CMPT 459 Final Milestone Report
Group members: Cheng Zhang 301364914, Zi Heng Wang 301303595, Shan Ying Yang 301305759
Submission Date: April 14, 2022

Project Title - Predicting the Outcomes of COVID-19 Cases

Problem Statement

The goal of this project is to predict the outcome group of confirmed COVID-19 cases into the following categories: 'hospitalized', 'non-hospitalized', and 'decreased'. To approach solving the problem, we have implemented the following three classification models: Random Forest classifier, k-Nearest Neighbor classifier, Support Vector Machines (SVM). Given the COVID-19 cases dataset, each of the models is trained against the preprocessed data in order to classify the untrained patterns. The performance of each model is measured by computing the corresponding F1-scores and accuracy scores. By comparing the performances of the three models we will be able to determine the model that best predicts the outcome groups for confirmed COVID-19 cases. We will also focus on choosing a model that best classified the dataset with a reasonable F1-Score.

Data preparation

The following datasets are used in this project: "cases_2021_train_processed_2" (training data), and "cases_2021_test_processed_unlabelled_2" (test data). The training data contains 17,212 rows of data, with 3,621 rows that contain a null 'province' value. The test data contains 4,304 rows of data, with 872 rows that contain a null 'province' value. Majority of the missing provinces for both training and test datasets are imputed using reverse geocoding with the given latitudes and longitudes. The remaining null values after imputation are kept in the datasets.

Feature Selection

All of the features in the train and test datasets are kept for this project. For both training and test data: The categorical features are : 'sex', 'province', 'country', 'data_confirmation'; The numerical features are: 'age', 'latitude', 'longitude', 'Confirmed', 'Deaths', 'Recovered', 'Active', 'Incident_Rate', 'Case_Fatality_Ratio'.

An additional feature of 'province_and_country'is added to both datasets, which concatenates each row's 'province' and 'country' values into one grouped string value. The training data's province_and_country' contains 148 unique values, and test data contains 49 respectively. A total of 16 features are used for the classification models.

Mapping the Features

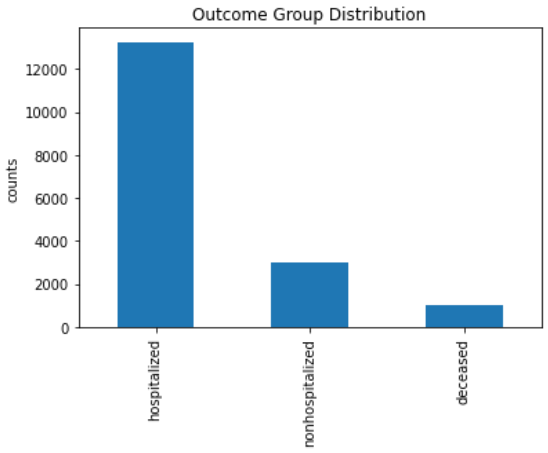
The categorical features for both training and test data are mapped to numeric values in the same manner. The feature for 'date_confimation' is first converted from string to DateTime object, and then converted to each date's corresponding ordinal values for future classification purposes. The rest of the features are first converted to Categorical objects, and then further encoded using pandas' pandas.Series.cat.codes. The 'outcome_groups' from the training data is also mapped to numerical values, where 0 represents 'deceased', 1 represents 'hospitalized', and 2 represents 'non-hospitalized'.

Balancing the Training Dataset

The training data has a skewed distribution of examples across the known classes, which is considered as an imbalanced dataset (Brownlee, 2020). Figure 1 displays the count distribution for the outcome

group classes across the training data. The class distribution shows that the distribution is extremely right-skewed, where the 'deceased' class is the minority class amongst the three classes for the outcome group. The approximate distribution proportion for 'deceased' to 'hospitalized' is around 0.22:1.

Figure 1. Imbalanced Class of Outcome Group



The minority class, 'deceased' is an important class to predict. Training on imbalanced data could lead to inaccurate predictive models. To address the problem of imbalance dataset, we used SMOTE to oversample on the outcome group, where the data is resampled by the minority class 'deceased'. The imblearn.Pipeline object is used to handle the SMOTE sampler with GridSearchCV. The Pipeline object allows the sampler to be executed before each fit on the train data.

