Predicting Diseases from Symptoms: A Multiclass Classification Approach

STA 141C Final Project: Allison Peng, Jasper Dong, Eric Sun, Wilson Zhou

Introduction:

According to a report in September 2023 from the Census Bureau, within the United States, 26 million people do not have immediate access to a health facility or cannot afford health insurance (Peterson Foundation, 2023). As an affordable solution, machine learning within the health field is increasing in popularity as prediction technology continues to improve. The amount of healthcare data available is increasing and provides a foundation for an accurate prediction model. To build a prediction model, we are planning to use a dataset from Kaggle, called Disease Prediction using Machine Learning.

Kaggle provided a downloadable dataset that contains 132 parameters and a singular response column. **Figure 2** provides a summary of the variable and observation counts. Each parameter represents a symptom (example: itching, rash, cough, etc.) and there are 42 possible diseases to predict from the given symptoms. Each parameter is binary-coded to determine if the patient has the symptom or not. The dataset contains 4692 observations.

Main Question: Can we construct a model that can forecast the disease associated with a patient's symptoms?

Exploratory Data Analysis pt 1: Visualization

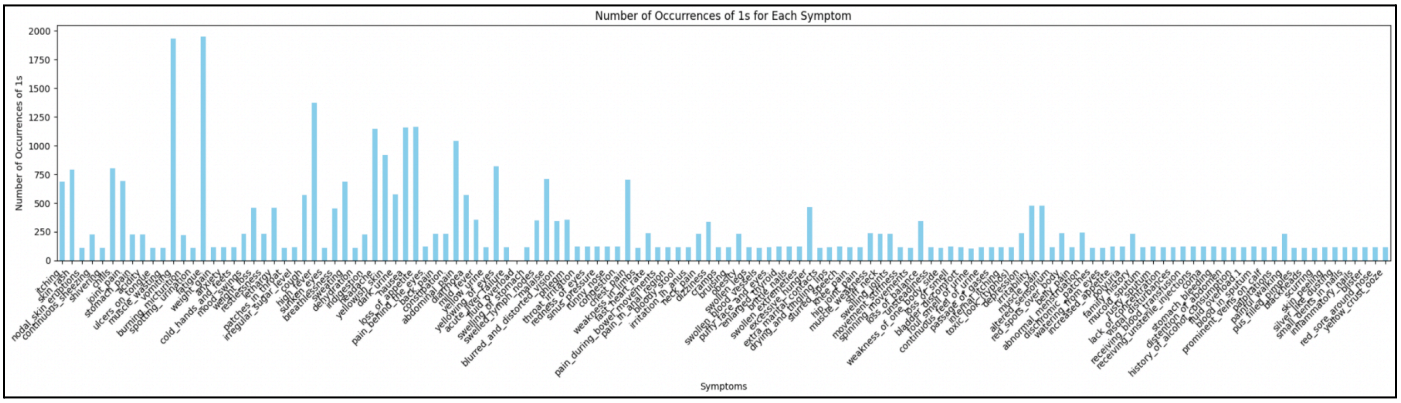

Figure 1: Histogram of occurrences of 1 for each symptom

Number of Symptoms/Features	132
Number of Diseases	42
Number of Observations	4692

