CMPT 459 - Special Topics in Data Mining

COVID-19 Outcome Prediction with Machine Learning

Final Project Report

Armin Hatami, ahatami@sfu.ca, 301278375
Adam Watson, ajw32@sfu.ca, 301457336
Minh Thanh Tran, mtt6@sfu.ca, 301416848

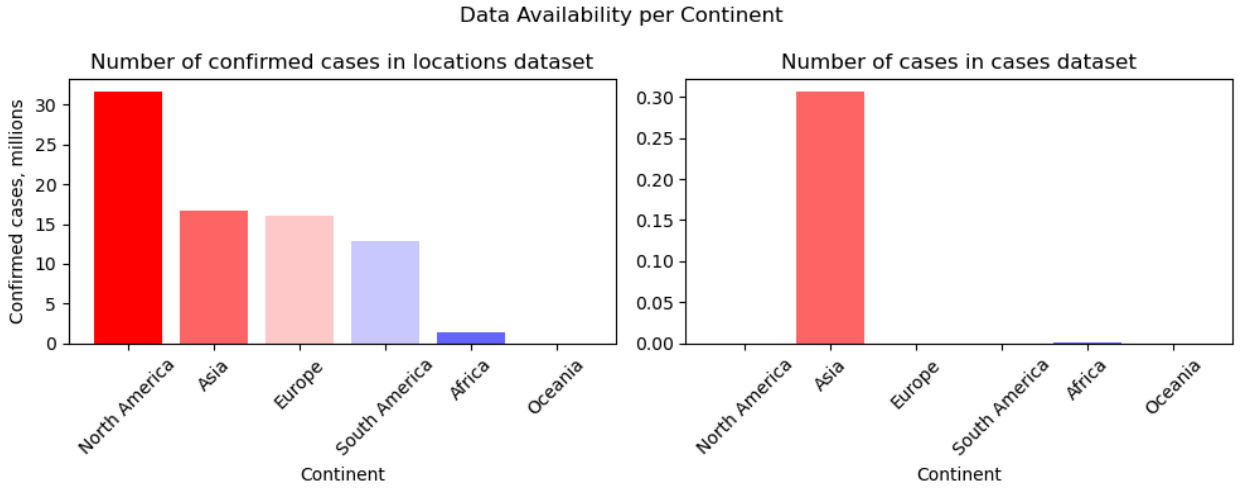
April 9th, 2024

Problem Statement

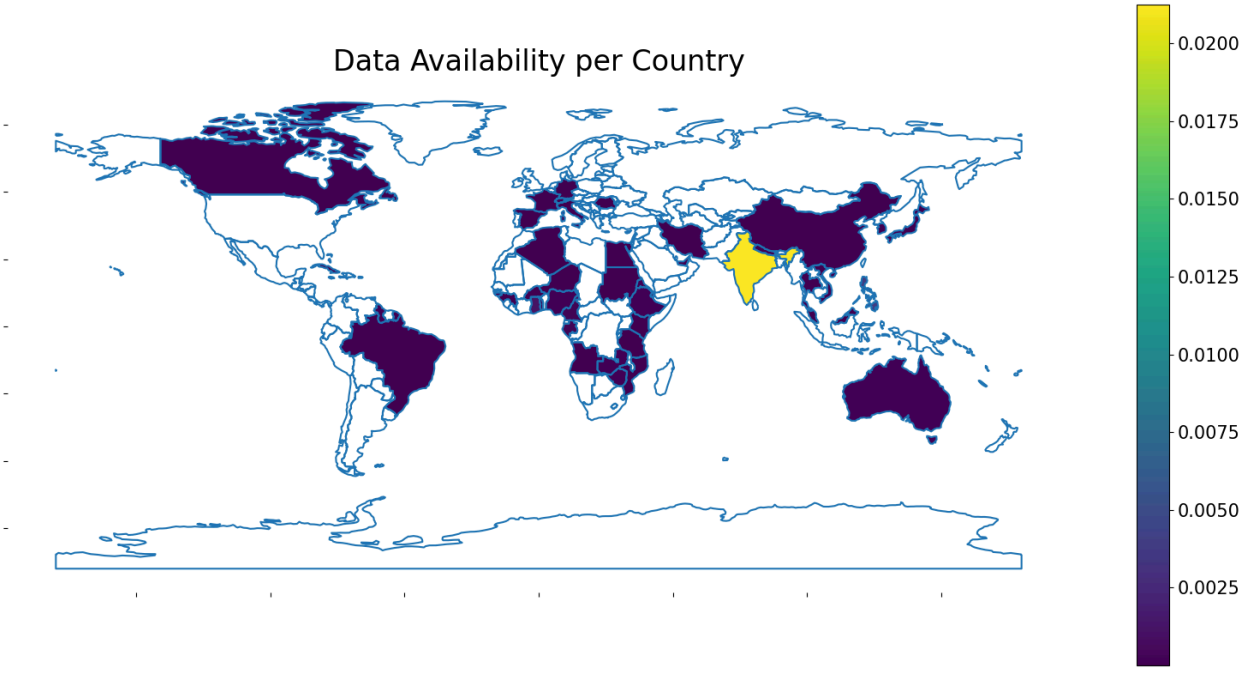
This project leverages data mining techniques to develop and evaluate classification models for predicting COVID-19 case outcomes based on demographic and geographic information. With a dataset from the year 2021 describing patient attributes such as age, sex, and location, the objective is to accurately classify cases into categories of "hospitalized," "non-hospitalized," or "deceased." Given the critical nature of this task, the focus lies on optimizing model performance, particularly in minimizing false positives and negatives. Through hyperparameter tuning, cross-validation, and model comparison, this project seeks to identify the most effective model for predicting outcomes of COVID-19 cases.

