CMM: Breast Cancer ChemoRepsonse Classification

Yimeng He
Department of Applied Math and Statistics
Johns Hopkins University
Baltimore, MD 21218
yhe79@jhu.edu

Jialin Guo
Department of Applied Math and Statistics
Johns Hopkins University
Baltimore, MD 21218
jguo81@jhu.edu

Evan Mata
Department of Applied Math and Statistics
Johns Hopkins University
Baltimore, MD 21218
emata2@jhu.edu

Abstract

Breast cancer is a serious disease that affects millions of people worldwide, and chemotherapy is an important treatment option that can help to target cancer cells and improve patient outcomes. Hence, the prediction of patients being responsive to chemotherapy, 'ChemoRepsonse', is essential. In this project, we analyze individuals' genetics and metadata to gain insights into the problem. Then we clean the data carefully, selecting only the most relevant features with maximum parsimony. We apply various statistical and machine-learning algorithms for preliminary analysis and finally pick random forest, XG boosting, Ada boosting and voting for classifiers. K-fold cross validation of $k = 10$ is also used to tune hyperparameters. Finally, we select the best model that outperfroms all others and evaluate its performance.

1 Introduction

Breast cancer often refers to a malignant tumor forms in the cells of the breast. It is one of the most common types of cancer diagnosed by women worldwide. Chemotherapy is an important treatment option for breast cancer as it kills cancer cells, shrinks tumor size and has many other advantages. The treatment can be given either before or after surgery depending on patient's response. However, it's crucial to determine beforehand whether a patient will respond to chemotherapy. They could be sensitive(1) or resistant(0). If they are sensitive, then it's better to take advantage of the therapy before surgery. In this project, we aim to predict the ChemoResponse of patients with breast cancer with the given dataset BR0. We treat ChemoResponse as our response variable(Y), and all other information as predictor variables(X).

2 Data Prepossessing

Our dataset (BRO) is consists of two parts. BRO genome contains 1,171 rows of patients and 13,300 columns of gene expressions like RFC2, HSPA6, PAX8. BRO meta contains 1,171 rows of patients and 12 columns of clinical variables like Age, ER, PR, HER2 Status. Our phenotype of Interest is located at last column of ChemoResponse (sensitive/resistant).

2.1 Genome Data

There are 13300 columns and 1171 observations in the Genomic data. All columns are continuous variables representing the gene expression levels for the given Gene (column). There are no missing values or unknowns.

Some genes had only non-negative gene expression levels, others had negative gene expression levels as well. For genes with non-negative gene expression levels, we linearly scaled them using min-max scaling to the range $[0, 1]$. For genes with negative gene expression levels, we instead scaled them to the $[-1, 1]$ range.

We performed scaling as non-linear models may benefit from feature scaling, especially when attempting to combine features of potentially different orders of magnitude in the raw data. It also increases the performance speed of some algorithms. This sort of scaling is not expected to affect purely linear models (such as Linear Regression) or Tree based models.

2.2 Meta Data

There are 13 columns and 1171 observations in the Meta data. It includes 1 target variable called ChemoResponse, 1 continuous variable, and 9 categorical variables.

There are some columns with both "unknown" and empty cells. Empty cells are imputed with "unknown" values. We first impute the missing data, and create an indicator column of when the imputed data was unknown for each column with any "unknown" or blank entries. Then we encode categorical data into numeric data. After that, the min-max scaler is used to decrease the influence of data with different magnitudes. An edge feature called 'DRFS_Year' will also be discussed in this section.

2.2.1 Data Imputation

Data imputation is a process of replacing missing values with substituted ones. It is imperative in machine learning because one can not apply any models to a data set with missing values. Also, this process can have a massive impact on the models' performance.

Assigning too many missing values may reduce the quality of data and lead to biased estimation. So we split features into two classes: features with less than 20% missing values and features with more than 20% missing values. We call them complete data and incomplete data, respectively.

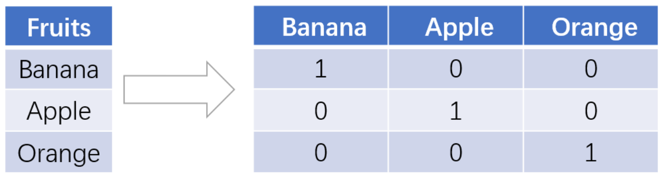
For complete data, we simply fill in the mode of that specific feature. For incomplete data, we will fill the unknown values with 'unknown', and treat 'unknown' as a separate class.