Figure 2: Table with counts of variables/observations

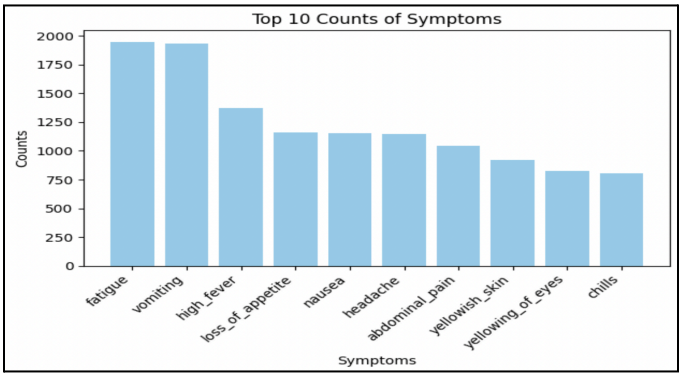

Figure 3: Top 10 Symptom Counts

Looking at **Figure 1**, we see that there is a variety of counts for each symptom. There are clearly certain symptoms that are more common among patients. More specifically, looking at **Figure 3**, the top 10 symptoms among patients were fatigue, vomiting, high fever, loss of appetite, etc. Looking at **Figure 4**, the distribution of the number of diseases among patients is uniform, with the same frequency for each disease. This information allows us to move forward with using a supervised machine learning method because we have a large dataset with sufficient information to train the model on.

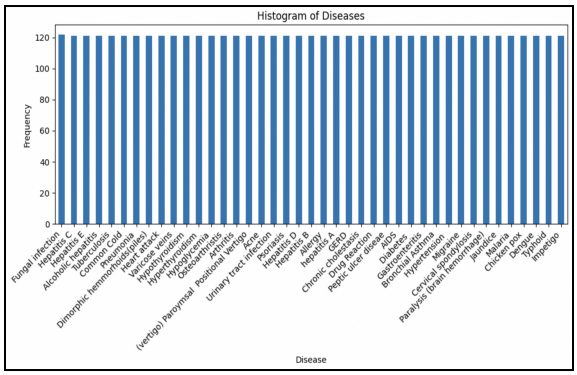

Figure 4: Counts of Diseases

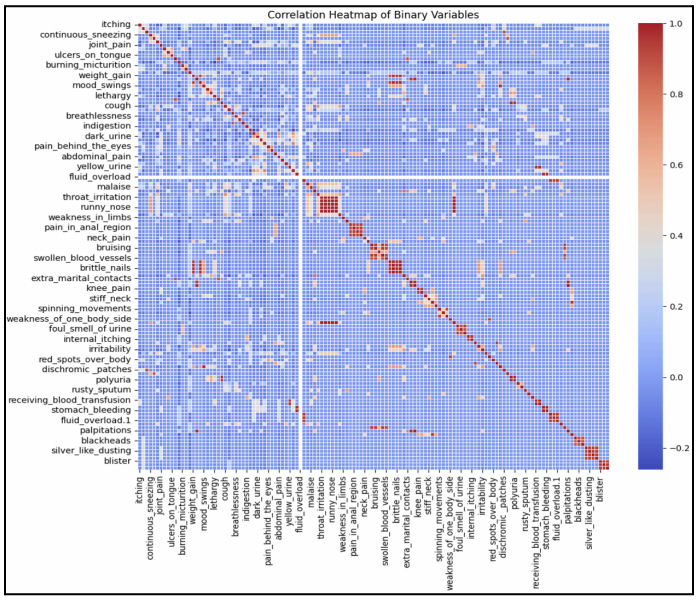

Figure 5: Heatmap showing the correlation matrix of all symptoms

Figure 5 shows the correlation matrix visualized with a heatmap to detect any multicollinearity among the symptoms data. Multicollinearity occurs when there is association between the independent variable or dependent variable. With several predictor variables, it is difficult to see the specific correlations, however, we see that a majority of the predictor variables are not correlated with each other. We looked at the predictor variables that are correlated with each other and removed 20 highly correlated variables such as redness_of_eyes, throat_irritation, sinus_pressure, loss_of_smell, congestion, etc. By removing these predictor variables, we can decrease the complexity of the model and reduce the computational power needed for our model.

Exploratory Data Analysis pt 2: Clustering

We may also perform clustering to see how similar the symptoms are to one another. To perform clustering, we decided to use the count for each symptom as a metric of how similar symptoms are to one another. We perform hierarchical clustering so that we could create dendrograms based on the Euclidean distance between symptom counts, where the heights of each fusion can help indicate similarities/dissimilarities between symptoms.

