REPORT
MILESTONE 2
CMPT 459 – DATA MINING

Predict health outcome groups for people worldwide using COVID-19 dataset

NAVJOT KAUR (301404765)
SAAYAN MAHESANIA (301386332)

PROBLEM STATEMENT

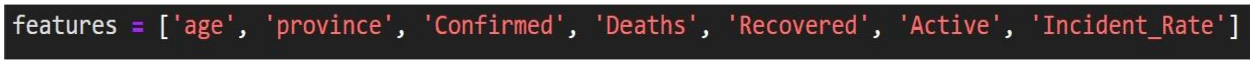
This project concerns with classifying and predicting the health outcomes of people who were diagnosed with COVID-19 disease in 2021. Two datasets are provided for which one is used for training and validation purposes, and the other is used for testing purposes on which the predictions are to be done. The predicted outcome is the health of the people i.e., whether they are ‘deceased’, ‘hospitalized’, or ‘nonhospitalized’. Two classification models are used for predicting the outcome_groups, which are RandomForestClassifier and KNeighborsClassifier. Out of these the machine learning model that provides better accuracy on validation data is considered, which comes out to be RandomForestClassifier and is used for testing purposes. The goal of this project is to build classification models and evaluate the performance so that a better accuracy and macro F1-score for the dataset is obtained.

The given data was messy at first and the data preprocessing, data cleaning, and dealing with outliers are done in the first milestone. For milestone 2, a new refined dataset is provided which is clean and ready to use for the tasks such as feature selection, balancing, building models, and hyper-parameter tuning which will be discussed in this report.

1.1 Feature Selection

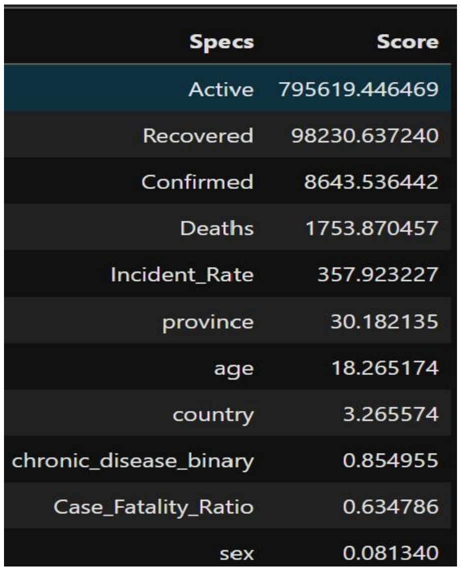
Feature selection is done so that the irrelevant features that do not influence the output are removed first. Latitude and Longitude are removed since country and province are already present in the dataset and the latter are much better estimators than the former. Moreover, the date_confirmation is removed because the data is given for one year and it does not have much effect on predicting the outcome. The missing province is given “no prov” value before mapping. From the rest of the features namely, ‘age’, ‘sex’, ‘province’, ‘country’, ‘chronic_diesase_binary’, ‘Confirmed’, ‘Death’, ‘Recovered’, ‘Active’, ‘Incident_Rate’, ‘Case_Fatality_Ratio’, the feature selection is done using chi2 feature selection method from where the top 7 features with highest scores were selected. It can be noted that the Mapping is done before feature selection to create all numerical attributes, and it would be easier to select them.

Final features selected:

