**COVID-19 dataset analysis**

**Problem Statement:**

The goal of the project is to evaluate different machine learning models from sklearn to predict the patient status (e.g. deceased, hospitalized , nonhospitalized, adn recovered) by using information (e.g. age, sex, city, etc) provided in the COVID dataset. Then apply cross validation to tune for the best hyperparameter for each model and elect the most accurate model.

**Dataset description and EDA：**

Dataset description:

* There are two datasets named ‘**Case dataset**’ and ‘**Location dataset**’. The total length of ‘case dataset’ is *557364*, ‘location dataset is *3954*.
* The **missing value** in case dataset are {'age': 296874, 'sex': 293734, 'province': 6568, 'country': 24, 'latitude': 2, 'longitude': 2,'date\_confirmation': 462, 'additional\_information': 522969, 'source': 209191}
* The **missing value** in location dataset are {'Province\_State': 168,'latitude': 80, 'longitude': 80, 'Active': 2, 'Incidence\_Rate': 80, 'Case-Fatality\_Ratio': 48}

Observations:

The most positive case is in India with the age around 30-50 (Figure 1. & Figure 2.) The outcome of ‘deceased’ is the lowest in the outcome distribution map (Figure 3.). Figure 4. and 5. shows the four provinces contribute most death cases and the three provinces have most recovered cases.