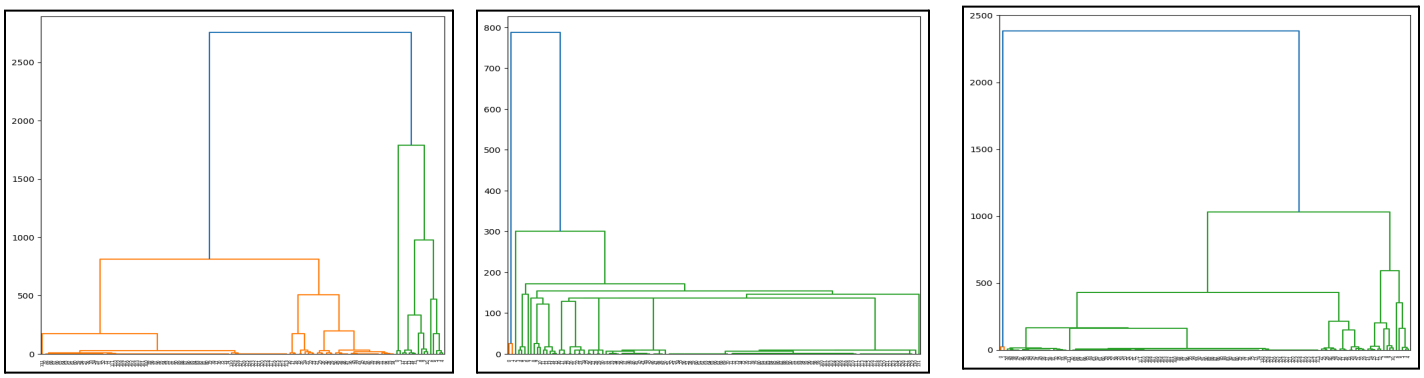

Figure 6: Dendrograms of Complete (Left), Single (Middle), and Average (Right) linkage methods.

Figure 6 shows three different dendrograms based on three different linkage methods: Complete, Single, and Average linkage. From the dendrograms, we can see that there are a few symptoms that may be similar to one another, but dissimilar to the rest of the symptoms. Especially in the dendrograms made using Single and Average linkage, those symptoms don't fuse with the rest of the symptoms until the final cluster. The dendrograms identify two or three clusters in which we can group symptoms together.

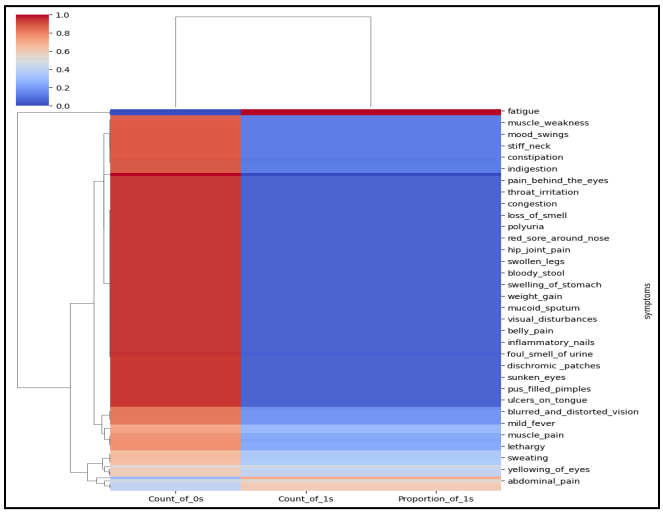

Figure 7: Cluster heatmap of symptoms

Figure 7 shows a hierarchical cluster heatmap of the symptoms in which we can easily identify the symptoms that are similar/dissimilar to one another. We can see that the heatmap identifies two main clusters. In particular, fatigue, lethargy, yellowing_of_eyes, and abdominal_pain all form one cluster, while the rest of the symptoms all form another cluster.

From our clustering results, we can broadly group symptoms into about two main clusters. Symptoms that are similar to one another would be more likely to appear together, which can help in medical applications, such as symptom assessment in patients.