Data Exploration and Visualization

We began our data exploration by plotting the data availability by continent. North America has the highest number of confirmed cases, which could mean that either the disease is more prevalent in that continent or the countries of that continent on average, have a better capability



of detecting positive COVID-19 cases (more testing, higher test accuracy). Countries may also be underreporting for political reasons. In the cases dataset (train and test), most records are concentrated in Asia, particularly India, and Africa is a distant second. We plotted a world heatmap of the cases dataset data availability normalized to country populations to better understand the location distribution of the cases data. We can immediately see that India has the highest data availability. Countries that have some data availability tend to have the same level of availability. The cases dataset has zero data availability for the majority of the countries. Some



anomalous locations that were detected during the dataset merging process were the cruise ships Diamond Princess and MS Zaandam, The Holy See (a.k.a The Vatican) and the Summer Olympics in 2020.

Data Preprocessing

We began with converting *date_confirmation* from the cases dataset. The feature was provided in string format, DD.MM.YYYY and required text processing to be converted to a DateTime object. Next, we aggregated the outcomes into categories of **Hospitalized/Under Treatment**, **Recovered**/**Discharged**, and **Deceased**. The *expected_mortality_rate* was determined to be the *case_fatality_ratio* feature from the locations dataset. The cases and locations datasets were then joined on the *country* and *province* keys in order to add the *expected_mortality_rate* to the dataset. The *latitude* and *longitude* data were converted into radians for clustering and classifier training purposes. We also converted *age* from string to integer values by creating a set text-processing decision rules to deal with issues such as number ranges, decimal points and NaNs.

Further analysis showed us that more than 87% of feature values for *age*, *sex*, and *additional_information* were missing. We chose to use the IterativeImputer from *scikit-learn* for imputing *age*, *sex,* and other numerical features. IterativeImputer is a multivariate imputer that fits a regressor for every missing value using other features in a round-robin fashion. We chose it because of its performance on different toy datasets [1]. *Country* and *province* were one-hot encoded, therefore missing values were considered a feature.

Data Preparation

A common issue we dealt with was needing an established country name in the datasets for countries such as Vietnam, Taiwan, and South Korea. We leveraged *thefuzz* python library [2] to merge the external datasets to the provided one.

Task 3 - Feature Engineering and Selection

Starting with the provided datasets, we dropped *Last_Update* and any redundant columns from the Locations dataset. The *Last_Update* feature describes the file's last updated time for the respective rows, which included nine unique timestamps and spanned from August 2020 all the way to August 2021. Since both the train and test sets only covered cases that occurred in 2020, we created an ordinal numerical feature called *month_confirmation,* which describes the month the case was observed. We can then drop *date_confirmation* as the year and day-of-year data do not provide valuable information.