2.2.2 Categorical Data Encoding

After filling in the missing values, we need to encode those categorical data into numeric forms so that we can train the model on them.

For this specific data set, we decide to use one-hot encoding to do encoding. One-hot encoding is one of the most prevailing methods to encode categorical data. It transforms each category of a categorical feature into a binary feature that takes on a value of 1 if the category is present and 0 otherwise.

Considering 1, there are three categories in the fruit column. So the encoded data frame should have three columns. Then we just need to assign 1s accordingly.



Fruits
Banana
Apple
Orange

Banana	Apple	Orange
1	0	0
0	1	0
0	0	1

Figure 1: Example of OH Encoding

2.2.3 Edge cases

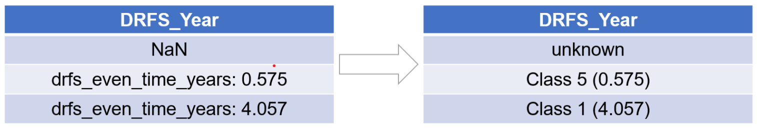
There is a special feature called 'DRFS_Year'. It uses string to denote years, and it also has over 50% of NaNs.

According to the rule described in the previous section. 'DRFS_Year' should be classified as incomplete data. As a result, we will impute the missing values with 'unknown' and treat each value in the feature as a different category.

Due to the special format of this feature. We decide to do the following arrangements.

1. Parse years from each entry.
2. Derive the distribution of years.
3. Divide those entries into 5 classes evenly.
4. Replace NaNs with 'unkown'.

For example, in 2, we first parse the year into numeric form (e.g. 2^{nd} row 0.575). After this operation, we can get the distribution of years. Then we divide the years into 5 classes and use the class number to replace years. In this example, 0.575 years of DRFS is assigned to class 5, and 4.057 to class 1. Finally, we replace NaN with unknown.



DRFS_Year
NaN
drfs_even_time_years: 0.575
drfs_even_time_years: 4.057

DRFS_Year
unknown
Class 5 (0.575)
Class 1 (4.057)

Figure 2: Example of DRFS_Year

2.2.4 Feature Scaling

Performing feature scaling can mitigate the influence of data with different magnitudes and could potentially enhance the performance in linear models.

We perform min-max scaling on the only continuous feature 'Age'. All data points will fall between 0-1 after this operation.

Min-Max Scaling is a common technique to scale the data between 0 and 1.

2.3 Dimensionality Reduction

Dimensionality Reduction is a technique where the number of features in a dataset is reduced. This has been shown to increase model performance when there is a finite amount of data, and limits the curse of dimensionality. It can significantly help with model speed, as well as with model performance. There are a variety of different potential dimensionality reduction techniques, we experimented with Mutual Information and Singular Value Decomposition (SVD).

We observed that models trained on our dataset when reduced via Mutual Information always outperformed models trained using data reduced using Singular Value Decomposition, so we report results assuming the dimensionality reduction was done using Mutual Information.

2.3.1 Mutual Information

Mutual Information is an information theoretic metric that evaluates how much predictive power a given variable or variables has for another set of variable(s). We can perform mutual information between each gene and our output ChemoResponse, then observe which genes have the highest mutual information with ChemoResponse and choose to only use that subset of genes. Mutual information, unlike alot of other dimensionality reduction methods, allows us to reduce our feature space but still keep our exact same feature names. This aids in explainability of our model.

We note that there were a large number of genes whos mutual information was effectively 0, but that our mutual information values were distributed in an almost linear decay aside from those, as one can see in 3. This implies that our data is unlikely to be fully explained by only one or two genes along the axes we care about.

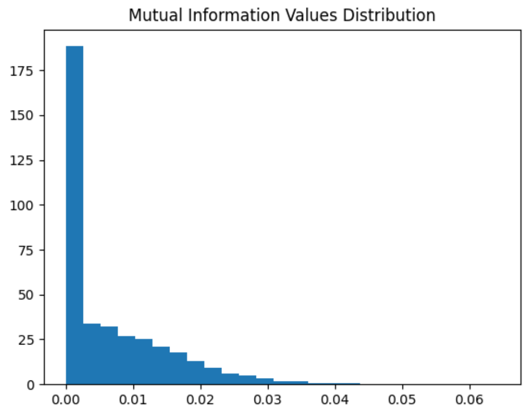


Figure 3: Distribution of Mutual Information Values

2.3.2 Singular Value Decomposition

Singular Value Decomposition, also known as SVD, decomposes the matrix of data points vs features into its singular values. It then keeps the chosen number of largest singular values (k in figure 4), and uses them to reconstruct an approximation of the input matrix in a lower dimension. This can be seen in 4. [Albatayneh et al. (2018)]

