recovered or deceased) of a case, based on statistics from two publicly available Covid-19 datasets. We will also do a robust evaluation on their performance, specially focusing on correct predictions on the "deceased" cases because they are the most essential.

Dataset Description and Exploratory Data Analysis

The original Covid-19 dataset contains two separate files. All information was extracted and frozen on September 20, 2020. Here is a general overview of each of them. More information is available from their GitHub main pages.

- *Case Dataset*: this contains some personal data (e.g., age) and outcomes for individual cases. (https://github.com/beoutbreakprepared/nCoV2019)
- *Location Dataset*: this contains the number of cases and health statistics based on locations. (https://github.com/CSSEGISandData/COVID-19)

Looking at each of them more in detail, we use tables to summarize some important properties: Data Type, Number of Missing Values and Number of Unique Values for each attribute in "*processed_individual_cases_Sep20th2020.csv*" (case) and "*processed_location_Sep20th2020.csv*" (location). We also use some visualization approaches to show attribute distribution based on its information and data types. The tables and visualizing graphics are as follows.

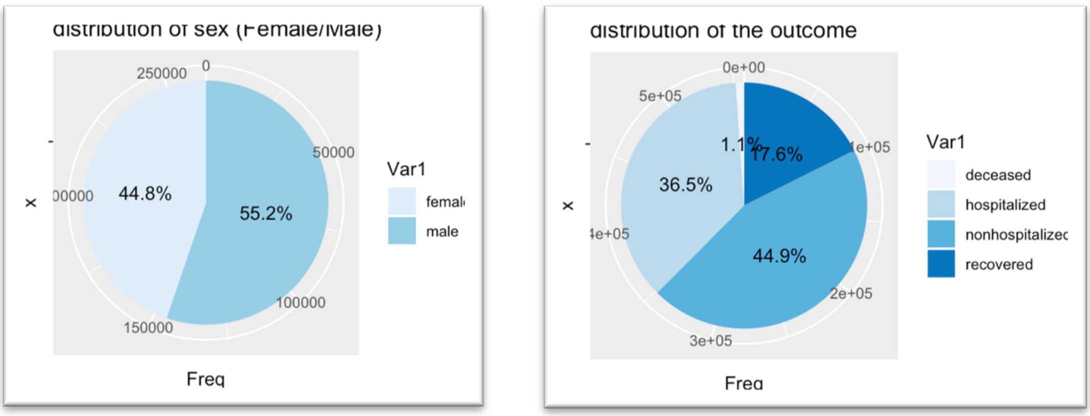
Table 1 – Summary Table of “*processed_individual_cases_Sep20th2020.csv*”

“*processed_individual_cases_Sep20th2020.csv*” (n = 557,364)			
Attributes	**Data Type**	**Number of Missing Values**	**Number of Unique Values**
Age	“char”	296,874	362
Sex	“char”	293,734	2
Province	“char”	6,568	1,180
Country	“char”	24	136
Latitude	“numeric”	2	8,459
Longitude	“numeric”	2	8,454
Date_confirmation	“char”	462	172
Additional_information	“char”	522,969	20,689
Source	“char”	209,191	9,597
Outcome	“char”	0	4

