Methods and Analysis

Link to GitHub: https://github.com/brichards21/PHP2550-Final-Project

Recall that the aim of this study is to assess if we can predict future isolates' performance against certain antibiotics where 'performance' is a binary discretion of either susceptibility or resistance. In other words, we want to assess if we learn from past isolates to develop a system to recommend antibiotics to new isolates. We are also generally interested in the kinds of patterns that we can extract from our current data about behaviors of drug susceptibility and resistance including how responses to these antibiotics shift over time.

Data Pre-processing

Separating the Data

This analysis focuses on isolate data collected from the Salmonella enterica, E.coli and Shigella, and Campylobacter jejuni species in the National Center for Biotechnology Information's (NCBI) Pathogen database. Initially, we had decided to evaluate isolates from these three bacteria within joint modelling and analysis procedures. However, through our data exploration process, we found that our Salmonella isolates had retained two additional pieces of information compared to the E.coli and Campylobacter isolates. This information are antigen formula, which gives the detected presence of a specific viral antigen indicating viral infection, and serovar/serotype which gives the distinct variation within a certain bacteria. Additionally, while isolate data for Salmonella were retained from 2002-2021, E.coli and Campylobacter records did not begin presenting themselves in our data until much later (2009 for E.coli and Shigella, and 2016 for Campylobacter). For these reasons, we split our dataset into two groups. One set for Salmonella and another set for the combination of Campylobacter and E.coli isolates. Notably, this separation was also convenient for the balancing of the different sample sizes within our datasets. Overall, we retained 8,672 observations on Salmonella, 3,880 on E.coli, and 3,497 on Campylobacter. By separating the data into one set for Salmonella and another set for E.coli and Campylobacter, we avoid a joint final model being trained heavily on the Salmonella group due to the relatively larger sample size.

Reformatting the Most Common Antibioitics

The full scope of our data contains isolates that are resistant and/or susceptible to 79 unique antibiotics, 76 of which these isolates have shown resistance to, and 62 of which these isolates have shown susceptibility to. Evidently, whereas some isolates have shown resistance to a certain drug, other isolates have shown susceptibility to the same drug. The idea is to create a system or function that will take genetic and baseline covariate information about isolates as input values and then within a subset of the most common drugs we've found the isolates to show the most response to, recommend the one that we've predicted with the *most* evidence that new isolate is susceptible to, along with the level of confidence that we have in this recommendation (i.e. the probability that we obtain from our classification model). Therefore, since this is a binary classification problem, on the data that we'll use to develop our models, we'll code '1' as drug susceptibility, '0' as drug resistance, and NA if we do not retain any information about that particular drug on a given isolate. The subset of most common antibiotics that we want to consider in our models fulfill one of two criteria: 1) represents at least 10% of all antibiotics recorded within a given bacteria's drug response (resistance/susceptible) group, or 2) in the case that no drug represents at least 10% of all drugs reported for susceptibility or resistance within a given bacteria, be within the top two most common drugs that isolates of a given bacteria have shown susceptibility or resistance to. **Table 1** shows the most common antibiotics that isolates within each bacteria showed resistance and susceptibility to. We combine the unique sets of these antibiotics as the sets to be assessed in the recommendation procedure.

Reformatting the Most Common Antimicrobial Resistant Genes

We also reformatted the prevalence of the antimicrobial resistant genes found in each of our isolates. There are 499 unique resistant genes found in the isolates within our data. In order to include some level of gene resistant presence as predictors in our model, we chose to include binary variables for the most common resistant genes observed. These variables took a value of 1 if found within a given isolate, and a value of 0 is not found within a given isolate. The genes that we decided to include as predictors in each of the two separate models are displayed in **Table 2**. We decided to retain the genes with over 2% prevalence in the

Table 1: Antibiotics Considered in Modeling

Species	Most Common Drugs: Susceptible	Most Common Drugs: Resistant	Unique Drugs Included in Model
Salmonella enterica	Tetracycline (22%) Streptomycin (16%) Sulfisoxazole (12%) Ampicillin (11%)	Tetracycline (22%) Streptomycin (16%) Sulfisoxazole (12%) Ampicillin (11%)	Tetracycline Streptomycin Sulfisoxazole Ampicillin
E.coli and Shigella	Meropenem (8%) Ciprofloxacin (7%)	Tetracycline (14%) Ampicillin (11%)	Meropenem Ciprofloxacin Tetracycline Ampicillin Gentamicin Erythromycin Florfenicol Azithromycin Clindamycin Nalidixic acid
Campylobacter jejuni	Gentamicin (13%) Erythromycin (13%) Florfenicol (12%) Azithromycin (12%) Clindamycin (12%)	Tetracycline (48%) Ciprofloxacin (21%) Nalidixic acid (20%)	

Salmonella isolates, and the genes with over 3% prevalence in either the E.coli isolates or the Campylobacter isolates. A higher prevalence threshold of 3% was chosen for E.coli and Campylobacter as to not overpopulate the joint E.coli and Campylobacter model with these genetic predictors.