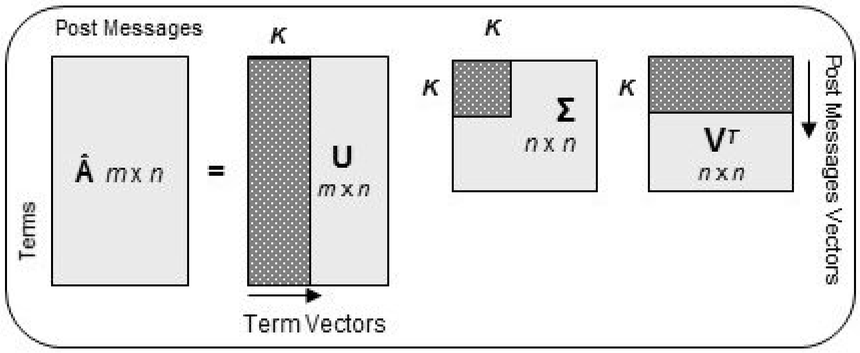


Figure 4: Truncated Singular Value Decomposition

Singular Value Decomposition's focus on the spectrum of your matrix enables it to preserve the maximum amount of variation within your data. However, this performed worse than dimensionality reduction in all our tested cases.

We hypothesize that this is because our number of features is so much larger than our number of training samples, so we already had a huge amount of variation. More specifically, SVD is preserving variation across the total dataset, not necessarily variation amongst the best predicting genes. It could be preserving variation that has no effect on our prediction, and since there are so many more features than data points, we expect there to be a relatively high amount of variation not relevant to our prediction.

We did note that the highest singular value was approximately 2150, while the 2nd highest was approximately 350. There is an order of magnitude difference between the two. There are significantly more singular values relatively close to 0. This implies there is a dominant eigenvector which separates our data, but the separation is not necessarily along the axes we care about, eg the separation is not necessarily along the lines of chemoResponse. We see this in 5, the yellow and blue dots represent people who are responsive to chemo therapy or not; our data is well separated into 4 clusters, its just that those clusters are not along the sensisitive/resistant axes we care about.

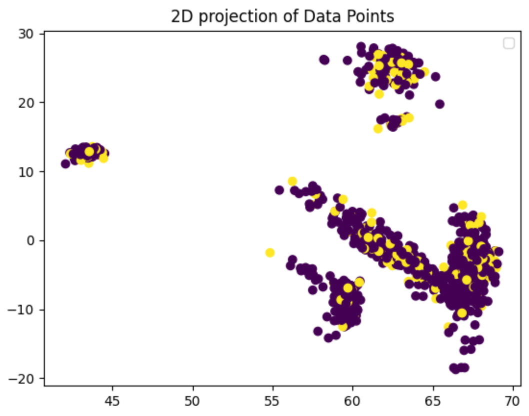
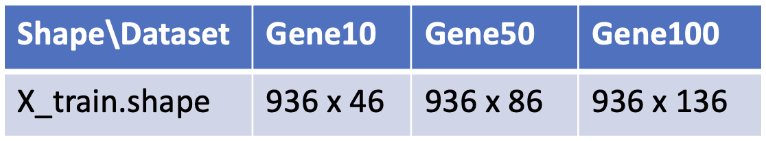


Figure 5: TruncatedSVD Projection into 2 dimensions

2.4 Data Concatenation

To avoid model overparameterization, and to preserve model interpretability, we choose 10, 50, and 100 columns from *geno* with the highest mutual information and concatenate them with the encoded *meta*

We finally get three data set *gene10*, *gene50*, *gene100*. [6]



Shape\Dataset	Gene10	Gene50	Gene100
X_train.shape	936 x 46	936 x 86	936 x 136

Figure 6: Three Datasets

3 Approach

In this section, we divide up each of the three datasets each into three parts, and apply a number of popular classifiers. Then, we select the methods with the best overall performances and use k-fold cross validation to tune their hyperparameters. Finally, we apply the parameter of our choice on *gene* datasets and view the confusion matrix.

