CMPT 459 - Spring 2024

Final Project Report

Illia Nasiri (301471585),
Sviatoslav Rublov (301540948)

All code was written and run in **Google Colab.** [Link]

0. Problem Statement:

Our problem lies in building a classifier for the COVID-19 dataset that will assign one of the classes to patients: “deceased”, “hospitalized”, “non-hospitalized”. The main difficulty lies in the fact that data is strongly skewed towards the label “hospitalized”.

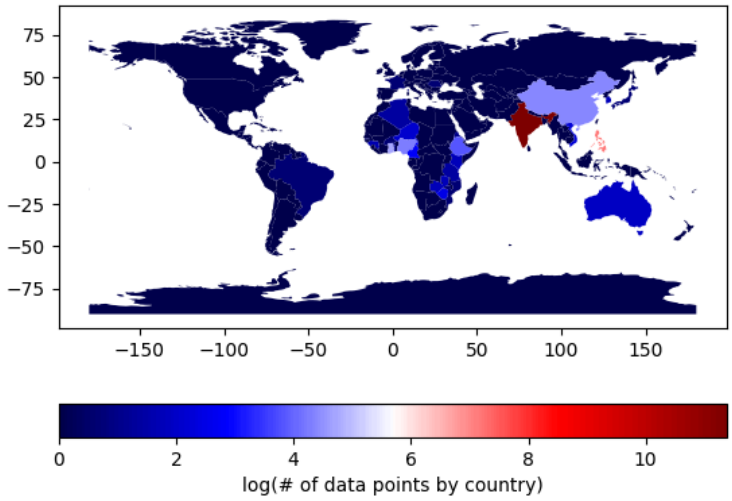
1. Data Visualization

1.0. Visualization Process: to visualize the data frequency, GeoPandas library along with its built-in *“naturalearth_lowres”* dataset containing geometry of world countries was used.

1.1. Anomalies: initially, when visualizing the data, it appeared that an overwhelming majority of data points (~98%) were collected in India. This showed itself as an artifact in the visualization where only India is colored, and rest of countries are ignored.

1.2. Solution:

To solve the problem of one data source dominating others, we took the logarithm of the counts, which makes it easier for the visualizer to color it properly.



2. Data Preprocessing:

Data preprocessing was done through a combination of steps: data combination and data cleaning.

Combination was done by doing the inner join of **Location**(left) and **Cases**(right) through *pd.merge()*. The resulting dataset was cleaned by removing unnecessary columns and keys. *“NA”* values were removed to start labeling the entries by putting labels: *“hospitalized”*, *“nonhospitalized”* and *“deceased”*. The ambiguity between some values in the initial dataset forces us to make assumptions regarding some values. For example:

```
hospitalizedList = ["discharge", "discharged", "stable","stable condition","Receiving Treatment", "Hospitalized"]
nonhospitalizedList = ["Alive", "recovered", "Recovered"]
deceasedList = ["death", "Deceased", "died"]
```

This implies that *"recovered"* is not in the hospitalized class as people may recover from the disease either at home or at a hospital. The same happens with *"Alive", "Receiving Treatment", "stable".* Thus this leaves lots of ways to permute them. Moreover, we could introduce the classifier which will help us separating these attributes into more meaningful class separation.

3. Feature Selection:

Our approach to feature selection consisted of inspecting the provided "clean" dataset as well as experimenting on the models that will be covered in more detail in **part 6.**

3.1. Selected Features: age, sex, chronic_disease_binary, Confirmed, Deaths, Recovered, Active, Incident_Rate, Case_Fatality_Ratio

3.2. Rationale Behind Removing Certain Features:

- Province: inspecting the "clean" dataset, we could see that most values were empty. Thus, we decided that it would be for the better to remove this as an attribute.
- Country: from the visualization part we knew that all data points were mainly collected in India; thus, there's not a lot of variety seen in this attribute which makes it not very informative for the trained models.
- Longitude and Latitude: We initially kept the two features thinking that they will be more informative than the country and city attributes because they're good at communicating geographical locality and the notion of "being close" within a city and a country. **But** through mere experimentation, we saw that removing these attributes has improved the performance on testing data. It increased from ≈72% to ≈79-80% on f1_macro.

4. Mapping the Features

Mapping the "**sex"** attribute was rather arbitrary and did not have any logical rationale behind it.

sex			**chronic_disease_binary**			**outcome_group**	
Original value	**Mapped to**		**Original value**	**Mapped to**		**Original value**	**Mapped to**
male	1		True	0		deceased	0
female	0		False	1		hospitalized	1

						nonhospitalized	2

5. Balancing the Data

Statement of the problem: The dataset was imbalanced predominantly consisting of training examples labeled: “hospitalized” and “non-hospitalized”, with far fewer examples of the class “deceased”. This meant the model would try to “cheat” by predicting “hospitalized” every time, not learning anything useful from the data.

class	“deceased” (0)	“hospitalized” (1)	“nonhospitalized” (2)
# of entries	997	13241	2974

Solution: we decided to **uniformly resample** the dataset so that every class occurs in the new dataset with the same exact frequency. This frequency is defined by the number of data points in the under-represented class, “deceased”, with 997 entries. Below is shown the histogram of the labels of the newly generated dataset:

class	“deceased” (0)	“hospitalized” (1)	“nonhospitalized” (2)
New # of entries	997	997	997

6.0) Models and Hyperparameter Tuning (high-level overview)

Models Used: 1. Random Forest 2. Gradient Boosting Machine 3. Neural Network 4. KNN

Normalization: having worked with the data, we have hypothesized that normalizing the data would benefit the performance of two models:

1. **Neural Network** because of how normalized data improves the convergence of the gradient descent.
2. **KNN**, because this algorithm relies on the notion of distance, and values largely differ in their range.