In addition to the provided data, we integrated three other readily available datasets to improve the predictive power of our classifiers. The three different datasets contained information about the number of hospital beds per 10,000 people, population density, and population income. We hypothesize that a higher number of beds per 10,000 people, a lower population density, and a higher population income will correspond to a lower mortality rate and vice versa.

We merged the Locations and Cases_2021 datasets to get the province capital's coordinates. Using the capital and case coordinates, we created a new feature called distance_to_province_capital with the Haversine formula [3]. We assume that the further away you are from the province's capital, the fewer healthcare resources you have access to, and therefore, an increased mortality risk. In the Locations dataset, province-level information was limited to a subset of the countries, and for this subset, the distance_to_province_capital actually corresponds to the distance to the country capital. An auxiliary boolean feature was created to describe this process named *no_province* and was added to the dataset.

Task 4 - Feature Mapping

Categorical attributes of *sex*, *chronic_disease_binary*, *province*, *country*, *income_group*, and *outcome_group* were converted into numerical features using different techniques. *Sex* and *chronic_disease_binary* were mapped using definitions **{'male':0, 'female':1}** and **{FALSE:0, TRUE:1}** respectively. *Province* and *country* were one hot encoded using *scikit-learn*'s OneHotEncoder (OHE), which created a new feature for every unique value in each attribute. This OHE implementation allows us to deal with unobserved values during testing time by creating an additional boolean feature for infrequent values [4]. *Income_group* was converted using an ordinal integer mapping, **{'Low income':0, 'Lower middle income':1, 'Middle income':2, 'Upper middle income':3, 'High income':4}**, to preserve the ordinal nature of the feature. Finally, the class labels for *outcome_group* were mapped as described in the project description.

Task 5 - Imbalanced class labels

For the Random Forest model, we used the *class_weight* attribute from *scikit-learn* [5]. This allowed us to adjust our predictions either by weighing them inversely according to the frequency of each class in the input data or by weighing them inversely based on bootstrapped samples. By doing so, the dataset remained unchanged while effectively mitigating the impact of class imbalances on our model's performance.

For the other 2 classifiers, we employed the Synthetic Minority Over-sampling Technique. SMOTE generates synthetic samples for the minority class by interpolating between existing minority class instances, effectively balancing the class distribution. By synthesizing new instances rather than simply duplicating existing ones, SMOTE helps mitigate the risk of overfitting and improves the generalization capability of the model. Additionally, SMOTE helps prevent bias towards the majority class, ensuring that the model's predictions are more reflective of the true underlying distribution of the data. However, in practice, we found that it distort the original distribution of the dataset, leading to a potentially inaccurate representation of the underlying population distribution.

Before SMOTE:	Class 0: 798, class 1: 10592, class 2: 2379.	Total: 13769
After SMOTE:	Class 0: 10592, class 1: 10592, class 2: 10592.	Total: 31776

Classification Models

The three classification algorithms we used are **Random Forest**, **Support Vector Machine**, and **Multi-Layer Perceptron**.

We chose the **RandomForest** (RF) Classifier algorithm because of its proven track record in classification tasks with tabular data [6]. The RF algorithm's ability to handle high-dimensional data without overfitting, robustness to outliers, and parallelizable nature make it appropriate for our problem statement. Additionally, we can utilize the feature importance measure to provide some interpretability for our model, which can be useful for feature selection. RFs may struggle with imbalanced datasets; however, we can use under/oversampling methods before or during the bagging process to balance the distribution of the classes.

We chose **SVM** because our preprocessed dataset has a very high dimensionality. SVMs are effective in high-dimensional spaces, making them suitable for datasets with a large number of features.. SVMs are less prone to overfitting, especially in high-dimensional spaces, compared to some other models like decision trees or neural networks. SVMs generalizes well to unseen data and can effectively handle non-linear relationships between features and target variables through the use of kernel functions, allowing them to capture complex patterns in high-dimensional data. SVMs can handle datasets with many irrelevant features, which might be the case here, because. SVMs focus on the support vectors, which are the most informative data points, and ignore the rest, making them robust to noise and irrelevant features. Moreover, in high-dimensional datasets, it's common for data to be sparse, and although we already preprocessed the sparse data away, SVMs perform well with them because they only rely on the support vectors.

