COVID-19 Outcome Prediction Analysis

Problem Statement

COVID-19 is a disease that has impacted hundreds of millions of people worldwide. The symptoms may range from mild to life threatening and that is due to various factors. The focus of our analysis is to predict the outcome of COVID patients based on a patient's characteristics such as age, sex, and so on. We created machine learning models that used select features to predict the outcome of a person who has been infected by COVID.

Data preparation

1.1 feature selection:

The following attributes were selected: 'age', 'sex', 'chronic_disease_binary', 'Confirmed', 'Deaths', 'Recovered', 'Active', 'Incident_Rate', and 'Case_Fatality_Ratio', 'longitude', 'latitude'.

1.2 mapping the features:

We mapped 'sex' to binary where female is '1' and male is '0'. It made sense to use either a '0' or '1' here because there were only two different genders distinguished. We also mapped 'chronic_disease_binary' with '0' for false and '1' for true. Using binary made sense because '0' represents false and '1' represents true to represent booleans. We mapped 'date_confirmation' with various numbers as there were many different dates.

1.3 balancing the classes in the training dataset:

We balanced the data using oversampling. This is because the majority class was 'hospitalized', so we needed a way to spread out the samples so that there would be less bias when training our models.
Before balancing:
1 13241
2 2974
0 997

After balancing:
1 13241
2 13241
0 13241

Classification models

1.4 building models and hyperparameter tuning:

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	leaf_size=10 n_neighbors=32 p=1	0.85	0.79	0.85
Random Forest	n_estimators= 143, min_samples_split =6, min_samples_leaf = 3, max_features= 6, max_depth= 15	0.95	0.93	0.95
SVM	kernel=linear c=0.7	0.78	0.61	0.78

KNN:
The first reason why I chose the KNN model was because it is intuitive to understand and implement. Another reason for choosing KNN is its relatively fast training time; when I tried to run SVM and MLP models, it took significantly longer to train the models. Lastly, because the model makes no assumptions about the data.

I chose k=10 and performed 10-fold cross validation. With 10-fold cross validation, we would take 10% of the data for testing then the other 90% will be used for training. I thought this was a reasonable number because of the size of our dataset.

For hyperparameter tuning, I tried RandomSearchCV initially to get an idea of the hyperparameter values then I used GridSearchCV. I used GridSearchCV because it is more thorough during its search. I tested the following ranges: leaf_size = 1 to 20, n_neighbors = 20 to 40, p = 1. I chose the ranges because of the RandomSearchCV results and because I thought the n_neighbours should be on the larger side to avoid overfitting. 30 to 40 seemed like a good range. Similarly, I tried different values for leaf_size and the range 1 to 20 seemed like the most reasonable one. I tried p = 1 and 2 initially, but p = 1 was always better so I only included p = 1 in the tuning file to save space and time. Another step I took was to standardize the features with

StandardScaler() because KNN is sensitive to the magnitude of the features.

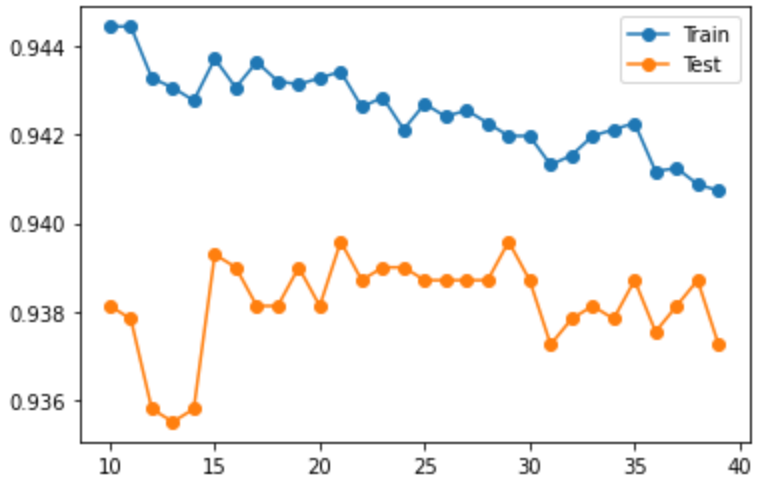
Random Forest:
I chose random forest as my model because overfitting can be controlled using hyperparameters and reduces errors caused by bias and variance. This is due to the model randomly choosing a subset of features for each tree and randomly choosing different data in the dataset. The randomness of choosing each data and features for each tree allows for training of different combinations and prevents repetition of choosing only some features and combinations. Additionally, class predictions are made based on majority vote from all the trained trees, which I believe will help determine a high accuracy for this dataset.