3.1 Data Segmentation

We split up our data into three parts: Training sets, Testing sets and Dev sets. We use training sets for model fitting and hyper-parameter tuning; use test sets for model performance evaluation and selection; finally, after selecting the optimal technique, we apply the prediction on dev sets to visualize final results and draw conclusions.

From a preliminary review, we notice that in 'ChemoResponse' column, there is about 238 labels belong to Sensitive(1), and 933 labels belong to Resistant(0). The rate is about 1:4, and the dataset is extremely unbalanced. To ensure an evenly split of data in each class, we define a function `train_test_dev_split()` apply stratified random sampling, such that in each set (train, test or dev) the rate of sensitivity to resistance response is about 20% to 80%. As shown in Figure [7]

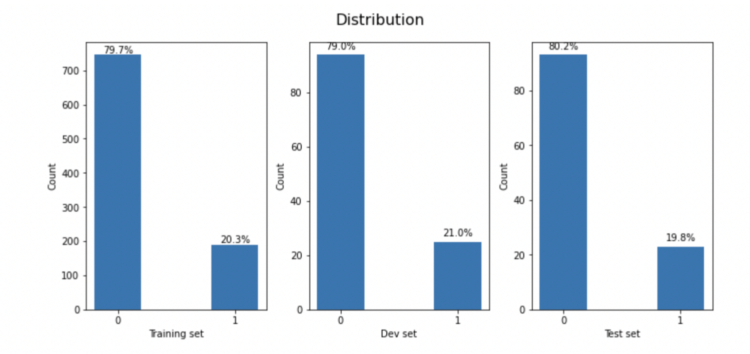


Figure 7: Sensitive to Resistance rate: 1:4

3.2 Model Selection

After splitting data, we apply our data on eight commonly used and popular classification techniques and visualize results using `matplotlib.pyplot`. Below are the methods and their brief description:

- **LDA(Linear Discriminant Analysis)**: find a linear combination of different features that separate classes and assumes the same variance among classes.
- **QDA(Quadratic Discriminant Analysis)**: extension of LDA, except it assumes different variances for each class.
- **GNB(Gaussian Naive Bayes)**: assume a Gaussian normal distribution and conditional independence among classes.
- **SVM(Support Vector Machine)**: find the hyperplane that best separates classes, and can have non-linear boundaries.
- **RF(Random Forest)**: train multiple decision trees on randomly selected features and predict.
- **KNN(kth Nearest Neighbor)**: classify based on the majority class of its kth nearest neighbor.
- **XGB(XG Boosting)**: a machine learning method that uses a decision tree and each time learns from falsely classified examples and combines their predictions before making the final prediction.
- **AdaBoost(Adaptive Boosting)**: assign weights to previously classified samples and train weak learners on these examples and combine predictions to make the final prediction.

We note that the models which assume normality (or approximate normality) of data are reasonably well justified when one looks at the distribution of different gene expression levels as shown in Figure [8]. We see that given 3 gene's gene expression level distributions, after linear min-max scaling, we have approximately normal distributions. This then serves to justify our choice of models.

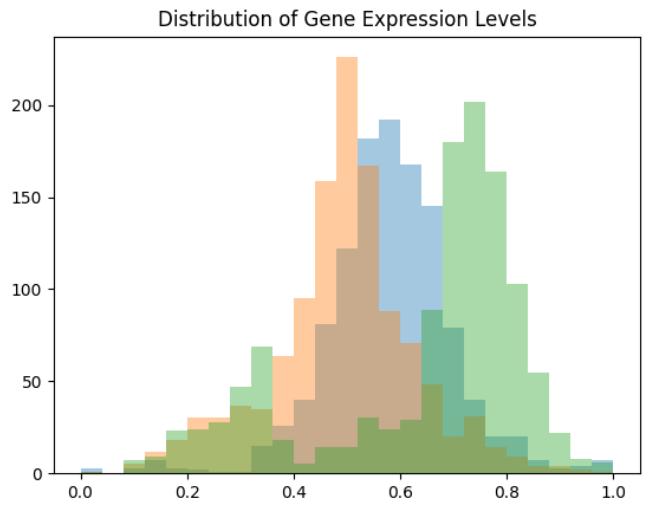


Figure 8: Distributions of 3 Gene Expression Levels

Among all, The model performance is evaluated through accuracy rate. The graphs are as follows in Figure [91011]. After observing their results, we choose three methods: **Random Forest, XG Boosting, Ada Boosting** as our classifiers. All of them are ensemble methods that utilize multiple weak classifiers (like decision trees) and combine their results and improve prediction accuracy. Furthermore, they can handle datasets consisting of both categorical and discrete data and can be tuned for hyperparameters. In the following section, we use k-fold validation to tune hyperparameters in Random Forest and XG boosting.