Methodology:

After exploratory data analysis, we have a better understanding of the underlying data structure. We know that there is sufficient symptom and disease data to build an effective multiclass classification model. **Figure 8** splits our methodology into three steps: data preprocessing, random forest model, and error calculation. We will first use the information found from EDA to preprocess the data to allow for the most effective model building.

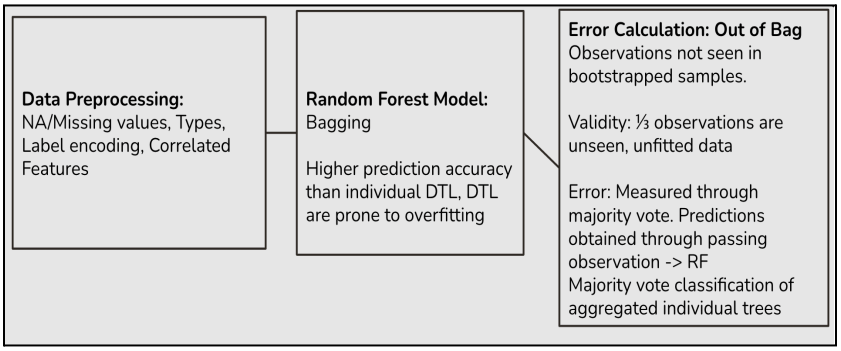

Figure 8: Methodology to answer central question

a) Data Preprocessing: NA/Missing Values, Types, Label Encoding

We first removed the NA values and any missing values in the dataset. Then, since the disease column is in text format, we label encoded, or convert each unique disease to an integer, the data to allow the model to classify the symptoms. We also ensured that there was no class imbalance within the split. Class imbalance occurs when there is an imbalance of unique classes between the test and training dataset. To ensure that both the training and test data have all 42 diseases, we used a stratified method during the split.

b) Data Preprocessing: Train/Test Split

As RFs are a supervised learning method (predict given target labels), we want to split the dataset into training and test sets. The training set is used to fit and discover the relationships in the data, whereas the test set is used for evaluation and accuracy. In our

initial proposal, we downloaded the test.csv associated with this Kaggle dataset and based our predictions based on that file. However, test.csv only contained 50 rows of unseen observations compared to our response variable (Disease) with 42 outcomes, rendering the evaluation invalid due to the small number of observations. Instead, we opted for a better 75/25 split, corresponding to 3690 and 1230 observations in each class, respectively.

c) Model Building: Random Forest Classification

Selecting an appropriate model was done after conducting the preliminary steps of data cleaning and EDA. We ensured that the response variable, which was in text format, was properly encoded into values interpretable by a machine. For predicting a multiclass response variable given a generous amount of binary features, we decided to select *Random Forests (RF)* as our model of choice. Compared to individual decision trees (DT) and pure bagging, many research studies have proven the credibility of Random Forests for optimal multiclass classification purposes. In order to understand why this is the case, we give a brief summary and overview of the strengths of RFs.

	Strengths	**Weaknesses**
Random Forest	- Low Variance due to ensemble methodology - Less prone to Overfitting - Decorrelated trees	- Difficult to interpret - Computational complexity
Decision Trees	- Simple Interpretability - Simpler understanding of Gini Impurity split	- Heavy Overfitting - Sensitive to small changes in training set - High Variance

Figure 9: Cost-Benefit Model Analysis

The goal of a model is to generalize well on unseen data. This is done through tuning the bias/variance tradeoff such that there is no heavy underfitting or overfitting. Random Forests compared to DTs are a stronger method for discovering the true, underlying relationships between the features and response.