We chose **Multi-Layer Perceptron** (MLP) due to its high performance with high-dimensional datasets. The size of the given training data has some benefits and some drawbacks – the dataset is not too large as to make the training time too long, but one of MLP's strengths is gaining accuracy in very large datasets. Since our focus is on performance and not interpretability, MLP's low interpretability matters less. There are many hyperparameters that can be tuned in an MLP model, such as the number of neurons in each layer of the model. The fact that there is so much customizability with hyperparameters means that a properly tuned model should get good performance. To create our MLP model, we used the scikit-learn library [7].

Task 6 - Model building and Hyperparameter Tuning

RandomForest

f1_macro	0.780236
f1_deceased	0.497151
f1_hospitalized	0.980513
f1_nonhospitalized	0.863043

We used Bayesian Optimization to perform hyperparameter tuning and *k*-fold cross-validation. We set *k* to 5 to reduce the computation time while producing consistent and reliable out-of-fold evaluation metrics. We conducted five searches with 50 iterations and used the macro F1-score as the search evaluation metric. The hyperparameters of interest and their

justifications are in the table below. The value ranges were determined with 10 iterations of Randomized Search, and the minimum values for *min_samples_split* and *min_samples_leaf* were set to 5 to avoid overfitting individual trees.

Hyperparameter	Value Range	Reasoning
n_estimators	10 - 3000	Generally, increasing *n_estimators* improves performance, but there is a point of diminishing return.
max_depth	3 - 30	Shallow trees may not capture the generalizations of the data, while deep trees are prone to overfitting.
min_samples_split	5 - 30	Prevents overfitting by regulating the minimum number of samples required to split a node. Large values may lead to underfitting.
min_samples_leaf	5 - 30	Prevents overfitting by regulating the minimum number of samples to form a leaf node, and can absorb noisy data points. Large values may lead to underfitting.
max_features	'auto', 'sqrt', 'log2'	Exploring different strategies for the Feature Randomness process.
class_weight	'balanced' 'balanced_subsample' None	Addresses class imbalance issues by assigning different weights to classes. 'balanced' adjusts weights globally, while 'balanced-subsample' extends this to individual trees.

Support Vector Machine

We used K = 5 because it is a commonly used value for cross-validation. It strikes a balance between computational efficiency and robustness of the model evaluation.

We used GridSearchCV for hyperparameter tuning. GridSearchCV exhaustively searches through a specified grid of hyperparameters and evaluates the model's performance using cross-validation on each combination of hyperparameters. It's a systematic approach that evaluates all possible combinations of hyperparameters, making it suitable for small to medium-sized hyperparameter spaces (12 combinations in this case). We will be guaranteed to find the optimal combination of hyperparameters within the specified grid.

```
F1: 0.7907 - F1 Deceased: 0.6788 - F1 Hospitalized: 0.9043 - F1 Nonhospitalized: 0.7890
```

Hyperparameters	Value Range	Reasoning
C (Regularization Parameter)	0.1, 1, 10	The regularization parameter C controls the trade-off between maximizing the margin and minimizing the classification error. By trying the most common values of C, We explore different regularization strengths, allowing the model to adapt to the training data while avoiding overfitting or underfitting.
Gamma (Kernel	0.1, 1, 'scale', 'auto'	We included manual and auto values to cover most possible

Coefficient)		cases of best gamma values
Kernel	'rbf', 'poly', ‘sigmoid’	Linear, RBF (Radial Basis Function), Sigmoid and Polynomial kernels are commonly used options. We opted to remove Linear as it is unlikely that there is a linear relationship in the dataset.

Multi-Layer Perceptron

```
f1_deceased:  0.45575743938797314
f1_hospitalized:  0.8859357696567
f1_nonhospitalized:  0.7750235331032319
f1_macro:  0.705572247382635
```

We set k to 5 for our k-fold cross-validation as it does not take too much computing time, while also being accurate. We used RandomSearchCV to tune the hyperparameters of our MLP model. For this, we used the deceased f1-score as our primary scoring criteria, as this was the most difficult metric to obtain a good score for. We opted not to use GridSearchCV as the search space was far too large due to the amount of hyperparameters that were tuned.