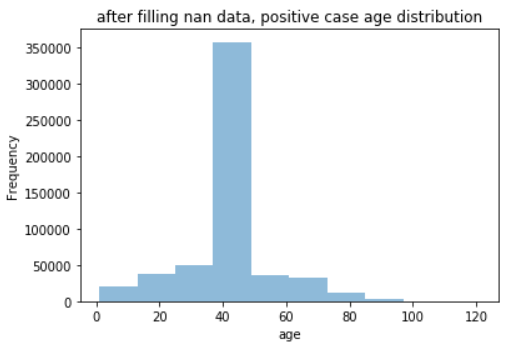


Figure 1. Age distribution after filling NaN

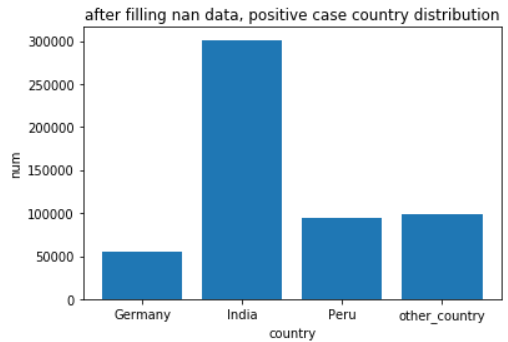


Figure 2. Positive case distribution in countries

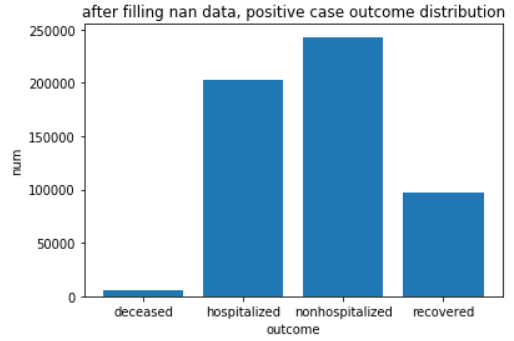


Figure 3. overall positive case distribution

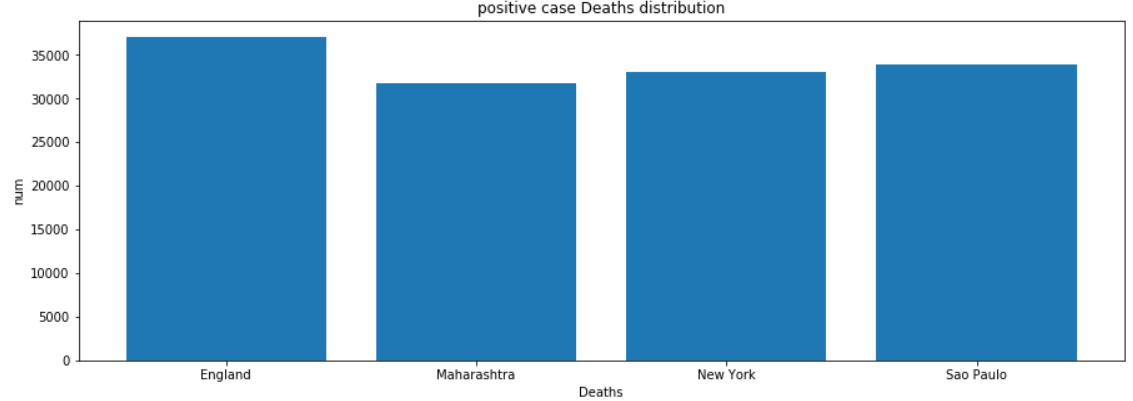


Figure 4. Countries with most positive cases

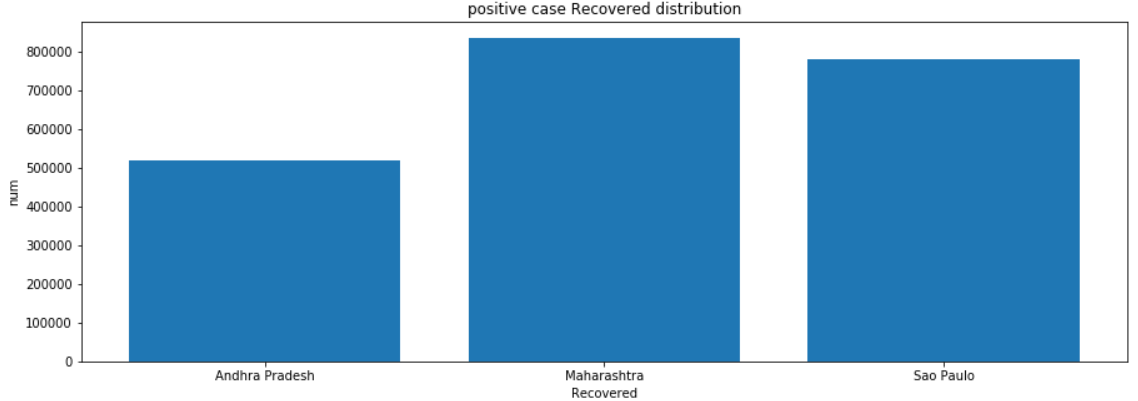


Figure 5. Countries with most recovered cases

**Data preparation:**

1. Data Cleaning:

*Case Dataset:*

* Age:

1. For ages with dash sign ‘-’, drop the sign and use the mean value of the two ages
2. For ages with plus or minus sign, remove the plus or minus sign
3. Convert all forms of string numbers to integers
4. For ages with month value, we add the integer result of the month divided by 12 to the age value
5. Drop the age data that have unknown characters in them

* Sex: replacing missing value based on ratio of male and female.

*Location Dataset:*

* Active: using the formula ‘ Active cases = total cases - total recovered - total deaths’ to evaluate the missing Active value. If the value is negative or non-feasible then add 0 to the missing index
* Incidence\_Rate: using the formula ‘Incidence Rate = cases per 100,000 persons’ to evaluate the missing Incidence\_Rate. Since there is no missing value for confirmed cases. Non-feasible values will be eliminated in the outlier section.
* Case-Fatality\_Ratio: follow the formula ‘Case-Fatality Ratio (%) = Number recorded deaths / Number cases’ to fill in the missing value. Check if the Number cases is greater than zero, otherwise fill 0 in the missing index- Latitude and Longitude: Since the missing value of latitude and longitude are 81, which are small enough to neglect, dropna() was applied to remove the missing value
* Province\_State: Since there are no extra evidences that support for this value, the optimal way would be removing the value
* Last \_Updated: After the group’s discussion, we all think that the last updated value would not influence the outcome. Hence the column of last updated value is dropped

2.Outlier Detection:

*Case Dataset:*

The age value is between [1,121]. As most people live 100 years, we think the age above 100 is outlier.

*Location Dataset:*

Negative active cases entry will be removed during this process. If we demonstrate the data in different attributes of the location dataset, many consecutive data points would be marked as outliers, and this may be caused by some locations having extremely high active cases. Therefore, outlier detection using IQR scores will lead to massive data loss. To remove data entry errors while keeping correct values as much as possible, the top and the bottom 10 percent of the data will be removed. By having this method, skewness will be fixed back to the normal range of 1 to -1.

3.Merging Datasets:

For the joining process, we chose to use the ***province*** and ***country*** as the join attributes. One

reason is that using province and country makes more sense for data analyzing as the data

will be most likely to be analyzed province by province or country by country. Furthermore,

from the accuracy perspective, two locations with close latitude and longitude may have very

different death rates or active cases. The reason for that may be because they belong to

different countries or provinces which are having different pandemic policies. Therefore,

using country and provinces is best suitable for joining in this case.

**Classification models:**

**AdaBoost (**Lizhou Ding):

I implemented the ***AdaBoost*** model. ***Adaboost*** is easy to implement. It is fast. It is possible to combine with any machine learning algorithm and there are not many parameters needed to tune. Also, it can be used with text and numerical data. Adaboost combines several weak learners to create a strong learner. It can correct previous errors or observations that are wrong classificated.