Table 2: Antimicrobial Resistant Genes Considered in Modeling

Species	Most Common Antimicrobial Resistant Genes	Unique Genes Included in Model
Salmonella enterica	mdsA (22%) mdsB (21%) tet(A) (6%) aph(3")-Ib (6%) aph(6)-Id (6%) tet(B) (4%) sul2 (4%) sul1 (3%) blaTEM-1 (3%) aadA1 (3%) fosA7 (3%) blaCMY-2 (2%)	mdsA mdsB tet(A) aph(3")-Ib aph(6)-Id tet(B) sul2 sul1 blaTEM-1 aadA1 fosA7 blaCMY-2
E.coli and Shigella	blaEC (13%) acrF (12%) mdtM (11%) tet(A) (4%) aph(3")-Ib (3%) sul2 (3%) aph(6)-Id (3%)	blaEC acrF mdtM tet(A) aph(3")-Ib sul2 aph(6)-Id "tet(O) blaOXA-193 50S_L22_A103V gyrA_T86I aph(3')-IIIa blaOXA
Campylobacter jejuni	tet(O) (24%) blaOXA-193 (17%) 50S_L22_A103V (12%) gyrA_T86I (10%) aph(3')-IIIa (7%) blaOXA (4%)	

Handling Missing Data: Multiple Imputation

Finally, in order to move forward with implementing our methods, we needed a way to handle the missingness in our data. Upon thorough investigation of the data which led us to believe that we were dealing with missingness at random (MAR), we decided to impute these missing values using Multiple Imputation. To do so, we utilized the `mice` function from the `mice` package in R (van Burren, Groothuis-Oudshoorn, 2011).

Multivariate Imputation via Chained Equations (MICE) imputes on a variable by variable basis, meaning that a new imputation model is specified with each new variable containing missingness. Linear regression/predictive mean matching was used to predict missing values for continuous variables, and logistic regression was used to predict missing values for categorical variables.

As a rule of thumb, we omitted the variables with over 80% missingness from our dataset first (which led to the omission of one of the self-made variables from exploratory data analysis assessing the measure

of similarity between the antibiotics that isolates of the same SNP cluster show resistance to), and then imputed the remaining variables. For each of our 2 datasets of interest, the Salmonella data and E.coli and Campylobacter data, we imputed five datasets with 5 iterations.

Random Forest Classification

For the purpose of classification, we decided on using the Random Forest algorithm. In order to implement this process in R, we will use the `randomForest` function from the `randomForest` package (Liaw and Wiener, 2002). The Random Forest model's building components are decision trees. Decision trees are a sort of supervised learning in which input is continuously split based on certain parameters. Random Forest fits various trees with different bootstrap samples and splits with random feature subsets on each node. Predictions are made using the average of the results from each tree. To partition the data, the decision tree and random forest employ information gain and impurity to ensure that non-informative features are not chosen. To optimize the decision tree, a splitting criterion must be considered; it is critical to decide on splitting options to minimize errors caused by a few observations in a node, which would eventually lead to a lack of statistical significance. However, overfitting can occur when the tree is too deep, and underfitting can happen when the minimal number of samples to split is too small. In order to combat these potential issues in R, we will consider tuning the following parameters for the algorithm on a training dataset: ntree indicating the number of trees, nodesize indicating the minimum size of the terminal node, and mtry indicating the number of randomly sampled variables in each split. Different combinations of these parameters will be explored in a grid search process and using 10-fold cross validation, will be chosen based on their resulting out-of-bag errors, a metric used to measure the prediction error of random forests on data points not utilized in a given bootstrap sample (akin to an internal testing set). More specifically, we'll choose the set of parameters that produce the lowest out of bag error. Performance of our trained models will be primarily assessed on a withheld testing set.

Additionally, to increase the accuracy of prediction and decrease the computational time, we decided to consider variable selection. We'll use the feature selection wrapper algorithm, Boruta, to potentially reduce the dimensionality of our data (`Boruta` function from the `Boruta` package in R) (Kursa and Rudnicki, 2010). In use, the Boruta algorithm first adds randomness to the dataset of interest by creating shuffled copies of all of the features up for consideration. These new shuffled copies are called shadow features. Next, the function applies the random forest on the data and evaluates the importance of each of the features in the classification process. With each iteration of the classifier, the Boruta algorithm checks if the un-shuffled, original feature has a higher importance than the most important of its shuffled shadow features. If an original covariate is not deemed more important to classification than the best of that covariate's associated shadow features, then it is deemed unimportant and removed as a variable worth keeping in the modeling process. The Boruta algorithm stops when either all features are either confirmed or denied as being important, or alternatively when the maximum number of a specified number of random forest runs is reached.