I chose k to be 10 because the models will be trained with a larger training data and smaller test data for testing. I think for this dataset having the models see more of the training data will help predict the test data better. I chose RandomizedSearchCV for hyperparameter tuning because it was faster compared to GridSearchCV. I think an exhaustive search is not necessary for this model because after a certain number of trees, the accuracy of the model decreases or remains the same, so doing an exhaustive search such as GridSearchCV didn't seem worthwhile to use for this model.

SVM:
While our dataset is not high-dimensional data per se, which SVM scales well with, it means that the model in the future can be scaled if more features are captured. The first reason why I chose SVM is that it is quite versatile, with regards to kernels at one's disposal, there are quite a few. While there were not any strong reasons outside of practical reasons why I chose a linear kernel in my SVM, it in my mind seemed to be a more simpler model which might help it not overfit. I initially tried an rbf kernel (radial basis function), the practicality of balancing both the c parameter and gamma parameter simultaneously was very tricky and I felt unsure if my model was overfitting therefore.

1.5 overfitting

KNN:
To avoid overfitting, I made sure to choose a K value that was large enough because for small K values and a lot of data, there is the risk of overfitting. I also performed some overfitting analysis by training the model for different values of the hyperparameter K and comparing their performances on both the train test data to check for overfitting.



Looking at the graph of their performance, the accuracy of the train and test data, the model does not seem to be overfitting because the accuracy of the train data is not increasing at the cost of the test data accuracy.

Random Forest:

To check for overfitting, I was comparing the accuracy score of the train set and the validation set using accuracy_score metric from sklearn . If the accuracy of the train set was significantly higher than the validation and test set then the model was overfitting. If the accuracy score was high for both train and validation set but not good for the test data then it also meant that the model was overfitting. Tuning the hyperparameters using RandomizedSearchCV helped a bit with this issue. After tuning the model, I was getting similar accuracy scores for the train and validation set.

SVM:

For my SVM I used a linear kernel. This is because it is concise when optimizing.
This is because with a linear kernel, one does not need to worry about optimizing both the C parameter and the gamma parameter, which one would have to do for different kernels, such as rbf, where not only would one need to optimize both the c and gamma parameters- they would need to be optimized simultaneously - that can be quite tricky. In SVM I tried to reduce the chance of overfitting by having a lower ‘c’ value, to decentivize overfitting. The c value is the penalty for a misvalued datum. If the penalty is high, the model will have a tendency to overfit. Too little and the model will underfit. Models with very large gamma values tend to overfit. So the gamma was not increased. For a linear kernel, I just needed to focus on the gamma parameter, which helped the optimization process.

1.6 comparative study

When comparing the macro F1-scores across the validation sets, the random forest has a high value, meanwhile the knn and SVM models have similar scores. Compared to the other models, random forest seems to be overfitting due to its high values for accuracy, but poor performance on the Kaggle test data. When comparing the F1-values for the 'deceased' class, we observed that the random forest has a high value while knn is lower and SVM has the lowest one.

Looking at how each model performed on each class, knn and SVM seemed to correctly predict the classes because the F1 scores for the classes are well spread out. However, knn had an appropriate accuracy, while SVM's was low and random forest's was too high. Additionally, all the models correctly predicted the majority class because all of them had a high F1 score on the majority class compared to the other classes.