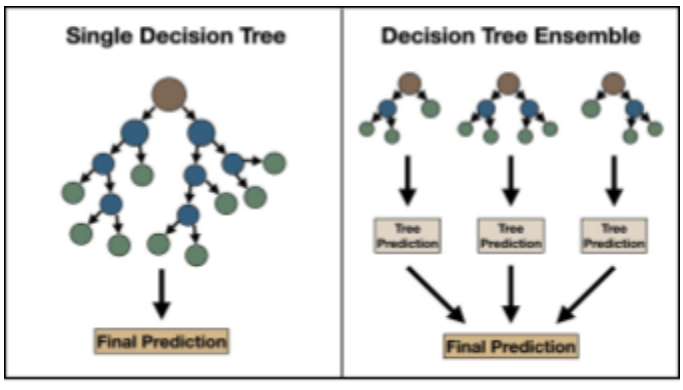

Figure 10: Visual comparison between decision tree and random forest

We initialized three instances of the Random Forest classifier. The difference among the models can be attributed to the various number of features to randomly subsample on for splitting the decision trees. By definition, a Random Forest has its max_features parameter set to *sqrt* (square root), which takes the square root of the number of existing features in the dataset, effectively splitting the tree by that new value of features. The other two instances had parameters set to log base 2 and none.

Main Results:

There are two robust methods of evaluating a random forest's model performance: Classification Matrix and Out of Bag Error. We will look at the results of our model and decide if we need to do further hyperparameter tuning to adjust the model's performance.

Classification Matrix:

Figure 11 displays the model predictions against the ground truth observations in a matrix. Metrics of evaluation include precision, recall, and F1 score. The resulting performance of all three random forest models using max_features = sqrt, log2, none all resulted in a 100% accuracy measurement for all metrics.

accuracy	1.0	1.0	1.0
macro avg	1.0	1.0	1.0
weighted avg	1.0	1.0	1.0

Figure 11: Classification matrix of random forest model: Columns are precision, recall and F1 score

Out-Of-Bag Error (OOB):

Utilizes unseen observations in *n* number of Bootstrapped Decision Trees due to random sampling process. On average, ⅓ of the observations in the original data are not selected in fitting the individual decision trees. Error is calculated through majority vote classification through passing in an observation's features into aggregated individual trees that each return a class.

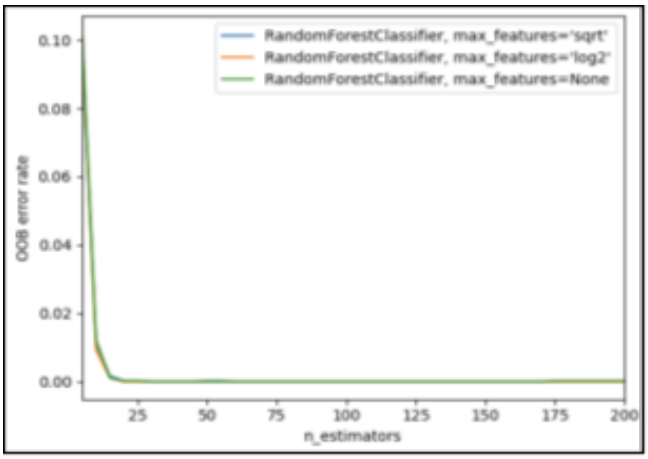

Figure 12: OOB error rate vs number of estimators

The OOB error rate on the Y-axis represents the mean error of each RF instance in the legend, given the number of decision trees used. It can be observed that the number of features to randomly split the nodes on does not have much significance on the error rate. The plot is used to analyze the number of trees at which the OOB error is stabilized and provides a clear graphical representation of a suitable selection of the n_estimators hyperparameter. In this case, it seems that a suitable number of trees could be around 20.

*Note - These results are calculated before *hyperparameter tuning*. In the next section, we will run an algorithm to figure out the best parameters for the random forest classification model.