**KNN** (ZiZe Zhang):

KNN is the best when it comes to classifying datasets that are ***non-parametric*** which means that we have little to no knowledge about the distribution data. Since we do not have much knowledge about the situation of COVID pandemic, we could observe the pattern of distribution by applying KNN. As one of the most classific machine learning models, KNN is simple to use and it provides high accuracy. The logic behind KNN is to identify the ***similarity*** between objects. Since we are predicting the outcome of peoples in different countries and areas with limited personal information, we can calculate the similarity between peoples in different areas and observe the result to classify. However, in terms of training, KNN model is slow since it takes the speed of Big N to complete the training.

**XGBoost**(Siyuan Wu): XGBoost is one of the most widely used classification methods in recent data science. Compared to usual learning models such as decision trees, boosted trees take an iterative approach. With each iteration, a new model will be implemented to correct the errors made by the previous ones. Therefore, I would expect a lower rate of error by using this model. Besides, XGBoost’s interchangeable loops used for building base learners enable it to run in parallelization, and it also has a very strong hardware optimization support. Thus, predictably, XGBoost would demand significantly less training time.

**Initial evaluation and overfitting:**

**AdaBoost** (Lizhou Ding):

Hyperparameter: n\_estimator=50, learning rate=1

Result: {test accuracy 0.7677, train accuracy 0.7697, recall 0.4838, precision 0.4633}

Overfit: A threshold of 5% difference between train and test set accuracy is set. As we can see, there is overfitting if I increase the n\_estimator from 100 to 1000.The accuracy is pretty good. However, for multiclass, imbalanced datasets, the recall value is more important. We want to detect as much target outcome as possible like deceased or recovered.

**KNN**(ZiZe Zhang):

Hyperparameter: n\_neighbor = 5, leaf\_size = 30, p =2

Result: {test accuracy 0.8545, train accuracy 0.8538, deceased recall 0.06, deceased precision 0.58} Since we are using metrics of recall to evaluate the model, KNN returns the best result in terms of recall on deceased class. It returns 78 true positive cases according to the confusion matrix.

Overfitting Evaluation:As the threshold of 5% is set to observe the tendency of overfitting, the KNN model shows no sign of overfitting.

**XGBoost**(Siyuan Wu):

XGBoost default setting: n\_estimator = 150, learning rate = 0.1, max\_depth = 3

result: {test accuracy 0.8557, train accuracy 0.8547, deceased recall 0.04, deceased precision 0.68}

Overfitting Evaluation: The accuracy difference between test and train is less than 5%, so there is no sign of overfit.

**Hyperparameter tuning:**

**AdaBoost** (Lizhou Ding):

As the learning rate and n\_estimator are the most important hyperparameters of adaBoost model, I tune these two hyperparameters. I use the GridSearchCV method to find the best hyperparameters’ value. The method uses cross-validated grid-search to find the best accuracy parameters’ value. I set the range of two hyperparameters, and let the method find the best value. The goal is to find the highest recall value. The advantage of it is it tries all combinations of parameters. Although it will cost more than RandomSearchCV, I think the time is reasonable based on our dataset size.

**XGBoost** (Siyuan Wu):

The hyperparameters being tuned are:

* n\_estimators: The number of boosting stages
* Max\_depth: Maximum depth of the individual regression estimators.

As stated on the Sklearn document, max depth limits the number of nodes in the tree and this parameter needs to be tuned for best performance. For this assignment, GridSearchCV is utilized to evaluate the effect of tuning the parameter, and with Max\_depth smaller than 12, the running time is long, yet still reasonable. However, as Max\_depth continues increasing, the running time also grows significantly, therefore, randomizedSearchCV is suggested for future tuning.

**KNN** (ZiZe Zhang):

The hyperparameters being tuned are:

* n-neighbor: number of neighbor
* leaf\_size: leaf size passed to BallTree or KDTree in the model
* p: distance calculation function

Since the dataset contains massive information, using randomizedSearchCVwill greatly reduce the runtime additional to the average longer runtime for KNN models. In addition, the range of hyperparameters is large to tune, hence it is more efficient to tune with RandomizedSearchCV. The downside of using randomizedsearchCV is that it may miss some of the important combinations of hyperparameters.

**Results:**

Lizhou Ding:

Hyperparameters : Learning\_rate: LR, n\_estimator: NE

Tuning method: GridSearchCV

|  |  |  |  |
| --- | --- | --- | --- |
| Hyperparameters | Accuracy | Overall Recall | Recall on ‘deceased’ |
| LR=1,NE=1000 | 0.4 | 0.4646 | 0.93 |
| LR=0.5,NE=100 | 0.79 | 0.5275 | 0.15 |
| LR=0.01,NE=500 | 0.8 | 0.4975 | 0 |
| LR=0.01,NE=1000 | 0.8 | 0.5 | 0 |

XGBoost(by Siyuan Wu):