Classification models

Building Models and Hyperparameter Tuning

Random Forest is one of the models we chose to perform classification tasks for this project. With Random Forest classifier, the model bags a subset of decision trees upon building itself, and then chooses the best output by majority vote (by probability). The primary reason for using the Random Forest classifier is because it is more accurate and effective in handling large datasets, as well as more effective in dealing with outliers, and missing values. Random Forest classifier is also suitable to use when the data contains both categorical and numerical features. Random Forests' bagging technique randomly samples features when creating each decision tree, which reduces variance and avoids overfitting when the model trains on the training data. In short, Random Forest is chosen over other models like Decision Trees and Naive Bayes because it has higher accuracy and efficiency when performing classification tasks. According to Lemons (2020), Random Forest classifier tends to perform better than Naive Bayes method with appropriate feature selections. Considering that the COVID-19 dataset for this project is relatively large with 17,212 rows of data, Decision Trees classifier is not considered because it will have higher bias and variance when training on large datasets.

With a support vector machine, the model uses a subset of training points in the support vector to get the best output. It maximizes the distances between nearest data point and hyper-plane will help us to decide the right hyper-plane. The pros of the support vector machine are relatively high efficiency in the high dimensional spaces, memory efficiency. However, the performance of this model does not work well when we have a large dataset because the required training time is high. Considering that the COVID-19 dataset for this project is relatively large, we decided to train the data using SVC to see the performance and training time in a large dataset compared to other methods.

KNN is the third model we chose for our classification tasks. KNN works by classifying each data point based on its k closest neighbors. The number of neighbors, k, is the main hyperparameter in KNN for fine tuning and is named "n_neighbors" in sklearn. The resulting class is chosen by either majority vote of its k neighbors, or by weighing each of those votes based on their distance. This is configured in a parameter called 'weights' in sklearn's KNeighborsClassifier. This classifier is well suited to large datasets because of its relatively low time complexity. The time complexity of KNN depends on whether it's using the ball_tree', 'kd_tree' or 'brute force' algorithm. Changing the algorithm will not affect the classification result so we have sklearn decide it automatically based on the input passed. The

third important parameter we configured is p, which specifies Manhattan distance when p=1 and Euclid distance when p=2. We fine-tuned hyperparameters, n_neighbors, weights, and p using GridSearchCV.

GridSearchCV was used for hyperparameter tuning for the Random Forest model. The number of folds, K, was initialized as a Stratified K-Fold cross-validation object. The value for K was set to 10 since we are dealing with a larger training set, as we hope to reduce prediction error and allow the model to train on more available data with Random Forest. In addition, GridSearchCV also performs cross-validation, so we do not have to split the data using "train_test_split" as it automatically splits the passed in data into training and validation sets. The two tuning parameters were "n_estimators" and "max_depth". The range of "n_estimators" varies between 30 to 850. The number of trees is maxed to 850 as we hope to see if there will be any improvement in the f1-score on the validation sets upon generating more trees. The range of "max_depth" was narrowed down in between 80 to 140, because initial evaluation shows lower max_depth tends to have poorer performance as the tree is not grown deep enough. Different values for "min_sample_leaf", "max_features", and cpp_alpha" were also tested to compare the model performance. Upon testing higher "min_sample_leaf" values, it was determined that value resulted in an underfitting of the training dataset. The value of "min_sample_leaf" was tested between 2 and 3, the value is set small to avoid overfitting the training dataset. The hyperparameter "max_features" is set to "log2", which limits the number of features to check for the best split. The hyper parameter of "ccp_alpha" refers to the "complexity parameter used for Minimal Cost-Complexity Pruning" (Pedregosa et al.), which is essentially the punishment parameter that prunes decision trees within the forest. The value for "cpp_alpha" was set to 0.0001 because setting it higher pruned too many trees resulting the model to underperform, while setting it lower pruned too few trees causing the model to have inaccurate train score.

GridSearchCV was used for hyperparameter tuning for the SVM model. The number of folds, K, was initialized as a Stratified K-Fold cross-validation object from the scikit-learn library. After several training, l find that the model has highest accuracy when k = 6. In addition, l also created a pipeline to apply StandardScalar and SMT to resample the imbalance data. The two tuning parameters of the SVC model are model__C and model__gamma, which defines the regularization parameter and kernel coefficient of the model. I test several kernel types like "sigmoid", "poly" ,"linear" and "rbf". The model has highest accuracy when the C equals to 8, gamma equals to 0.01 and kernel type is "rbf". In this process, l find that the training time for "linear" and "sigmoid" is significantly higher than "rbf", and the mean test f1-score is mainly affected by the strength of regularization.