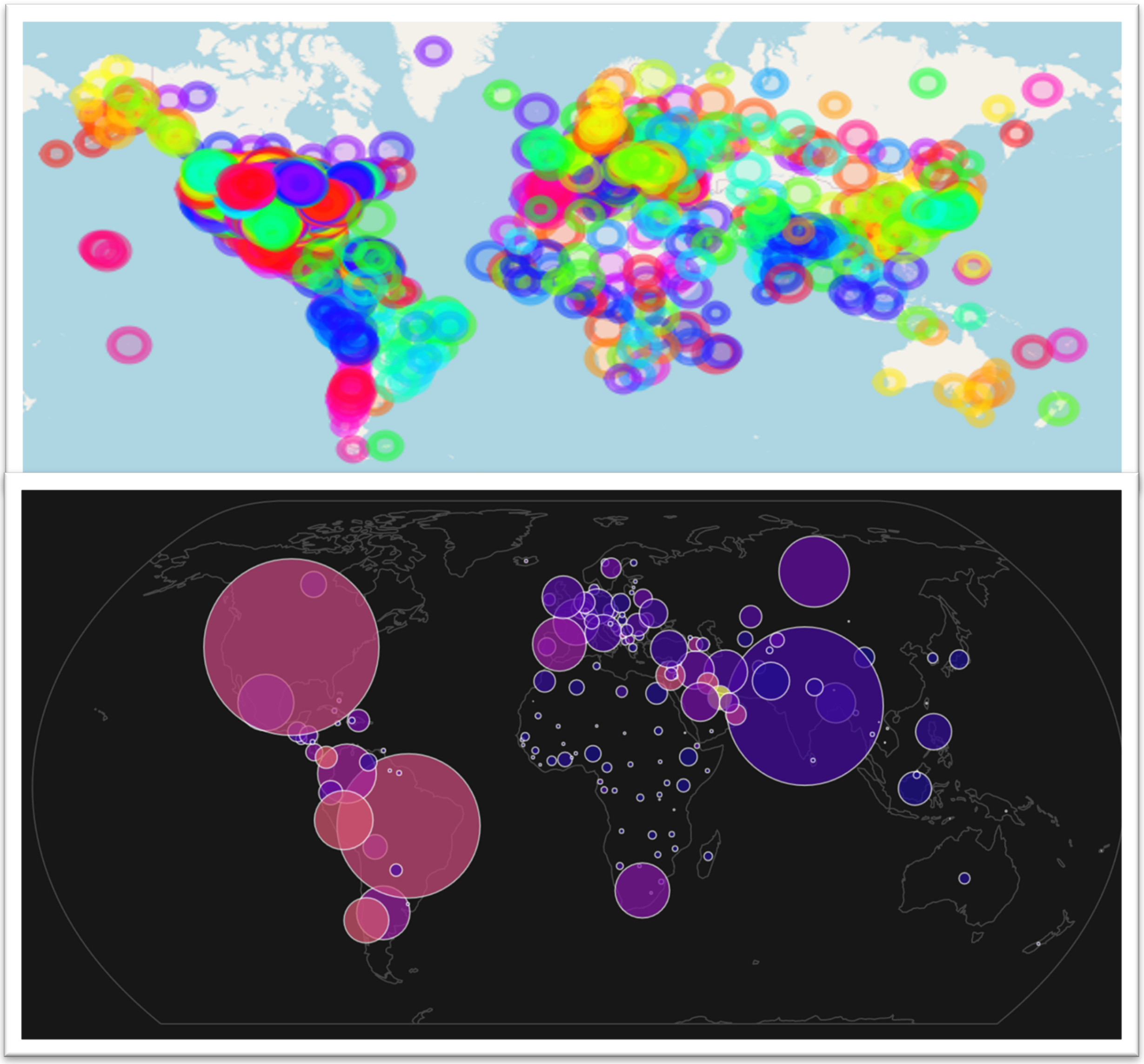
Table 2 – Summary Table of “*processed_location_Sep20th2020.csv*”

“*processed_location_Sep20th2020.csv*” (n = 3,954)						
Attributes	**Data Type**	**Number of Missing Values**	**Number of Unique Values**	**Summary Statistics (For Numerical Data Only)**		
				Min	**Median**	**Max**
Province_State	“char”	168	563			
Country_Region	“char”	0	188			
Last_Update	“char”	0	3			
Lat	“numeric”	80	3,874	-52.37	37.94	71.71
Long_	“numeric”	80	3,864	-174.16	-86.88	178.06
Confirmed	“int”	0	2,116	0	498.5	1,167,496
Deaths	“int”	0	583	0	9	37,076
Recovered	“int”	0	590	0	0	2,577,446
Active	“numeric”	2	1,939	-2,577,446	421	337,913
Combined_Key	“char”	0	3,954			
Incidence_Rate	“numeric”	80	3,863	0.0	1,204.4	14,871.2