Hyperparameters :n\_estimator: NE, max\_depth: MD

Tuning method: GridSearchCV

|  |  |  |  |
| --- | --- | --- | --- |
| Hyperparameters | Accuracy | Overall Recall | Recall on ‘deceased’ |
| NE=100, MD = 3 | 0.85 | 0.57 | 0.03 |
| NE=100, MD = 6 | 0.86 | 0.60 | 0.07 |
| NE=150, MD = 3 | 0.85 | 0.58 | 0.04 |
| NE=150, MD = 6 | 0.86 | 0.61 | 0.08 |
| NE=150, MD = 9 | 0.86 | 0.63 | 0.14 |

KNN (ZiZe Zhang):

Hyperparameters: n\_neighbor, leaf\_size, p (distance calculation)

Tuning method: RandomizedSearchCV (max\_iter:10, random\_State:0.5)

|  |  |  |  |
| --- | --- | --- | --- |
| Hyperparameters | Accuracy | Overall Recall | Recall on ‘deceased’ |
| n\_neighbor=5，leaf\_size=30, p=2 | 0.83 | 0.6 | 0.06 |
| n\_neighbor=22，leaf\_size=31,p=4 | 0.85 | 0.6 | 0.04 |
| n\_neighbor=50，leaf\_size=50,p=1 | 0.85 | 0.6 | 0.04 |
| n\_neighbor=10，leaf\_size=90,p=30 | 0.85 | 0.59 | 0.04 |

**Conclusion:**

|  |  |  |  |
| --- | --- | --- | --- |
| Models | Accuracy | Overall Recall | Recall on ‘deceased’ |
| Adaboost | 0.79 | 0.5275 | 0.15 |
| XGboost | 0.79 | 0.4875 | 0 |
| KNN | 0.85 | 0.6 | 0.06 |

**Adaboost**:

* advantage:
  + compatible with text and numeric data
  + efficient and easy to use
* disadvantage:
  + sensitive to noise and outlier
  + imbalanced data will decrease accuracy

**XGboost**:

* advantage:
  + Allows parallel processing, which speeds up computing time.
  + Having regularization feature that prevents overfitting
* disadvantage:
  + Computing time grows significantly with large datasets and Max\_depth

**KNN**:

* advantage:
  + high accuracy
  + do not require prior knowledge
* disadvantage:
  + slow runtime

**Conclusion:**

Overall, Adaboost returns the most optimal results among the three models. Although KNN has the best accuracy, the goal here is to increase the recall on the ‘deceased’ class to prevent false negatives. Hence adaboost is selected since it returns the best result of “recall on ‘deceased’”.

**Lessons learnt and future work:**

Throughout the entire project, we have experienced the entire process of data mining from imputing data to selecting models and to tuning hyperparameters. This project conveys a general idea of how to perform data mining in a real-life setting and provides an opportunity for us to apply the knowledge we learn in class to a practical situation. In the project, we start from preparing a dataset to showing classifying results. We have learnt that a good preparation is crucial for classification. A raw dataset is hard to classify as well as visualizing any reasonable result. It is important to choose a reasonable method to clean data and impute missing data. A different method will lead to a different result. However choosing a suitable classifier model is also significant. In the project, we have a big size dataset. The dataset is imbalanced and contains multiclass. We carefully choose models and evaluate them to find the model has less runtime cost and achieves high target accuracy. Throughout the project, we also learn how to combine real-life situations to model selection. By trying different metrics, we have decided to select the model by evaluating the recall on the deceased class since in reality it is more important to determine if a person is deceased or not . In the future, we could utilize more information to improve our result. For example, we can utilize the latitude and longitude. Based on them, we can limit regions, like high active case regions may result in high deceased cases. Also, the aspects that we need to improve are data imputation and model evaluation. We can improve the model accuracy by generating a cleaner dataset using function in sklearn instead of doing manual calculation. In terms of model evaluation, we should consider the real-life situation and select the most appropriate metric to evaluate the model.

**Contributions:**

Lizhou Ding:

I do the preprocessing for the Case dataset including data cleaning and imputing missing values and dealing with outliers. I plotted all graphs for the milestone1. I implemented adaBoost model for milestone2 including splitting dataset, evaluation, and analyzing overfitting. I tuned hyperparameters for adaBoost in milestone3. I start the first draft of the report in milestone3.

ZiZe Zhang:

In milestone 1, I was in charge of exploring and analysing data , cleaning the ‘age’ column, joining the cases and location dataset, and finalizing the report. In milestone2, I implemented KNN model with data splitting, prediction, and overfit prevention. For milestone3, I hypertuned the KNN model hyperparameter by implementing GridSearchCV method.

Siyuan Wu:

For milestone 1, I implemented functions to remove outliers and fix the skewness of the data. Moreover, I did the aggregation for the information and prepared the data for the later joining process. For the second milestone, I implemented the XGBoost model and modularized team member’s code. In milestone 3, I am in charge of hyper tuning the XGBoost model using GridSearchCV and finalizing the report.