Hyperparameter	**Value Range**	**Reasoning**
learning_rate	‘constant’, ‘invscaling’, ‘adaptive’	Only used when solver=’sgd’. This controls the learning speed.
alpha	0.000001 – 0.1	This is a penalty term that combats underfitting and overfitting. We choose a larger range for proper regularization.
activation	‘logistic’,’relu’,’tanh’,’linear’	Each activation function is significantly different from another, so I included them all in my range of values.
solver	‘lbfgs’,’sgd’,’adam’	This parameter optimizes class weights. I chose to include all solvers as the dataset is significantly unbalanced, making the right choice of solver important.
learning_rate_init	0.0001 – 0.01	Controls initial step-size of the optimizer. The default is 0.001. We chose this range as more extreme values would result in bad optimization or take too long to compute.
hidden_layer_sizes	2 layers: (100,25) – (150,100) 3 layers: (100,50,25) – (200,150,100)	We chose between 2 and 3 hidden layers because too many hidden layers would lead to overfitting.

Results

Model	Hyperparameters	Average Macro-F1 Score	Average 'Deceased' Macro-F1 Score	Mean Accuracy
RandomForest Classifier	n_estimators = 3000 max_depth = 30 min_samples_split = 5 min_samples_leaf = 5 max_features = 'auto' class_weight = 'balanced_subsample'	0.780	0.497	0.929
Support Vector Machine	C: 0.1, gamma: 0.1, kernel: 'sigmoid'	0.791 (on SMOTE data)	0.679 (on SMOTE data)	0.798 (on SMOTE data)
Multi-Layer Perceptron	alpha=0.5, hidden_layer_sizes=(150, 125, 100) learning_rate='adaptive', learning_rate_init=0.0001, solver='lbfgs'	0.799 (on SMOTE data)	0.707 (on SMOTE data)	0.803 (on SMOTE data)

Task 7 - Overfitting

RandomForest

Previously, we used the *min_samples_split* and *min_samples_leaf* hyperparameters to mitigate overfitting by controlling the minimum number of samples needed to form a split or a node. We evaluated the average macro F1-score through 5-fold cross-validation, resulting in 0.78 and 0.77 for the train and test sets, respectively. In addition, we computed the F1 score specifically for the deceased class and obtained comparable results. The F1 scores for the deceased class were consistent with the overall evaluation, yielding similar values of 0.52 and 0.50 for the train and test set, respectively. The difference between the F1 scores of the train and test sets suggests that overfitting is not a significant issue.

Support Vector Machine

In SVMs, the regularization parameter, C, controls the trade-off between maximizing the margin and minimizing the classification error. A smaller value of C results in a wider margin and more tolerance for misclassifications, which can help prevent overfitting by reducing the influence of individual data points [8]. Since our selected value for C is only 0.1, the risk of overfitting is low. The difference between the F1 scores of the train and test sets once again suggests that overfitting is not a significant issue.

Multi-Layer Perceptron

To reduce the risk of overfitting, we never used more than 4 layers of neurons to test our model. The more layers in the neural network, the higher the model complexity, which increases the risk

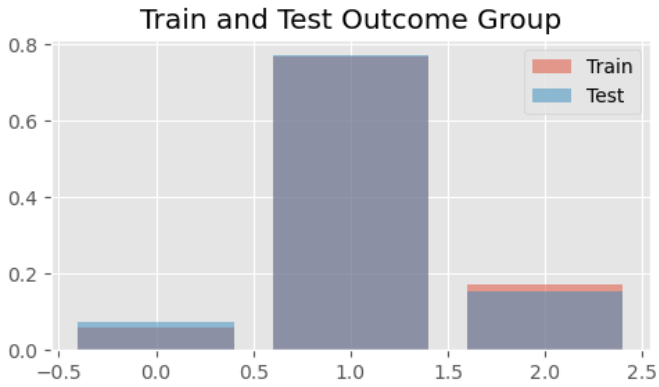
of overfitting. We also evaluated the average macro F1-score, which was 0.767 in the training set and 0.761 in the test set.

