COMP 4448
Data Science Tools 2
University of Denver

Predicting Lung Disease Recovery - Classification Model Comparisons

Logan Hahn

Purpose:

This project aims to predict whether a patient will recover from a lung disease using a set of clinical and demographic attributes. By developing and applying three different classification models, the objective is to identify the model that can best assist in evaluating recovery likelihood based on patient characteristics like age, gender, smoking status, lung capacity, treatment history, etc.

Significance:

Predicting recovery outcomes in lung disease patients can provide a massive value to both the patients and the healthcare providers. Hospitals can allocate resources more efficiently, personalize treatments, and intervene with treatments early by identifying high-risk patients based on these attributes. This would lead to increased patient recovery and higher treatment success rates by healthcare providers.

Research Question:

Which algorithm, Logistic Regression, Random Forest, or XGBoost, would be the most effective in predicting whether or not a patient with lung disease will recover based on their demographic, lifestyle, and clinical characteristics?

Dataset:

The dataset used in this project was sourced from the online public dataset source, Kaggle:

Link: https://www.kaggle.com/datasets/samikshadalvi/lungs-diseases-dataset

It contains medical and demographic information on individuals diagnosed with a lung disease. The dataset itself has 5200 rows and 8 columns, that's 5200 patients and 8 features including the target variable, "Recovered", which is a binary categorical variable indicating whether a patient recovered or not. (string values: Yes & No).

All input features with their types and descriptions are provided in the table below:

Feature	Type	Description
Age	Numerical	Age of patient in years
Gender	Categorical	Male / Female
Smoking Status	Categorical	Whether patient is a smoker, Yes / No
Lung Capacity	Numerical	Patient lung capacity measured in Liters
Disease Type	Categorical	Type of diagnosed lung disease
Treatment Type	Categorical	Type of medical treatment received
Hospital Visits	Numerical	Number of times patient visited the hospital

Data Cleaning:

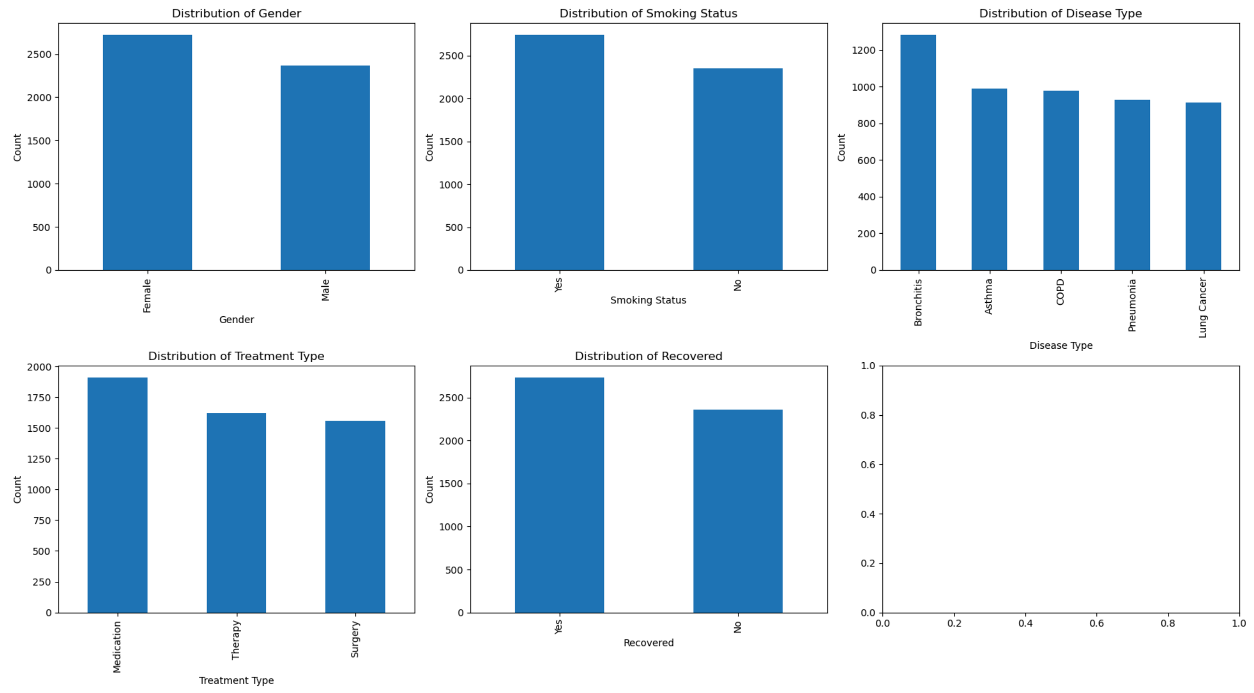
While inspecting the dataset I found there to be 300 missing numerical and categorical values from the features. I looked at the proportion of missing values per column and relationship between those missing values and other variables. I couldn't see any clear patterns suggesting they were "Missing Completely At Random". In hope to retain as much data as possible I imputed the missing values with mode for the categorical variables and attempted to simply used median for the numerical values. This resulted in distorted distributions spotted in my EDA and hindered performance in modeling. I then tried using the KNNImputer from the sklearn module and found slightly better results. I also found 91 duplicate rows that were then removed.