Also, knn runs faster than SVM and random forest.

Our best performing model was the knn model because it gave the highest score on the Kaggle dataset.

Model predictions

1.7 Prediction on the test set

Base Performances:

KNN:

KNN:	precision	recall	f1-score	support
0	0.76	0.82	0.79	13241
1	0.93	0.92	0.92	13241
2	0.86	0.81	0.83	13241
accuracy			0.85	39723
macro avg	0.85	0.85	0.85	39723
weighted avg	0.85	0.85	0.85	39723

SVM:

SVM:

	precision	recall	f1-score	support
0	0.72	0.62	0.66	13241
1	0.82	0.97	0.89	13241
2	0.79	0.76	0.77	13241
accuracy			0.78	39723
macro avg	0.78	0.78	0.78	39723
weighted avg	0.78	0.78	0.78	39723

Random Forest:

RF:

	precision	recall	f1-score	support
0	0.90	0.96	0.93	13241
1	0.97	0.98	0.98	13241
2	0.98	0.91	0.95	13241
accuracy			0.95	39723
macro avg	0.95	0.95	0.95	39723
weighted avg	0.95	0.95	0.95	39723

From the outputs we can see that the knn and SVM model have F1 scores that are spread out for each class, while random forest is overfitting the data. Knn has a good accuracy while SVM's accuracy is low and random forest's accuracy is too high. So, knn was determined to be our best performing model.

Conclusion

In task 1.1, we chose the important features for the dataset, in task 1.2, we mapped the categorical features and in task 1.3 we balanced the imbalanced dataset by oversampling. We found that for our best performing model, which is the knn, gave the highest F1 score on the kaggle dataset without balancing the dataset, which we found interesting. In 1.4 we chose to build the knn, random forest and SVM models. In 1.6 and 1.7, we found that knn was our best performing model. We also saw that random forest was overfitting and SVM gave the lowest accuracy for the dataset. We realized that feature selection, having balanced data and choosing a suitable model is very important to succeed in this project.

Lessons learnt and future work

We learned that getting a high F1 score for unseen data is really hard to master and requires a lot of testing and understanding of the dataset. Getting the desired results requires choosing a good algorithm for the dataset, which requires research and good knowledge of different models. To improve myself in this type of project, we should work on it earlier and give ourselves more time to analyze the data.

References

https://towardsdatascience.com/how-to-effortlessly-handle-class-imbalance-with-python-and-smote-9b715ca8e5a7
https://medium.datadriveninvestor.com/k-nearest-neighbors-in-python-hyperparameters-tuning-716734bc557f
https://medium.com/analytics-vidhya/summary-of-knn-algorithm-when-used-for-classification-4934a1040983#:~:text=The%20main%20advantage%20of%20KNN,can%20be%20a%20suitable%20algorithm.
https://www.mygreatlearning.com/blog/knn-algorithm-introduction/
https://www.w3schools.com/python/python_file_write.asp
https://medium.com/analytics-vidhya/why-is-scaling-required-in-knn-and-k-means-8129e4d88ed7
https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74
https://machinelearningmastery.com/random-forest-ensemble-in-python/
https://www.quora.com/How-can-I-tell-whether-my-RanDom-Forest-model-is-overfitting
https://towardsdatascience.com/decision-trees-and-random-forests-df0c3123f991#:~:text=With%20that%20said%2C%20random%20forests,and%20therefore%20yield%20useful%20results.

Contribution

Mark: Worked on tasks 1.1 & 1.3 in milestone 1. Then worked on the SVM model for the final milestone.

Stephanie: Worked on tasks 1.6 & 1.7 in milestone 1. Then worked on the knn model for the final milestone.

Halakseka: Worked on tasks 1.4 & 1.5 in milestone 1. Then worked on the random forest model for the final milestone.

Everyone made equal contributions to the project.