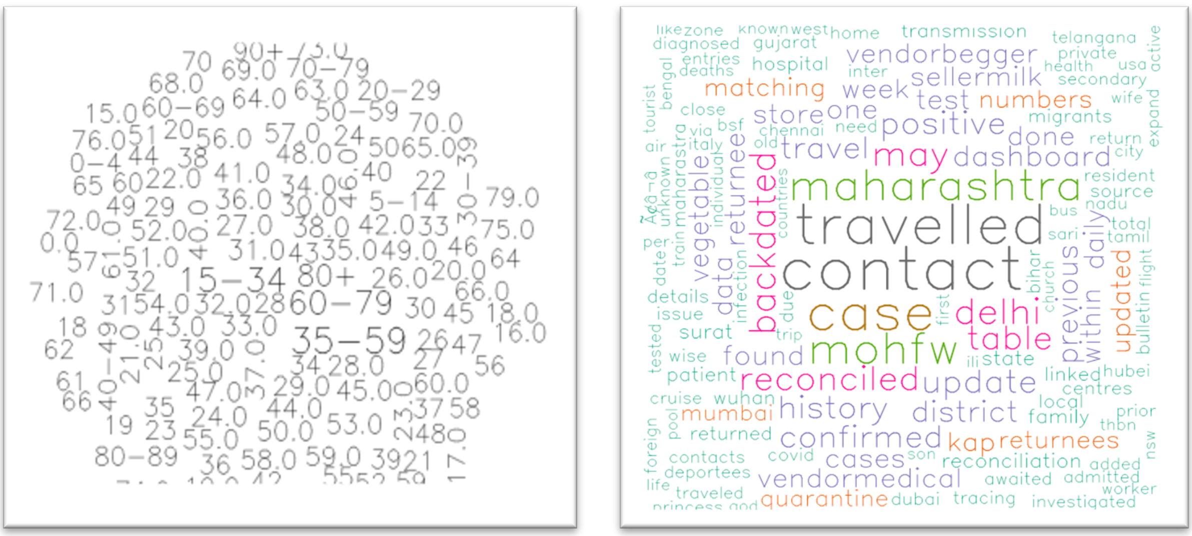
Graph 3 – Pie Charts on the *Sex* and *Outcome* Attributes from “case.csv”



Graph 4 –Number of Covid-19 Cases from “location.csv” – based on latitude & longitude (R and Python)



Graph 5 – Word Cloud Keywords of ‘*Age’* and ‘*Additional_information’* from “cases.csv”



Dataset Preparation

The whole data pre-processing includes several stages: data cleaning, missing values imputation, outlier detection, transformation and dataset joining. In the end, we would have one merged dataset that is cleaned to an extent and ready to be trained and tested in the classification models.

i. Data Cleaning

In this stage, we held a reserved strategy and did not remove too much data, because many issues in our data can be fixed later. For some attributes with different formats, we transformed all values inside to a standard format. Finally, we did a robust check summary on each attribute to make sure they make sense. Some specific details are as follows:

- We removed some obviously useless attributes such as "*Last_Update*".
- We removed some data entries that do not make too much sense. For example, a few rows in "location.csv" have negative number of active cases in those locations.
- We also removed some data entries with important attributes missing, e.g., if both "*latitude*" and "*longitude*" in "cases.csv" (case) are unavailable.
- We transformed strings in "age" attribute to a standard format. For example, we split the range information to "*age_min*" and "*age_max*".

ii. Missing Values Imputation

According to the statistics in table 1 and 2, there is a lot of information that is missing. We used different strategies to impute the missing entries:

- For geographical data "*Lat*" and "*Long_*" in "location.csv", we filled the missing values with the mean geographical coordinates based on their province and country information.
- For the missing values in "*Case.Fatality_Ratio*", we filled them by computing reasonable values using the statistics from "*confirmed*" and "*death*" columns.
- If "*age*" or "*sex*" are missing, we encoded them to be "-1" or "NaN" for further use.

iii. Outlier Detection

In most cases, outlier detection methods work on numerical data. As we have cleaned the abnormal values for categorical variables in previous steps, here we focus on using Z-score and IQR methods to detect potential outliers for numerical attributes. Because most attribute distributions do not look normal, Z-score may not be really helpful even if following the Central Limit Theorem (>30 samples). Overall, we primarily use **IQR** to detect outliers.

Attributes	**IQR**		**Number of Outliers**		**Action & Reason**
	LB	UB	Z-score (0.95)	IQR	
Latitude	-54.80	65.01	2	1	**None.** They are just geographical coordinates and make sense.
Longitude	-125.50	145.94	0	142	
Lat	20.09	55.47	80	415	
Long_	-124.85	-49.83	80	530	
Confirmed	0.0	5096.0	82	639	**None.** They are what happened in reality. Large numbers come from big countries.
Deaths	0.0	118	78	609	
Recovered	0.0	0.0	48	614	
Active	-225	3457	2	543	**Remove** 6 negative numbers which make no sense.
Incidence_Rate	0.00	4315.46	80	115	**None**. Locations with high incidence rate are small communities with high infection rates.
Case.Fatality_Ratio	0.00	6.56	48	246	**Remove** obvious outliers (e.g. >100% fatality rate)

iv. Transformation

Most classification models do not allow strings as input. Hence, we should use one-hot encoding to create binary indicators for categorical variables. In this case, attributes *"source", "province", "country", "additional_information" and "sex"* are transformed.