Exploratory Data Analysis:

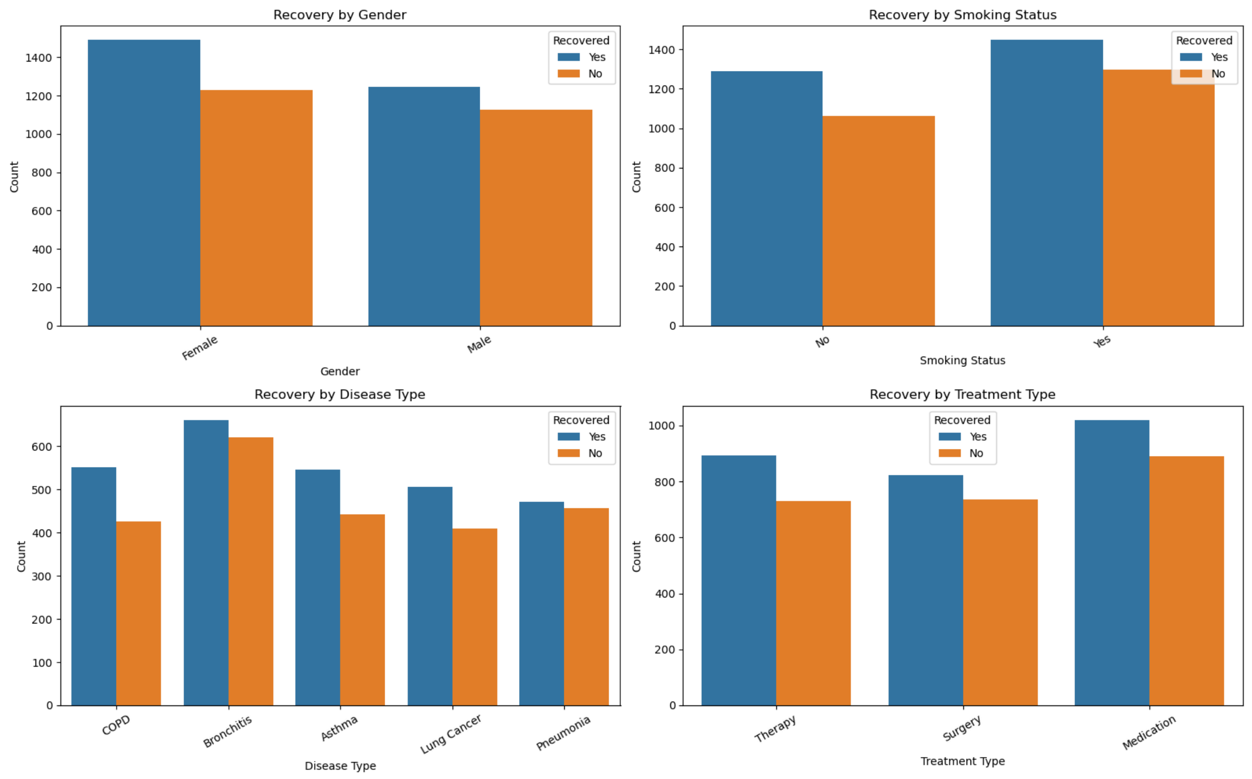
To get a better understanding of the patterns in my data I conducted an initial analysis of the categorical and numerical variables to look at distributions and variable's relationships to the target variable.