Visualizations and Tables

Firstly, we will plot **out of bag error vs. number of trees** both overall and by class (0 vs 1) to assess the prediction errors of our random forest tuning process. As we are assessing the performance of our random forest models especially on the testing set, we'll be looking to plot and compare the different **ROC curves** corresponding to area under the curve values, as well as **variable importance plots** assessed by measures of Mean Decrease Accuracy and Mean Decrease Gini to relatively compare the significance of our predictors in classification. These plots will be helpful to extract any strong associations with drug susceptibility and resistance. If of interest, we will also produce tables for **confusion matrices** as well as tables to compare the following **performance metrics**: accuracy, sensitivity, specificity, and area under the curve. Finally, we will compile tables to demonstrate the antibiotics that we predict and recommend our isolates will show strong susceptibility to for both our training and testing set, with emphasis on prediction on our testing set. We will also provide with what level of confidence these drugs have been recommended by our algorithm which will help to assess its efficiency.

Approaching our study with the random forest classifier model is appropriate to our aims, especially considering that we can assess our levels of confidence in prediction as random forest implementation in R doesn't only return the predicted class of each observation, but also the probabilities associated with those

predictions. Additionally, random forests is a type of non-parametric supervised machine learning model so we don't have to be held to stringent assumptions of our data like we would with parametric techniques like logistic regression. We are also interested generally in the kinds of patterns that we can extract from our current data, especially when it comes to the most pertinent covariates in the classification processes. We want to find which antimicrobial genes as well as which baseline covariates are most helpful in deciding how to treat cases of foodborne illness in Salmonella, E.coli, and Campylobacter. And so, what is especially handy about random forests is its emphasis on feature selection.

As feature selection is a built-in mechanism on random forests, the output of our models are able to provide a easily interpretable measure of how integral each variable of interest is to classification. This process will be paramount in extracting tangible insights about associations between drug susceptibility and resistance, and the supporting characteristic information logged in the NCBI database about each new case. Even pertaining to our goal of assessing how time may play a roll in predicting drug response, the mechanisms of the random forest algorithm in R makes it easy to assess the significance of time comparative to the other variables considered in modeling.

The biggest limitation of implementing this model on our data is that it assumes our data are complete and non-missing. As touched on earlier, to remedy this, we utilized multiple imputation.

Initial Implementation

We'll demonstrate the utilities of random forests on our data with a simple use case. For this example, we used one of the imputed datasets for the Salmonella enterica isolates and focused on the isolates' resistant/susceptible response to the antibiotic, tetracycline.

However, as aforementioned, before we ran this implementation, we first utilized the `Boruta` function from the `Boruta` package in order to perform variable selection.

```
boruta <- Boruta(tetracycline ~ ., data = imp_data, doTrace = 2, maxRuns = 500)
```

From 12 iterations, the Boruta algorithm decided that none of the 22 predictors of consideration were unimportant, so we used them all in the example classification.

We fit a model with drug susceptibility to the antibiotic, tetracycline, as our outcome of interest (a value of 1 for susceptible and a value of 0 for resistant), and the 22 variables collection date, isolation type, minimum SNP distance to an isolate of the same isolation type, minimum SNP distance to an isolate of a different isolation type, serovar, country, isolation source, number of isolates "close" to that isolate (being close to another isolate being defined as having a minimum SNP distance less than or equal to 7 to an isolate of either the same or different isolation type), antigen formula, average similarity score between isolates within the same SNP cluster of antibiotics that isolates show susceptibility to, and finally a set of 12 binary variables indicating the genetic presence of 12 of the most common antimicrobial resistant genes in Salmonella.

Note that our outcome of interest, susceptibility on tetracycline is roughly balanced with 51.3% of the isolates showing resistance to tetracycline, and the remaining 48.8% showing susceptibility.

```
set.seed(1)
rand_forest_test <- randomForest(tetracycline ~., data = imp_data)
```

The output of the random forest command gives the number of trees that were grown as a part of the process which in this case was the default 500 trees, and the number of variables tried at each split, which was 4, i.e. the value found by taking the largest integer value less than or equal to, otherwise known as the *floor* value of the square root of the number of columns in the dataset (floor of the $\sqrt{24}$). We will tune these values for the official study.