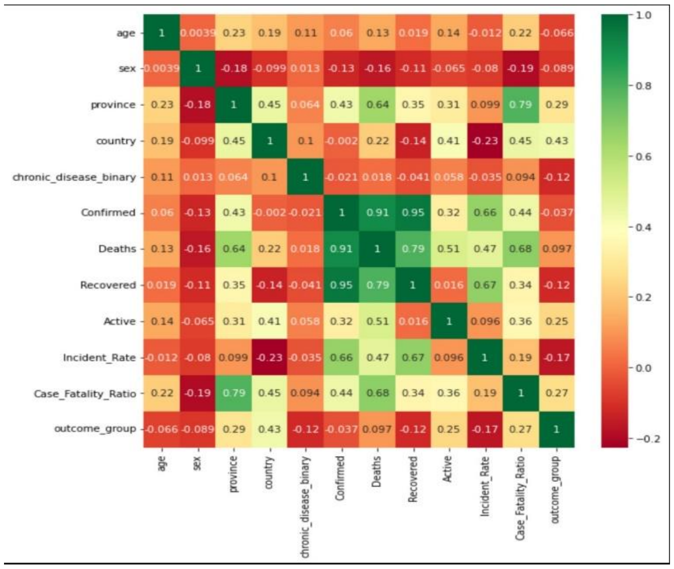


```
features = ['age', 'province', 'Confirmed', 'Deaths', 'Recovered', 'Active', 'Incident_Rate']
```

These features are selected because of better scores.



Specs	Score
Active	795619.446469
Recovered	98230.637240
Confirmed	8643.536442
Deaths	1753.870457
Incident_Rate	357.923227
province	30.182135
age	18.265174
country	3.265574
chronic_disease_binary	0.854955
Case_Fatality_Ratio	0.634786
sex	0.081340



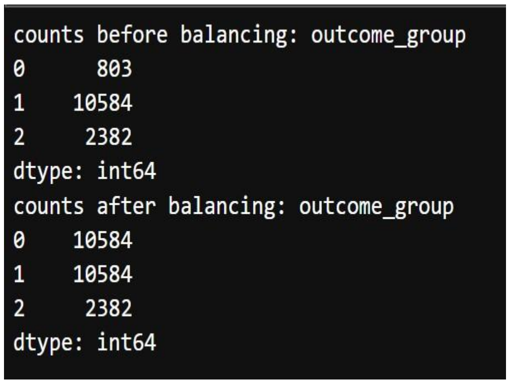
	age	sex	province	country	chronic_disease_binary	Confirmed	Deaths	Recovered	Active	Incident_Rate	Case_Fatality_Ratio	outcome_group
age	1	0.0039	0.23	0.19	0.11	0.06	0.13	0.019	0.14	-0.012	0.22	-0.066
sex	0.0039	1	-0.18	-0.099	0.013	-0.13	-0.16	-0.11	-0.065	-0.08	-0.19	-0.089
province	0.23	-0.18	1	0.45	0.064	0.43	0.64	0.35	0.31	0.099	0.79	0.29
country	0.19	-0.099	0.45	1	0.1	-0.002	0.22	-0.14	0.41	-0.23	0.45	0.43
chronic_disease_binary	0.11	0.013	0.064	0.1	1	-0.021	0.018	-0.041	0.058	-0.035	0.094	-0.12
Confirmed	0.06	-0.13	0.43	-0.002	-0.021	1	0.91	0.95	0.32	0.66	0.44	-0.037
Deaths	0.13	-0.16	0.64	0.22	0.018	0.91	1	0.79	0.51	0.47	0.68	0.097
Recovered	0.019	-0.11	0.35	-0.14	-0.041	0.95	0.79	1	0.016	0.67	0.34	-0.12
Active	0.14	-0.065	0.31	0.41	0.058	0.32	0.51	0.016	1	0.096	0.36	0.25
Incident_Rate	-0.012	-0.08	0.099	-0.23	-0.035	0.66	0.47	0.67	0.096	1	0.19	-0.17
Case_Fatality_Ratio	0.22	-0.19	0.79	0.45	0.094	0.44	0.68	0.34	0.36	0.19	1	0.27
outcome_group	-0.066	-0.089	0.29	0.43	-0.12	-0.037	0.097	-0.12	0.25	-0.17	0.27	1

1.2 Mapping the features

Mapping of categorical features is done by using the codes method where each unique category of the column is given a number. The mapping is done for sex, chronic_disease_binary, country, province, and date_confirmation. For sex, 0 is given to females and 1 for males. For chronic_disease_binary, 0 is for false and 1 is for true. For country, province, and date_confirmation there are more than 2 unique categories, so they are mapped in the ascending order of alphabets or numbers.