Categorical Variables

To visualize the distributions of the categorical variables I created bar graphs for each feature. (*See plots below*)



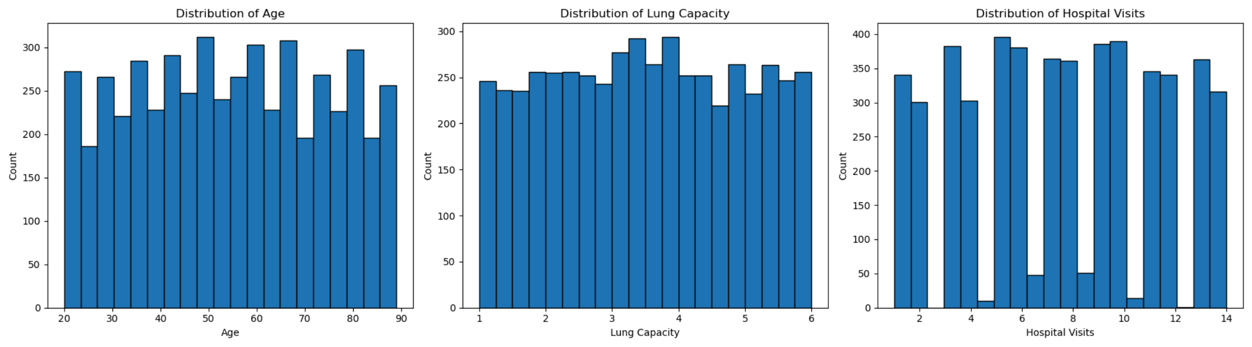
As you can see from the graphs here, all categories appear to be fairly distributed and well represented in the data. No single category in any feature is a dominant class. This includes the target variable, "Recovered", shown in the bottom middle graph and is also well balanced, (Yes=52%, No=48%). I also plotted grouped bar charts segmented by "Recovered" status. (*see plots below*)



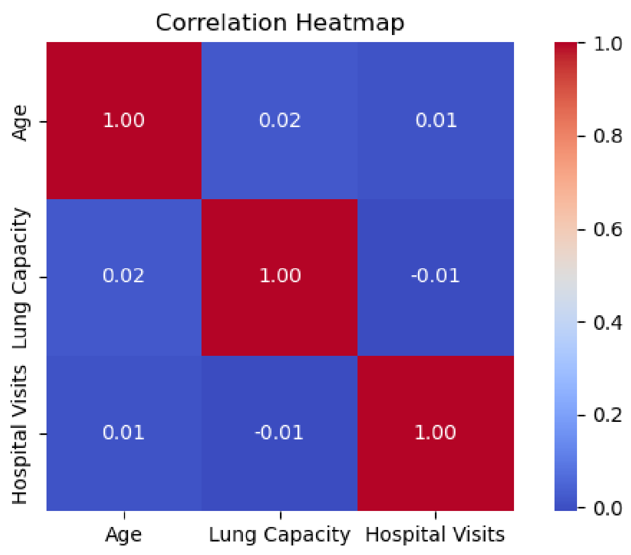
Visually, there doesn't appear to be any categorical features that exhibit a strong association with the target variable. To validate this I also performed a chi-squared test that revealed only one of these features had a significant p-value. The feature, "Disease_Type" had a p-value of 0.04, which compared to any other feature had by far the highest association to the target variable. This shows that while only one categorical feature could be a decisive predictor individually, their interactions may still yield predictive power.