Also, we used the "group by" function to aggregate the information for cases in US from the county level, used in the location dataset, to the state level, used in the cases dataset. Different attributes have a different aggregating strategy such as mean or sum.

v. Dataset Joining

Finally, we used the "merge" function on attributes "Province" and "Country" to join two datasets "location.csv" and "cases.csv", which has fewer unique values and makes it better than using "latitude" and "longitude" because potentially we lose fewer data.

Classification Models

As a group of three, we tested several classification models and evaluated their performances. Finally, we selected five of them from Python and R libraries. For an initial evaluation, we mostly used the default hyperparameters as the baseline models. More tuning steps are in later sections.

We chose KNN because KNN is a simple and straightforward classification tool to implement. It can also give us an asymptotical baseline error rate. We also chose Decision Tree as the base model. Random Forest will be its bagging version and XGBoost and LightGBM become its boosting ones.

Models	**Environment**	**Package**	**Hyperparameters**
Decision Trees	Python	$sklearn.tree.DecisionTreeClassifier$	All default
Random Forests	Python	$sklearn.ensemble.RandomForestClassifier$	All default
XGBoost	Python	$xgboost.XGBClassifier$	All default
LightGBM	Python	$lightgbm.LGBMClassifier$	All default
KNN	R	$library(FNN)$	K = 10

Initial Evaluation and Overfitting

We implemented several evaluation metrics: accuracy, precision, recall, F-score (micro, macro or weighted) and Kappa score. We also used the confusion matrix to intuitively see how well separated the four classes are, in particular, which class is easier or harder to separate.

Recall, F-scores and the Kappa score are the ones we focus on. Accuracy will be a great metric if predicted classes are uniformly distributed, whereas it is not the case in our dataset. We use F-score because it is a comprehensive measure based on precision and recall. Meanwhile, Kappa coefficient is good to measure inter-rater reliability for class labels.

In general, all models have good accuracies on both the training and test data. The F-scores show very similar patterns. The confusion matrix does show some difference, which suggests that different models may struggle in different labels to classify.

By tuning a few hyperparameters, models do not seem to overfit too much because there is no significant gap between training and testing accuracies. However, the performance and overfitting issues of our models really depend on how we split the data.

Model	Initial Evaluation Metrics (on Training and Test Sets)						
	Accuracy	F-Measure*				Kappa Score	Confusion Matrix
		'd'	'h'	'n'	'r'		
Decision Trees	0.89	0.31	0.88	0.99	0.71	0.82	Mainly struggle in classifying 'd'
	0.88	0.20	0.87	0.99	0.70	0.81	
Random Forests	0.89	0.28	0.88	0.99	0.72	0.83	Mainly struggle in classifying 'd'
	0.88	0.21	0.87	0.99	0.70	0.81	
XGBoost	0.87	0.22	0.87	0.99	0.65	0.80	Mainly struggle in classifying 'd' and 'r'
	0.87	0.18	0.87	0.99	0.64	0.79	
Light GBM	0.86	0.19	0.86	0.99	0.60	0.78	Mainly struggle in classifying 'd' and 'r'
	0.86	0.18	0.86	0.99	0.60	0.78	
KNN	0.87	0.19	0.86	0.99	0.68	0.80	A bit struggle in classifying 'h' and 'r'
	0.87	0.20	0.86	0.99	0.68	0.79	

Hyperparameter Tuning and Results

We split our dataset with train to test ratio 75:25. For KNN, R library "FNN" has a default knn. cv() function which can do leave-one-out (n-fold) CV and automatically compute CV misclassification rates on the training set. We fit a sequence of different k values to tune the KNN. For DecisionTree, RandomForest and LightGBM, we leverage scikit-learn to train and tune models. Considering efficiency, we excluded XGBoost, although we used it as one of our baselines.

We choose the "GridSearchCV" function to perform CV. Specifying a grid of parameter values (presented in the result table below), it will exhaustively search over them to find the best estimator on the desired metric. We also set a number of 5 folds for the trade-off between the runtime and the model performance. For each model, we select 3 most essential hyper-parameters to form different parameter grids. The whole process is replicated for multiple times.

The table above presents part of hyperparameter-tuning experiments. Pease refer to the result file attached to see all experiments we performed. <u>All metrics posted are evaluated on the training set</u>.