The mapping for outcome_group is done such that {0: ‘deceased’, 1: ‘hospitalized’, 2: ‘nonhospitalized’}. Mapping is helpful since it converts whole data into numerical attributes and the predictions task is made easier.

1.3 Balancing the classes in the dataset



```
counts before balancing: outcome_group
0      803
1    10584
2     2382
dtype: int64
counts after balancing: outcome_group
0    10584
1    10584
2     2382
dtype: int64
```

After printing the training data outcomes, it was seen that the data is highly imbalanced where hospitalized data entries are 10584 and 2382 entries for the nonhospitalized group with only 803 entries for the deceased group. Before balancing, the training and test data is normalized and training data is split into the training and validation data using test_train_split since we don't want data to be redundant in our validation set. Balancing of data is required because our group of interest is deceased which means the F1-score needs to be calculated so using the oversampling method by RandomOverSampler using minority class (deceased) only, the training data was balanced (leaving validation data out). Counts are shown in the picture.

1.4 Building models and hyperparameter tuning

After the training data is split into train and validation sets and it is balanced on minority, two classification models are chosen: Random Forest Classifier and KNeighborsClassifier. Many models including SVM, XBoost, KNN, DecisionTree, NN, and RandomForest were fitted to the dataset from which RandomForest and KNN outperformed the other classification models. So, we decided to tune their hyperparameters. Initially, the plan was to select one of the RandomForest/KNN and the other model to be NN, however, because of similar metric scores of Random forest and NN, we discarded NN because of its high computation power and time. These two models, RandomForest and KNN moved forward for hyperparameter tuning. Additionally, RandomForest is less sensitive to outliers compared to other models like decision trees. SVM and MLP could be chosen but they were comparatively slow and used a lot of computational power.

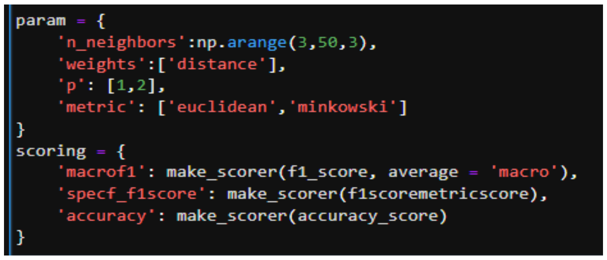
For hyperparameter tuning, GridSearchCV and RandomizedSearchCV were used for KNN and RandomForest respectively. The number of K selected for cross-validation is 5 because if the K is lower, the model's accuracy drops and if the K is higher the model has access to more data and the validation set has significantly less data which would result in lower accuracy since the data is being balanced by oversampling. So, we went with the basic choice which was cv=5.

For KNN, we found that on our system it is not computationally expensive since it is fast, so we decided on GridSearchCV as it trains the model on every possible hyperparameter combination. Although it was computationally expensive, still GridSearchCv was fast. For RandomForest, since it was computationally expensive, we chose RandomizedSearchCV because it takes the random combinations of hyperparameters and is not very expensive.

We first decided to randomly (manually) check some numbers for hyperparameters to test for scores at huge ranges so that we can further classify a range to input in the k-fold RandomizedSearchCV and GridSearchCV. The range of values that we found to produce a good accuracy score was chosen for hyper parameter tuning. Finally, the SearchCV resulted in tuning and choosing the best parameters for the model based on the best f1score for the deceased class as it was the class that had the least data, and which could be underrepresented by the model.