Task 8 - Comparative Study

The results table shows that the RF classifier had a similar macro-F1 score on the non-sampled dataset compared to the MLP and SVM classifiers on resampled data. The MLP and SVM classifiers had a higher mean F1-score on 'Deceased' and mean accuracy, however, these metrics are distorted as the dataset size has been significantly changed. When comparing the classifiers that trained on resampled data, the MLP classifier achieved a higher F1 score on the deceased class than the SVM classifier. We chose the RF classifier to make the predictions for submission as it had a similar score to the SVM and MLP classifiers without resampling the original dataset. Resampling is experimental and there is no one-for-all solution, as seen in this survey of various techniques used in unbalanced datasets [9].

Model Predictions

We plotted the normalized distribution of the test predictions against the training *outcome_group*. Assuming the distributions of the train and test data are the same, our model tends to overpredict the deceased class while underpredicting non-hospitalized. The Kullback-Leibler Divergence (KLD) score between the two distributions is 0.003. This supports our assumptions in the overfitting section; our model has not overfit.



Conclusion

We trained three different classifiers to predict COVID-19 case outcomes. We integrated three different external datasets to improve the predictive power of our classifiers. Based on our findings, the RF classifier outperformed the SVM and MLP classifiers. We also had higher confidence in the RF classifier's performance metrics as it is based on the original data records without any resampling. Resampling using SMOTE led to classifiers with higher macro-F1 scores and F1 scores on the deceased class; however, it also reduced the mean accuracy. SMOTE creates synthetic samples, which might not always represent the test data distribution.

Lessons Learnt and Future Work

We learned that traditional metrics such as accuracy do not accurately assess model performance on minority classes when dealing with imbalanced datasets. Instead, prioritizing metrics such as the macro and individual F1 scores and area under the ROC curve can offer more valuable results. We also experimented with resampling techniques such as oversampling and undersampling to deal with class imbalance. Some techniques remove unique data records, while others synthesize new points. We learned that as resampling changes the sizes of our dataset, our

evaluation metrics become distorted and harder to compare. For future work, we recommend stacking models and creating a meta-learner based on the prediction probabilities of each model. This is a rather brute-force approach commonly used in many Kaggle competitions. Integrating additional datasets for the feature engineering step improved our CV scores by a few percentages. One can also incorporate datasets for other topics, such as cigarette and smoke-related product consumption and economic indices.

References

1. https://scikit-learn.org/stable/auto_examples/impute/plot_iterative_imputer_variants_comparison.html#sphx-glr-auto-examples-impute-plot-iterative-imputer-variants-comparison-py
2. https://github.com/seatgeek/thefuzz
3. https://en.wikipedia.org/wiki/Haversine_formula
4. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html
5. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
6. https://arxiv.org/abs/2207.08815
7. https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
8. https://www.geeksforgeeks.org/how-to-avoid-overfitting-in-svm/
9. https://arxiv.org/pdf/1608.06048.pdf

Contributions

- Armin: Developed the code for Tasks 1 and 2 and wrote their sections in the report. Created the data availability bar chart and world map. Searched for and incorporated the three additional datasets along with manually cleaning them. Investigated undersampling, oversampling, SMOTE, and ADASYN for class imbalance. Developed the data processing pipeline for data cleaning, feature engineering, feature selection, missing data imputation, and feature mapping (Tasks 3-4). Trained the RandomForest classifier and wrote its report components. Wrote problem statement, comparative study, model predictions, conclusions, and partially wrote class imbalance, and lessons learnt and future work on the report. Created visualization for the model predictions section. Aggregated work for submission.
- Minh Thanh Tran: Researched and implemented SMOTE for resampling, trained and fine tuned SVM classifiers and its hyperparameters with GridSearchCV, wrote reports for task 5 class imbalance, SVM model, SVM overfitting, lessons learnt and future work, conclusions and various other components. Formatted overall report. Integrated external dataset to measure air quality and pollution but scrapped at the last minute after derived features do not significantly improve model performance.
- Adam: Trained MLP Classifier and wrote its corresponding components on the report. Implemented RandomizedSearchCV for hyperparameter tuning, utilized SMOTE for

class balancing. Provided optimal tuning file for MLP model. Worked on unused visualizations. Found and cited references. Formatted references section of the report. Wrote .readme file, including libraries used. Contributed to overfitting, model building and classification sections on report. Researched advantages and disadvantages of each classification model. Edited and proofread sections of the report.