Numerical Variables

I plotted histograms to look at the distributions of the three numerical variables. (*see plots below*)



Again, these features appear to be distributed equally with no clear dominant classes. Hospital visits show the most variability between groups but no major differences.



I also plotted a correlation heatmap for the numerical variables to look at relationships between these variables. (*see plot to left*) The heatmap revealed very weak linear relationships between these features with all coefficients close to zero. In terms of my modeling, this is favorable since it suggests multicollinearity won't be an issue and all variables could be included in modeling without requiring dimensionality reduction.

Following my EDA, I've found that most variables, numerical or categorical, lack a significant influence on the target variable. However, that doesn't mean recovery outcomes may be influenced by complex inter-variable interactions. These insights will be useful for my next steps and modeling.

Data Pre-processing & Splitting:

Before training the models the data required some pre-processing, these are the steps I took to ensure compatibility and optimal performance.

Target Variable Encoding

The target variable, "Recovered", originally had the string values "Yes" and "No" to indicate whether a patient recovered from their lung disease or not. For my binary classification task I remapped these values to 1 and 0 accordingly. ("Yes" to 1, and "No" to 0).

Feature Grouping and Transformation

First, I grouped the features into categorical and numerical subsets. I then applied OneHotEncoder to the categorical variables to transform each unique value into a binary feature column. I also used drop='first' just to further avoid potential multicollinearity even though my EDA shows it shouldn't be a problem. I applied StandardScaler to the numerical variables to ensure numerical values are uniform on a similar scale. Standardized input features can lead to faster convergence which is important to gradient based optimization used in logistic regression/XGBoost. I attempted the modeling without scaling the numerical variables and using different methods of scaling, (MinMaxScaler, RobustScaler, etc.) but did not see any better performances in the models.

Train/Test Splitting

The dataset was split 80% into the training set and 20% into the split set. Given that I am working with about 4800 patient records following data cleaning, I thought that keeping the training set at a higher 80% would allow for better training in my models.

<u>Model Building:</u>

To answer my research question, I developed three supervised classification models, (Logistic Regression, Random Forest, XGBoost) using predefined models from scikit-learn and XGBoost. These are linear, ensemble, and boosting based models to allow for comparisons with a variety of different algorithms. Each model was trained on the processed training set, evaluated with cross validation and tested on the test set.

Logistic Regression

Performance:

- Cross-validation accuracy: 0.5169 (std: 0.0103)
- Training accuracy: 54%
- Test accuracy: 57%

Test Classification Report:

- Class 0 (Did not recover): precision = 0.56, recall = 0.13, F1 = 0.21
- Class 1 (Recovered): precision = 0.57, recall = 0.92, F1 = 0.70

Interpretation:

Unfortunately, the Logistic Regression performed only slightly better than random guessing with a cv accuracy of 52%. It's recall for class 1 was very high meaning it was great at identifying people have recovered correctly. But since class 0 recall was so poor it means the model assumed the vast majority of patients would recover even when they don't. This all suggests the data's relationships may be non-linear and a linear model like logistic regression couldn't capture them well.

Random Forest

Performance:

- Cross-validation accuracy: 0.5174 (std: 0.0163)
- Training accuracy: 100%
- Test accuracy: 51%

Test Classification Report:

- Class 0 (Did not recover): precision = 0.45, recall = 0.41, F1 = 0.43
- Class 1 (Recovered): precision = 0.56, recall = 0.59, F1 = 0.58

Interpretation:

The random forest model had perfect training accuracy, meaning it memorized the training data completely and failed to generalize to new data which is definite overfitting. This model also was only able to predict slightly above random guessing with an accuracy of 51%. Tuning tree depth and number of estimators in the next step may allow it to generalize better.