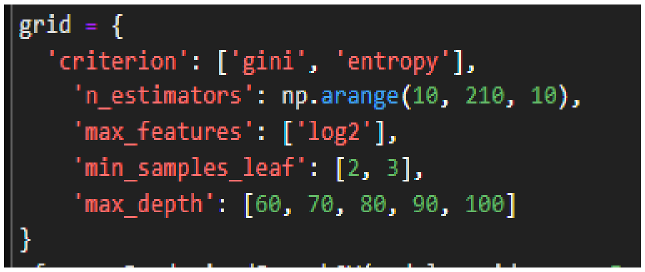
Hyperparameters used are:

KNN



```
param = {
    'n_neighbors':np.arange(3,50,3),
    'weights':['distance'],
    'p': [1,2],
    'metric': ['euclidean','minkowski']
}
scoring = {
    'macrof1': make_scorer(f1_score, average = 'macro'),
    'specf_f1score': make_scorer(f1scoremetricscore),
    'accuracy': make_scorer(accuracy_score)
}
```

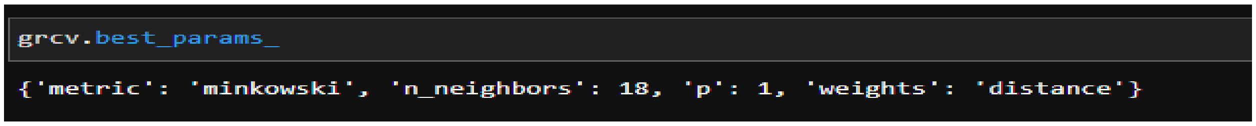
RandomForest



```
grid = {
  'criterion': ['gini', 'entropy'],
    'n_estimators': np.arange(10, 210, 10),
    'max_features': ['log2'],
    'min_samples_leaf': [2, 3],
    'max_depth': [60, 70, 80, 90, 100]
}
```

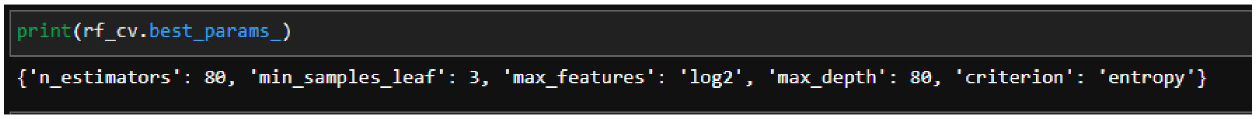
Model	Hyperparameters	Mean macro F1-score across the validation sets	Mean F1-score on 'deceased' across the validation sets	Mean overall accuracy across the validation sets
KNN	See pic i	0.850355	0.884156	0.892781
RandomForest	See pic ii	0.757269	0.459906	0.916044

i.



```
grcv.best_params_

{'metric': 'minkowski', 'n_neighbors': 18, 'p': 1, 'weights': 'distance'}
```

ii.



```
print(rf_cv.best_params_)
{'n_estimators': 80, 'min_samples_leaf': 3, 'max_features': 'log2', 'max_depth': 80, 'criterion': 'entropy'}
```

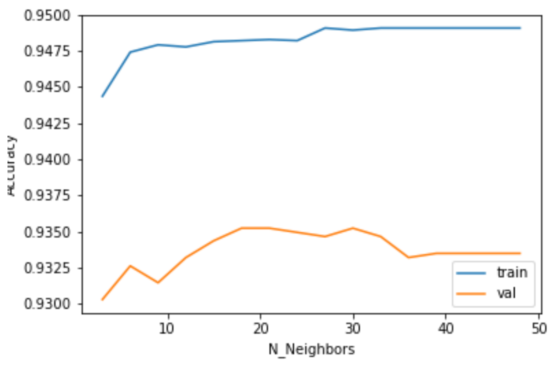
1.5 Overfitting

We measured the accuracy scores for the training and validation datasets on the two models by changing their main hyperparameter and plotted the plot by tuning n_neighbors for KNN and max_depth for RandomForest.