Partial Results of Decision Tree					
Hyperparameters			Accuracy	Overall Recall	Recall on 'deceased'
criterion	max_features	min_samples_leaf			
gini	0.8	1	0.88026	0.66572	0.12010
gini	0.8	2	0.87883	0.66242	0.11253
gini	0.6	2	0.87871	0.66259	0.11462

gini	0.6	3	0.87804	0.66027	0.10757
gini	0.4	1	0.88040	0.66576	0.11984
entropy	0.8	1	0.88032	0.66579	0.12037
entropy	0.8	2	0.87897	0.66215	0.11123
entropy	0.6	2	0.87893	0.66287	0.11410
entropy	0.6	3	0.87796	0.66026	0.10679
entropy	0.4	1	0.88029	0.66539	0.11880
Partial Results of Random Forest					
Hyperparameters			**Accuracy**	**Overall Recall**	**Recall on ‘deceased’**
criterion	**n_estimators**	**max_features**			
gini	50	0.8	0.88175	0.66600	0.11514
gini	50	0.8	0.88060	0.66585	0.11514
gini	100	0.6	0.88140	0.66582	0.11358
gini	100	0.6	0.88125	0.66590	0.11488
gini	200	0.4	0.88130	0.66613	0.11514
entropy	50	0.8	0.88133	0.66624	0.11593
entropy	50	0.8	0.88144	0.66619	0.11567
entropy	100	0.6	0.88140	0.66634	0.11567
entropy	100	0.6	0.88130	0.66613	0.11593
entropy	200	0.4	0.88135	0.66594	0.11567
Partial Results of LightGBM					
Hyperparameters			**Accuracy**	**Overall Recall**	**Recall on ‘deceased’**
n_estima tors	**boosting_type**	**num_leaves**			
50	gbdt	21	0.68799	0.69478	0.69504
50	gbdt	31	0.70061	0.70399	0.69713
50	dart	41	0.73741	0.71744	0.66449
50	goss	21	0.72798	0.71707	0.67572
100	gbdt	21	0.66753	0.68807	0.72141
100	gbdt	31	0.68744	0.69876	0.71097
100	dart	41	0.70545	0.70943	0.69922
100	goss	21	0.68871	0.70198	0.72298
200	gbdt	21	0.69472	0.69929	0.69452
200	gbdt	31	0.70619	0.70618	0.69347
200	dart	41	0.73560	0.71850	0.66345
200	goss	21	0.72681	0.70502	0.64447

Best Models and Conclusion

Performance Evaluation of Each Tuned Best Model on the Test Dataset				
Models	**Best Tuning Parameters**	**Accuracy**	**Overall**	**Recall on**

			Recall	deceased
KNN	*k* = 15	0.87421	0.65036	0.09179
Decision Tree	*criterion*=gini, *max_features*=0.6, *min_samples_leaves*=1	0.88299	0.67270	0.13469
Random Forest	*criterion*=entropy, *max_features*=0.6, *n_estimators*=50	0.88378	0.67264	0.12920
LightGBM	*boosting_type*=dart, *n_estimators*=200, *num_leaves*=21	0.69226	0.70856	0.74079

The table above shows the best result for all models on the test set after hyperparameter tuning. It points out that the LightGBM model is much better than other models on the recall score for 'deceased'. It mainly results from having balanced weights for different outcome labels while other models are unweighted. One thing to note is that the high recall score for 'deceased' (0.74079) is at the cost of the corresponding low precision score (0.03731).

The Decision Tree model is the second-best model that achieves 0.13469 on the recall score for 'deceased'. As for runtime efficiency, LightBGM and Decision Tree are way faster than Random Forest to fit. In conclusion, our final LightGBM model (*boosting_type*=dart, *n_estimators*=200, *num_leaves*=21) is the best estimator that achieves highest recall score 0.74079 for 'deceased'.

Lessons learnt and Future Work

It is kind of challenging to do classification if the distribution of the predicted classes is skewed, which in our cases, only 1.1% "deceased" data. To further improve this project, we could collect more data through the open source. We also learnt some experience from previous milestones. For example, dropping too much data with missing values could potentially influence the model's performance a lot. These are the most crucial points that are worth noticing.

Contributions

Although our group had a diverse background, in general, we contributed evenly to this project.

- **Andy Wang**: built the KNN model in R, did EDA and drafted every project report
- **Qirui Wu**: built the Boosting machines in Python and took main responsibility in data cleaning
- **Liyang Zhou**: built the Decision Tree and Random Forest in Python and finalized reports