Note: we have tried normalizing the data for the other two models, and the performance did not change.

Tuning Strategy and Cross Validation: for all classifiers we picked the appropriate hyperparameters and trained them by using the Scikit-learn implementation of **GridSearchCV.** For larger parameter search spaces, we used a more efficient alternative **HalvingGridSearchCV.** These algorithms are manageable for a relatively small dataset, because training each model takes less or around a second. The grid

searches were scored by 2 customly defined scoring functions that compute f1_deceased_score and f1_macro_score**.** The implementation of Grid Search splits the training dataset into 5 chunks by default, **(5-fold cross validation),** meaning we train on 80% and test on 20%. This grid search checks the model's performance for every setting of the hyper-parameters across the 5 folds. It finds the best hyperparameters and retrains (refits) the model on 100% of the dataset after. The default number of 5 for cross validation seems to be reasonable because of the relatively manageable and small size of this dataset. Also, since we're working on a group using Colab, we are able to run this search across multiple machines.

6.1) In depth overview of each model

Note: *ACCURACIES FOR EACH CLASSIFIER ARE STATED IN* ***PART 8***

Model 1: Random Forest

Why use this classifier? As this is an ensemble classification method, it has the advantage of reduced variance over classifiers such as decision trees.

Best Hyperparameters:

min_samples_leaf	min_samples_split	n_estimators
10	22	19

Model 2: Gradient Boosting Classifier

Why use this classifier? It is a powerful classifier, which gives preference to some specific features to detect more complex patterns. It is possible due to the reduction in importance of such attributes which are introducing noise into the model and are of less significance.

Best Hyperparameters:

min_samples_leaf	min_samples_split
6	18

Model 3: Neural Networks

Why use this classifier? Unlike the classical approaches we have seen in class for classification, neural networks have the advantage of being universal approximators,

which allows them to learn decision boundaries of an arbitrary complexity on the predicate that they're deep or wide enough for the task. So it could be said that this method has the advantage of being more flexible.

Best Hyperparameters:

architecture	activationFn	learning_rate_init	alpha	max_iter
(12,12,4)	tanh	0.005	0.00075	2000

Model 4: KNN

Why use this classifier? It is easy to interpret, has no training phase and fits well complex decision boundaries, which will enable us to produce quite accurate and meaningful classification. Also, if it has good performance, it would mean that we can expect the classes to form somewhat close to clusters and groups.

Best Hyperparameters:

n_neighbors
26

7) Overfitting Analysis:

General factors aiming in reduction of overfitting:

- **Grid Search and Cross Validation:** combining grid search and cross validation was effective in detecting overfitting as well as preventing it. This is because we have included a large range of hyperparameters which ensures that grid search builds models of varying complexity (from simple to more complex). Then, it would pick the model with the best **average** performance during cross-validation procedure. The fact that we're considering the average performances on each of the validation folds further reduces the possibility of overfitting by reducing the likelihood of the model performing better by a mere coincidence.

Model-specific factors aiming to reduce overfitting:

- **Random Forest and Gradient Boosting Classifiers:** to reduce the possibility of overfitting in the two aforementioned classifiers, we included hyperparameters such as min-support (minimum number of training examples in leaf nodes), and *min_samples_split* in the grid search procedure. We have also limited the maximum depth of each individual estimator in the ensemble so that the goal of the ensemble is learning general patterns rather than overfitting the data.

8. Comparative Study: below is the table with the performances across 4 classifiers trained using the GridSearch process.

classifier	Average f1 deceased (averaged over 5 folds of cross validation with best hyperparameters)	Average f1_macro (averaged over 5 folds of cross validation with best hyperparameters)	Average accuracy
Random Forest	0.700450	0.791332	0.794378
Gradient Boosting Classifier	0.713196	0.796812	0.798729
Neural Net	0.703630	0.789165	0.790371
KNN	0.695033	0.781214	0.784690

Looking at this table, it could be seen that all 4 of the models seem to perform almost equally well with very slight deviations. Yet, looking just at the numbers, **the gradient boosting classifier SLIGHTLY** outperforms other classifiers across all metrics.

Conclusion:
To wrap up, the data preparation part of the project was quite linear and standard. We did not find anything unexpected. But when we got to the stage of building and working with the classifiers, we were stunned to see how significantly the *f1_deceased_score* of the neural network went up from 0.04 to 0.70 by simply normalizing the data. Also, what seemed very surprising was that all of the 4 models had almost the same performance across all the metrics and only had small variance.

Lessons Learnt and Future Work: We learn a lot as a team. First, it was a lot of hands-on experience with cleaning impure data with a lot of missing values and mislabellings. Secondly, we have acquired knowledge as to how to work with highly skewed data for classification tasks. Lastly, we have learned how to do hyper-parameter tuning better by working with Scikit-learn's GridSeachCV.

Two possible ways to extend this project further are:
1. To attempt to build a model with better performance by picking larger ranges for hyper-parameter tuning.

2. To apply clustering algorithms to the data to try and see whether there is any interesting dependence emerges between the ground truth labels and the clustering. In other words, trying to find out more about how classes are related through clustering.

3. This could be a more theoretical extension to this project. As mentioned previously, we found that the neural network performance on this dataset significantly increased after normalization; thus, it would be very interesting to apply some rigorous analysis to see why exactly the performance of the model improved so significantly after normalization.

Contributions by part:

Part	Sviatoslav	Illia
1	0%	100%
2	100%	0%
3	50%	50%
4	50%	50%
5	0%	100%
6	50%	50%
7	50%	50%
8	50%	50%
9	100%	0%