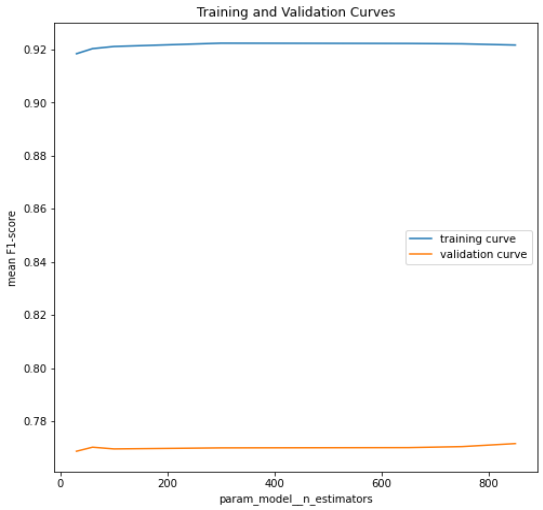
Table 1. Classification Model Results with Hyperparameters

Model	Hyperparameters	Mean macro F1-score on validation sets	Mean F1-score 'deceased' on validation sets	Mean accuracy on validation sets
Random Forest	n_estimators: 650, max_depth: 100, max_features: log2, min_samples_leaf: 3, min_samples_split: 2, ccp_alpha: 0.0001	0.77836	0.48580	0.92900
SVM	C: 8, Gamma:0.01	0.788	0.523	0.9318
KNN	n_neighbors: 9, p: 2, weights: 'distance'	0.96719	0.97137	0.97381

Overfitting

Figure 2 displays the training and validation curves for the Random Forest classification model. By comparing the training and validation accuracy for the model, the model is not considered overfitting. The best estimator for Random Forest has a mean f1 score of 0.91 and 0.78 for training and validation scores respectively. Generally, classification models perform better in predicting the training data because it learns it too well. Although the difference between the training score and the validation score is as large as 0.09, the overall shape and trend of the curves in Figure 2. look really similar, indicating that the model is not overfitting.

Figure 2. Random Forest's Training and Validation Curves



For overfitting and fine-tuning the KNN model, GridSearchCV is used. One interesting thing we found is that the optimal value for n_neighbors and parameter p keeps changing each time we run GridSearchCV on the KNN Classifier. We found this is because as n_neighbors increase to over 7 or 8, the mean_test_accuracy and f1_score begins to flatten in the plot. (Shown in figures below) Furthermore, we found the score suffered when n_neighbors was below 7 and the models with weights set to 'distance' are significantly better scoring than the models with weights set to 'uniform'.

Figure 3. KNN's Mean Test Accuracy

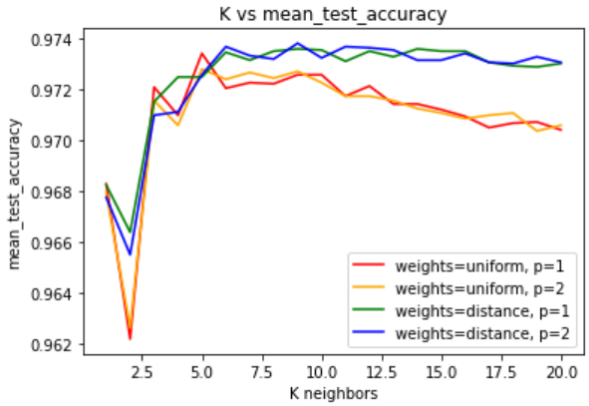
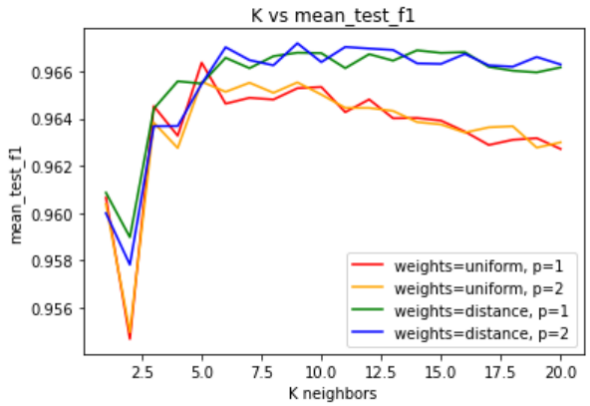


Figure 4. KNN's Mean Test F1-Score



Comparative Study

Table 2 shows each model's scores on training and validation sets.