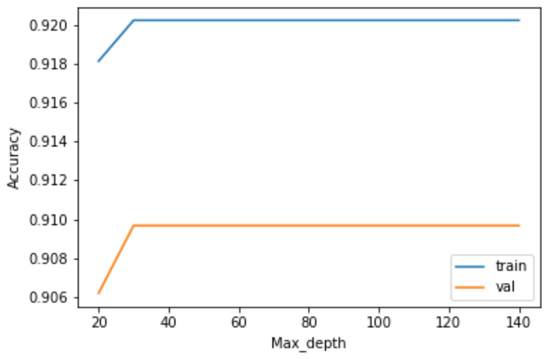
It is evident from the below-given images that the KNN is slightly overfitting because it started to fluctuate at n_neighbors = 18, however, the training accuracy stays increasing. The RandomForest model suggests that when increasing the depth there are no oscillations in the validation dataset hence suggesting that RandomForest did not overfit. The gap between training and validation for RandomForest remains constant and does not change. Moreover, RandomForest is less sensitive to noisy data, unlike KNN.

Plots:

KNN:



RandomForest:



1.6 Comparative study

RandomForest:

RandomForest was a good choice as they are much more robust against overfitting as they use multiple decision trees, also because of this, it is easy to hyper-tune them.

The main disadvantage for RandomForest compared to KNN is the amount of computation power and time it takes to tune and train the model. It takes a longer time to find an optimal parameter on a range of choices hence it was also a reason we had to switch to RandomizedSearchCV from GridSearchCV to reduce time, unlike KNN where we used GridSearchCV. In the end, we do get satisfactory results however, we still cannot be sure that if we did find the best optimal parameter as some could be missed because we tried to save time and computation power.

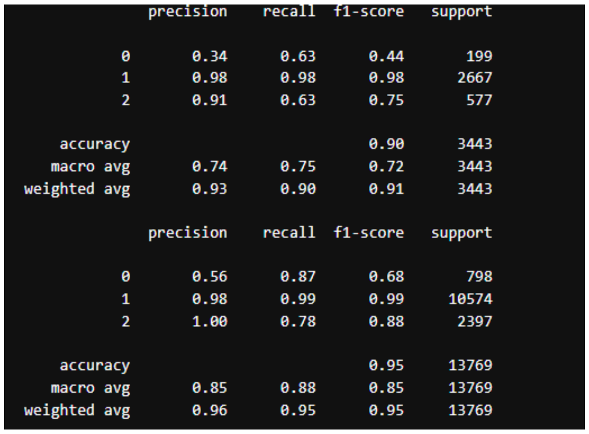
KNN:

The major advantage of the KNN Classifier is that it does not require any training time, specifically KNN is an Instance-based learning model, and it only learns from the training dataset at the time of prediction of validation and test datasets making it much faster than RandomForest and other models like SVM and Regression models. Another advantage of KNN that we found was that it was much easier to tune KNN as only one main hyperparameter, n_neighbors was needed to be tuned.
Disadvantages of KNN we found out that scaling dataset plays a major role in accuracy we could and need to have spent more time in scaling each feature properly so that KNN does not generate incorrect predictions, moreover KNN is also much sensitive to noisy data unlike RandomForests.

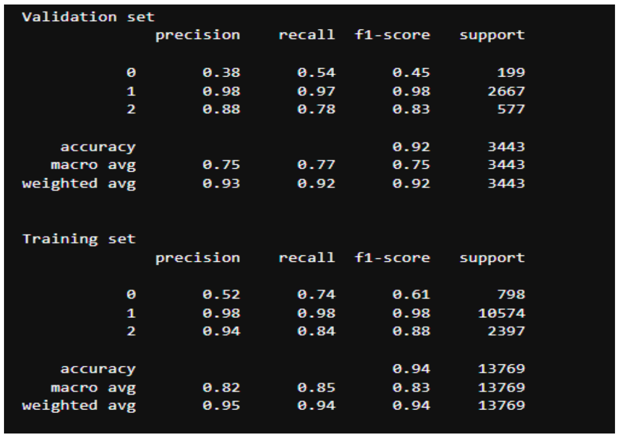
The F1 scores for the deceased in both models for the validation dataset seem to be lower than in the training dataset. This can be a result of the imbalanced data that we balanced by oversampling the training data. This can make the precision to be low in the training dataset for the deceased class as some of them could be misclassified. We can see that RandomForest performed better than the Knn model on the accuracy metric (from plots in section 1.7) Hence RandomForest is selected for predictions.