XGBoost

Performance:

- Cross-validation accuracy: 0.5012 (std: 0.0092)
- Training accuracy: 93%
- Test accuracy: 53%

Test Classification Report:

- Class 0 (Did not recover): precision = 0.46, recall = 0.41, F1 = 0.43
- Class 1 (Recovered): precision = 0.57, recall = 0.62, F1 = 0.59

Interpretation:

XGBoost had a high training accuracy of 93% and a much lower test accuracy of 53%. While this looks like overfitting, it had the most balanced results across both classes, outperforming other models in terms of F1-score. While accuracy was still very poor, barely above random guessing again, the fact that XGBoost had the best balance of performance across both recovery and non-recovery classes would make it the best performing model out of the three pre-tuning.

<u>Optimization & Hyperparameter Tuning:</u>

In order to improve the performance of each model I aimed to find the best possible hyperparameter configurations using GridSearchCV with 5 fold cross validation. I performed this a number of times, refining the hyperparameters each time to find the best possible performance. Since accuracy between each model was roughly the same above random guessing, I used F1 score as the scoring metric so precision and recall are considered.

Logistic Regression

Performance Summary:

- Training accuracy: 52.0%
- Test accuracy: 52.2%
- Training precision = 55.3%, recall = 51.8%, F1 = 53.5%
- Test precision = 57.5%, recall = 54.3%, F1 = 55.8%

Interpretation:

After tuning I was able to slightly improve the model balance between precision and recall, especially for the minority class (Not Recovered). Despite these improvements, I think the performance was still constrained by the linearity of the model if there are potential nonlinear interactions that weren’t captured.

Random Forest

Performance Summary:

- Training accuracy: 99.4%
- Test accuracy: 54.1%
- Training precision = 99.2%, recall = 99.8%, F1 = 99.5%

- Test precision = 57.7%, recall = 65.4%, F1 = 61.3%

Interpretation:

Again, my tuning only marginally improved performance. Recall for the target variable improved a bit, indicating better coverage of true positives. The biggest issue is that training accuracy remained at nearly 100% so tuning was unable to mitigate overfitting by any measure. Despite limiting tree depth, adjusting feature splits, and finding the best number of estimators, I believe this model had too much capacity allowing it to learn noise and maybe specific relationships in the training set that didn't generalize to the test set. The boost in test F1-score (to 61.3%) shows that tuning helped a bit, but not enough to overcome it's inclination to overfit.

XGBoost

Performance Summary:

- Training accuracy: 61.98%
- Test accuracy: 54.37%
- Training precision = 60.0%, recall = 85.8%, F1 = 70.6%
- Test precision = 56.4%, recall = 78.8%, F1 = 65.8%

Interpretation:

XGBoost had a relatively high training F1 score of 70.6% and a decent test performance of 65.8% indicating a stronger fit than the other two models without severe overfitting. The test recall of 78.8% shows it's effective at identifying patients who are more likely to recover, which would definitely be valuable in clinical prediction tasks. The fact that again, test accuracy is barely above random guessing shows that more than likely this dataset just lacks predictive features for the purpose of predicting patient recovery.

<u>Conclusion</u>:

After thorough testing and tuning of the three classification models, I've determined that XGBoost showed the best overall performance. It consistently had the best balance between training and test set performance especially regarding recall and F1 for the recovered class. While failing to achieve a decent accuracy alongside the other two models, focusing on the F1 score would very much be useful in health-related classification tasks such as this. Paying attention to false positives and false negatives is very important in healthcare settings where misclassifications could be fatal.

Regardless, the failure of these three supervised models for this classification task is more than likely due to the dataset being used. For the task of predicting lung disease patient recovery, the features in this data may just not be great predictors. They don’t capture the clinical severity or biological complexity of an individual patient's condition. The data also only captures a patient's data in that instance in time, things like temporal focused data or progressional data could provide models more context to make accurate predictions. I could tune the models as much as I’d like but the one thing that would make the best positive difference is to enrich the data. That could be including more patient-specific health data, using time-series data or even redefining the target variable to be multiclass to represent different levels/stages of recovery would likely give the models the data they need.

So, while XGBoost did outperform the other two models and has great, beneficial applications for healthcare, more effective research data could turn these models into powerful tools to support medical decision making.