Table 2. Model Score Comparisons on Training and Validation Sets

	Random Forest		**SVM**		**KNN**	
	Train	**Validation**	**Train**	**Validation**	**Train**	**Validation**
F1-Score	0.914	0.778	0.796	0.788	0.998	0.967
F1-deceased	0.793	0.486	0.546	0.523	0.998	0.971
F1-hospitalized	0.989	0.980	0.984	0.983	0.998	0.989
F1-non hospitalized	0.960	0.869	0.856	0.853	0.998	0.941
accuracy	0.972	0.929	0.934	0.932	0.998	0.974

Random Forest

The overall accuracy for the Random Forest model differed by 0.043, which is a reasonable gap. The macro F1-score for training and validation sets differed by 0.136, with training score scoring higher. This could be resulted from low value of "ccp_alpha", as setting the cost complexity pruning higher could help in matching the train and validation score. In terms of F1-Score on the three classes, the training set scored higher than the validation set by 0.307. The large difference between the training set and validation sets' F1-Score on deceased resulted from training on the imbalanced data. Although we have oversampled the training data, it would still affect the overall performance in correctly classifying the "deceased" label as the class data for it might not be diversified enough.

An advantage of Random Forest is that the classifier has high accuracy when predicting large datasets. Random Forest uses the bagging techniques, which works well to classify data that contains both categorical and numerical features. We can see from Table 1 that the mean accuracies for the Random Forest model on training and validation set are quite consistent, each maintained above 92%. Another advantage of the Random Forest classifier is that the model can handle overfitting well as it generates different decision trees to classify the training data. From Figure 1 we can see that the overall trend of the validation curve is much similar to the training curve. However, a major disadvantage of Random Forest is its high computational time, which could miss the opportunity to find the optimal hyperparameter. Random Forest has a relatively high computational complexity, especially when it builds on categorical features that have high levels (i.e. more classes). The computational time for each build increases as the number of specified trees to build increases. The long running time made it difficult to better tune the hyperparameters, as it would take more than an hour to classify when the "n_estimators" parameter is over 500.

SVM

The advantage of SVM is that it works really well for high dimensional data but it does not perform well when we have a large dataset because the required training time is higher. Due to the relatively large dataset of COVID-19, SVM is not the best model to train the data.

KNN

The advantage of KNN is the training time is very fast. One disadvantage I found is hard to manage overfitting, even when the train data is split into train and validation sets. This is because the resulting validation and test scores are unreasonably high (> 94%) and when we submitted our predictions on kaggle, the actual score was about 75%.

Best Model

The best model for classifying the COVID-19 dataset is chosen to be the Random Forest model. The private score for the Random Forest model scored 0.81631. Considering the objective to classify a dataset that contains both numerical and categorical features, Random Forest seemed to be a more suitable classifier than the SVM model, as SVM is more suitable to treat binary classification problems.Considering the size of the dataset and a small subset of null values in the data, Random Forest is also chosen over the KNN model because it handles large dataset and missing values well. KNN, however, is a more suitable model for smaller datasets with less noisy data. KNN generally does not work well with large datasets as the cost to calculate the distances is much higher. Notably, the test data's private Kaggle scores for KNN and Random Forest are 0.78432 and 0.81631, respectively. Therefore, even though KNN had a much higher train and validation score, we have decided to stick with the Random Forest model that has more consistency.

Model predictions

Random Forest Classifier: Figure 5 shows the output of Random Forest model's mean macro F1-Score and F1-Score for all three classes on validation sets. The overall macro F1-Score is 77.8% The model's F1-Score for "deceased", "hospitalized", and "non-hospitalized" are 48.6%, 98.0%, and 86.9%, respectively.

Figure 5. Random Forest's Validation Performance

```
best mean_test_f1:  0.7783571540034184
best mean_test_f1_deceased:  0.4858000395170077
best mean_test_f1_hospitalized:  0.9800315558575796
best mean_test_f1_nonhospitalized:  0.8692398666356681
```

SVM: Figure 6 shows the output of SVM model's mean macro F1-Score and F1-Score for all three classes on validation sets. The overall macro F1-Score is 78.6% The model's F1-Score for "deceased", "hospitalized", and "non-hospitalized" are 52.3%, 98.3%, and 0.853%, respectively.

Figure 6. SVM's Validation Performance

```
best mean_test_f1:  0.7865292329684057
best mean_test_f1_deceased:  0.5238455690701832
best mean_test_accuracy:  0.9308617754450134
best mean_test_f1_hospitalized:  0.9834768638270891
best mean_test_f1_nonhospitalized:  0.8522652660079447
```

KNN: Figure 7 shows the KNN model's macro F1-Score, test accuracy and F1-Score for all three classes validation sets. The overall macro F1-Score is 96.7%. The model's F1-Score for "deceased", "hospitalized" and "non-hospitalised" are 97.1%, 98.9%, and 94.1% respectively.