1.7 Prediction on the test set

Screenshot of Knn: (above val, below train)   RandomForest:



```
              precision    recall  f1-score   support

           0       0.34      0.63      0.44       199
           1       0.98      0.98      0.98      2667
           2       0.91      0.63      0.75       577

    accuracy                           0.90      3443
   macro avg       0.74      0.75      0.72      3443
weighted avg       0.93      0.90      0.91      3443

              precision    recall  f1-score   support

           0       0.56      0.87      0.68       798
           1       0.98      0.99      0.99     10574
           2       1.00      0.78      0.88      2397

    accuracy                           0.95     13769
   macro avg       0.85      0.88      0.85     13769
weighted avg       0.96      0.95      0.95     13769
```



```
Validation set
              precision    recall  f1-score   support

           0       0.38      0.54      0.45       199
           1       0.98      0.97      0.98      2667
           2       0.88      0.78      0.83       577

    accuracy                           0.92      3443
   macro avg       0.75      0.77      0.75      3443
weighted avg       0.93      0.92      0.92      3443


Training set
              precision    recall  f1-score   support

           0       0.52      0.74      0.61       798
           1       0.98      0.98      0.98     10574
           2       0.94      0.84      0.88      2397

    accuracy                           0.94     13769
   macro avg       0.82      0.85      0.83     13769
weighted avg       0.95      0.94      0.94     13769
```

The results for KNN and RandomForest suggest that the accuracy and f1-score of RandomForest on the validation set were better than KNN, the reason it was chosen for testing purposes. The f1-score is above 0.40 for the validation set for both models and the macro f1-score is above 0.70. The results obtained are after balancing as when an imbalanced dataset was used the f1-score dropped significantly.

Conclusion

This was a highly interesting project where every task from data preprocessing to data prediction is performed. Firstly, the messy data is cleaned and the relevant features are selected by mapping the categorical features to numerical attributes. After that, since the data was highly imbalanced, the balancing of data was done by oversampling the majority class. Then RandomForest and Knn models were built and tuned for the best hyperparameters which were fitted to the training set to get the prediction of the validation set. Looking at the accuracy measures it was concluded that the model did not overfit much because the accuracy scores of training and validation sets did not have a significant gap. They were almost similar for both models. The performance of both the models was compared and it was found that RandomForest's accuracy and f1-score were better than KNN's, so RandomForest was chosen to be the final model from where the CSV has been generated.

The great key finding while tuning was selecting good features, even if one feature was dropped or added, it would have a significant effect on the accuracy scores of the validations set.

Lessons learnt and future work

The most important lesson learnt is that for a machine learning model, to train and predict correct outcomes, the dataset should be balanced, and the feature selection should also be good since it contributes to the accuracy of the model. Moreover, a training model cannot be judged on just the accuracy score of the validation set. It is necessary to investigate other metrics like recall, precision, and f1-score to evaluate our model and hyper-tune/data-preprocess model and dataset respectively. Another lesson learned is that machine learning requires a lot of computational power and time, hence while training or hyper tuning a model, requiring a good computational resource such as good GPU cores is a necessity.

References

https://machinelearningmastery.com/bagging-and-random-forest-for-imbalanced-classification/

https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/

sklearn documentation

Contributions

Navjot: Mapping the features, Balancing the classes in the training dataset, building RandomForest and hyperparameter tuning on RandomForest, Comparative Study, Prediction on test using RandomForest

Saayan: Feature Selection, build KNN and hyperparameter tuning on KNN, Overfitting, Comparative Study, Prediction on test using KNN

Both team members contributed equally to writing the report.