The random forest function also provides a measure of the out of bag estimate of the error rate. While this is reported as 2.41% in the model output, we can also plot the out-of-bag error against the number of trees overall and by class.

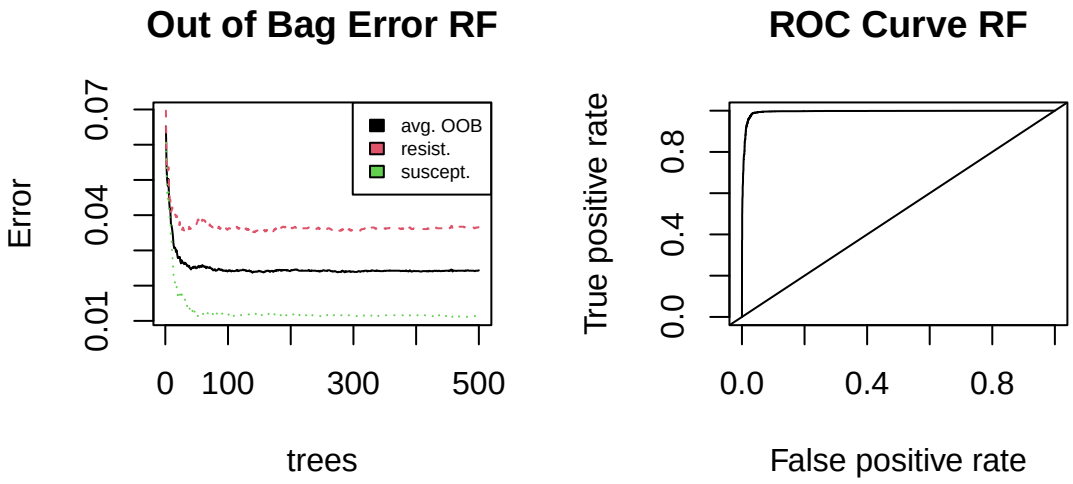

Figure 1: Out of Bag Error Plot and ROC Curve for Tetracycline RF

Table 3: Tetracycline RF Performance Metrics

Metric	Value
Accuracy	0.9756435
Sensitivity	0.9624424
Specificity	0.9888966
AUC	0.9944642

Notably, **Figure 1 (left)** shows that our model is stronger at predicting the tetracycline susceptibility class than the resistant class, a plot that can be supplemented with other performance related visuals like ROC curve which we plot alongside the out of bag error plot **Figure 1 (right)**.

We also can extract the statistics of interest based on the confusion matrix outputted from predicted classification. We just collected these metrics on our full dataset for now for demonstration purposes (**Table 3**).

We will also extract information about the most important predictors. Ultimately, we'll be comparing the results and important predictors of multiple models with susceptibility/resistance responses to each of the antibiotics of interest, in order to recommend antibiotics that we predict new isolates will show susceptibility to. In this simple case though, we show what a simple plot of this comparison would look like (**Figure 2**).

Moving forward, we'll work to properly tune the hyperparameters within our random forest models and systematically expand this toy example to predict isolates' performances on larger sets of antibiotics which will hopefully retain usefulness in the assessment of future isolates to come.

References

Stef van Buuren, Karin Groothuis-Oudshoorn (2011). mice: Multivariate Imputation by Chained Equations in R. Journal of Statistical Software, 45(3), 1-67. DOI 10.18637/jss.v045.i03.

A. Liaw and M. Wiener (2002). Classification and Regression by randomForest. R News 2(3), 18–22.

Miron B. Kursa, Witold R. Rudnicki (2010). Feature Selection with the Boruta Package. Journal of Statistical Software, 36(11), 1-13. URL http://www.jstatsoft.org/v36/i11/.

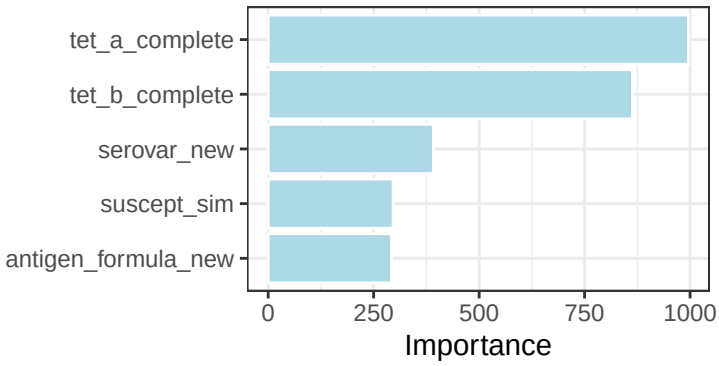

Figure 2: Top 5 Most Important Covariates