Figure 7. KNN's Validation Performance

```
best mean_test_f1:  0.9671964572939
best mean_test_f1_deceased:  0.9713747634330556
best mean_test_f1_hospitalized:  0.9890515497520868
best mean_test_f1_nonhospitalized:  0.9411630586965577
best mean_test_accuracy:  0.9738168951791243
```

Conclusion

The process of predicting the COVID-19 outcome groups was done in three stages: data preparation, data classification with different models, and evaluating the model results. During data preparation, missing data for the "country" feature is imputed using reverse geocoding, and a combined key is added to the data. Imbalance training data is also handled by oversampling the dataset with SMOTE. During data classification, the three models of Random Forest, SVM, and KNN were used to classify the data. Different hyperparameters were also used to tune each model in order to find the best performing classifier.

Upon building and evaluating the three models, the best classification model for this project is the Random Forest model in terms of the nature of the training dataset. Overfitting in Random Forest is negligible, but it would be worthwhile to increase the cost complexity pruning for the model to pull the training and validation score closer. It is notable that Random Forest and SVM had similar performance in terms of the validation and training score, but Random Forest model was chosen considering SVM is not as suitable for multiclass problems. Interestingly, even though both of the SVM and KNN models had higher mean validation scores than the Random Forest model, their actual test scores on the Kaggle competition are lower. Considering KNN's Kaggle submission score has a large difference to its own validation macro F1-Score, we have decided to stay with the Random Forest model that has a more consistent score.

Lessons learnt and future work

A lesson learnt from this project is that more careful data preprocessing and appropriate feature selections could improve model performance. As we tuned the hyperparameters when building the model, we have realized that the model performance stabilizes after certain sets of hyperparameters are reached. For example, the training and validation curves for the Random Forest models are flattened after the number of trees exceeded 60. Therefore, more appropriate feature selection would definitely help in boosting the accuracy of all models.

We also learned that imbalanced dataset has largely impacted model performance. It is notable in Random Forest that the model's accuracy in predicting the minority class is low even after resampling the data. This could be due to the fact that we resampled all classes, perhaps resampling by minority would increase the overall accuracy in predicting the "deceased" class. Choosing appropriate hyperparameters is also crucial for building each model. For example, setting the appropriate range of estimators, neighbors, and C can effectively improve runtime in building and searching for the best estimator.

In future work, it is necessary for us to appropriately preprocess the dataset and perform adequate feature selection prior classifying the data. For example, we could try adding additional features that combine different features and see if it helps in improving the accuracy. We can also improve model performance by choosing the right way to resample the dataset. For instance, we could seek ways to better resample the data through either oversampling by minority. Lastly, choosing appropriate hyperparameters would be another critical step to effectively testing our models.

References

Brownlee, J. (January 14, 2020). A gentle introduction to imbalance classification. *Machine Learning Mastery*. https://machinelearningmastery.com/what-is-imbalanced-classification/

Huneycutt, J. (May 18, 2018). Implementing a random forest classification model in python. Medium. https://medium.com/@hjhuney/implementing-a-random-forest-classification-model-in-python-583891c99652

Lemons, K., 2020. A Comparison Between Naïve Bayes and Random Forest to Predict Breast Cancer. *International Journal of Undergraduate Research and Creative Activities*, 12(1), pp.1–5. DOI: http://doi.org/10.7710/2168-0620.0287

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Contribution

Shan Ying (Felicity) Yang

- Data preprocessing and feature selection in milestone 1 and final milestone, i.e. data cleaning, imputing missing values, dealing with outliers, adding combined keys for models
- Balancing the classes in the training dataset.
- Implementation of Random Forest Classifier, which includes building the model, evaluating model performance, tuning appropriate hyperparameters, and producing and analyzing model results.
- Writing report

Cheng Zhang

- Tackle imbalance data
- Implementation of SVC, which includes building the model, evaluating model performance, tuning appropriate hyperparameters
- Feature selection in milestone 1 and final milestone
- Writing report

Zi Heng Wang

- Implementation of KNN Classifier, which includes building the model, evaluating model performance, tuning the hyperparameters, and producing and analyzing the results
- Participated in data cleaning, imputing missing values, and outlier handling in milestone 1
- Participated in writing documentation and report