Hyperparameter Tuning

Our results from the initial random forest model led to a 100% accuracy, which is typically unlikely for a classification model. As a result, we decided to tune the model to see if the random forest parameters could be improved. If the model still remains at 100% accuracy, likely reasons would be that the model is very effective for the data, or the provided data is not reliable. After further inspection of the random forest classification model, the hyperparameters we decided to tune are:

- *n_estimators, max_depth, max_features, min_samples_split, min_samples_leaf, bootstrap*

We used the function RandomizedSearchCV, which takes a hyperparameter distribution and randomly selects parameters to test the model's performance with each random combination. The function uses cross validation to find the best combination of parameters and returns the parameters that give the least error for the model. Before using tuning, we used the default parameters determined by the function. After tuning, we got the following results:

- *n_estimators: 136, max_depth: 49, max_features: sqrt, min_samples_split: 6, min_samples_leaf: 2, bootstrap: True*

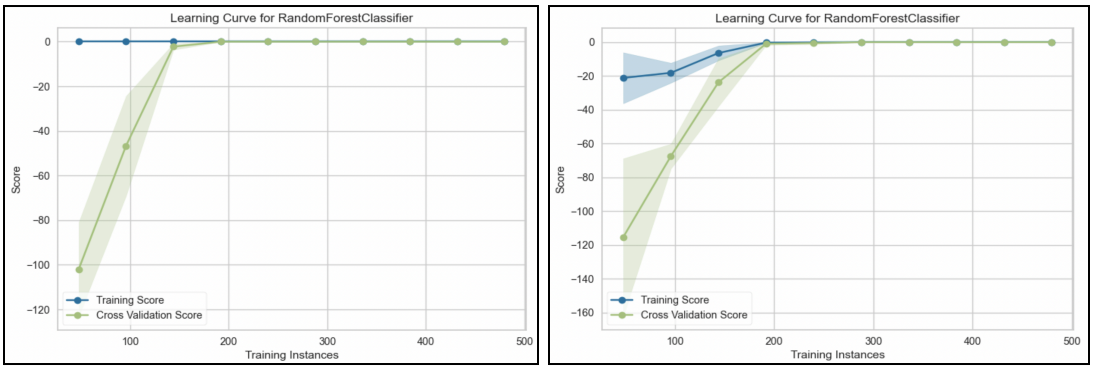

Figure 13: Learning curve for pre-hyperparameter tuning (left) and post-hyperparameter (right) tuning with negative mean squared error

Figure 13 (left and right) shows that we initially had overfitting when using the random forest classifier. The training data had 100% accuracy and the testing data had high error. However, the pre-hyperparameter tuning model shows unusual behavior for a random forest classifier because the training data received 0 error with very few instances. Post-hyperparameter tuning demonstrates an improved performance of the model as we can see the error gradually decreasing as the instances increase. Both models lead to a 100% accuracy, however the hyperparameter tuning allowed for improved model performance.

Discussion/Outlook:

Our goal was to develop a machine learning method to classify given medical symptoms into 42 possible diseases. After data analysis and visualization, clustering techniques, model selection and tuning, we successfully developed a random forest classifier that predicts given symptom data with a high accuracy. However, we ran into issues with the final accuracy of the model. With the initial fitting of the random forest model using default parameters, we ended with an extremely high accuracy of 100%. This seemed unlikely for any machine learning model, thus we looked into the possible causes. We first inspected the initial train/test split of the data. The original data from Kaggle was pre-split into only 50 observations for the test data, which initially caused overfitting of the model due to class imbalance. Thus we changed the split to be 75/25. This did not change the final accuracy of 100%. We then looked into the random forest model and performed hyperparameter tuning to improve the model. The results allowed us to improve the model performance, with a more reasonable learning curve. Other possible methods of improving the model is to obtain more observations to improve the model training. Future work could also involve applying other tree-based models such as using XGBoost which uses a gradient boosted trees algorithm to make accurate and simple predictions.

Conclusion:

Preliminary exploration of the data set aided in visualizing and understanding the structure of the dataset. In particular, we were able to identify correlated predictor variables that aided us in fitting our random forest model, and clustering helped in indicating similarities and dissimilarities between symptoms. Our fitted random forest model performed exceptionally well, with high accuracy and low error. Hyperparameter tuning of our model seemed to further improve the performance of our model, reducing any bias that may have been present in the model before tuning. We successfully were able to answer our main question and fit a model to predict diseases given symptoms. The use of random forest for multiclass classification has large applications within the medical field and can be implemented as a screening process for patients to receive an initial diagnosis.