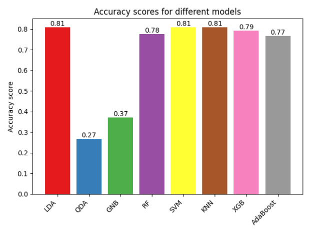


Figure 9: gene10

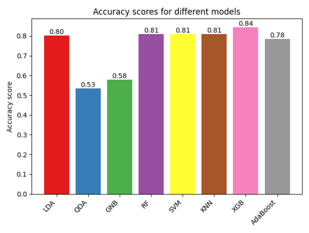


Figure 10: gene50

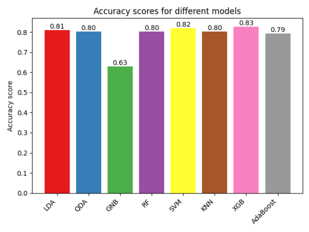


Figure 11: gene100

3.3 K-Fold Cross Validation

The validation method is essential in model training as they help improve model generation. If we always train our model on fixed test sets, they are exposed to overfitting and receive optimistic results that have very poor generation ability. The validation technique helps to reduce such situations and can tune for hyperparameters that generalize well on new data. Here, we use k-fold cross-validation and set $k = 10$. [12,Ashfaque and Iqbal (2019)]

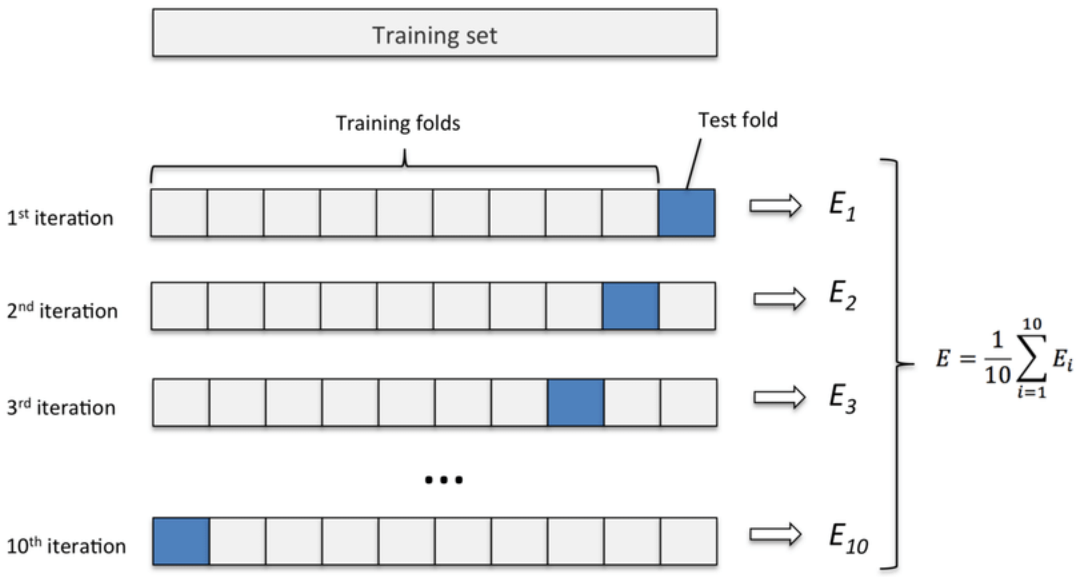


Figure 12: K-fold with k=10

Here, we partition the training sets into ten equally sized folds. In each iteration, one of the folds is used as a validation set and the other folds are training sets. We use AUC (Area Under Curve) score as the primary evaluation standard, as we care about both sensitivity and specificity, and find the best tree depth from `max_depth` in [1, 3, 5, 10, 20, 30, 40, 50] that has the highest mean AUC score. Plots of each fold are generated and the best parameter is stored. The example plots of *gene50* and charts of all datasets are as follows. We note that it is usually possible to tell when overfitting occurs as AUC begins to dip down after initially rising. [131415]

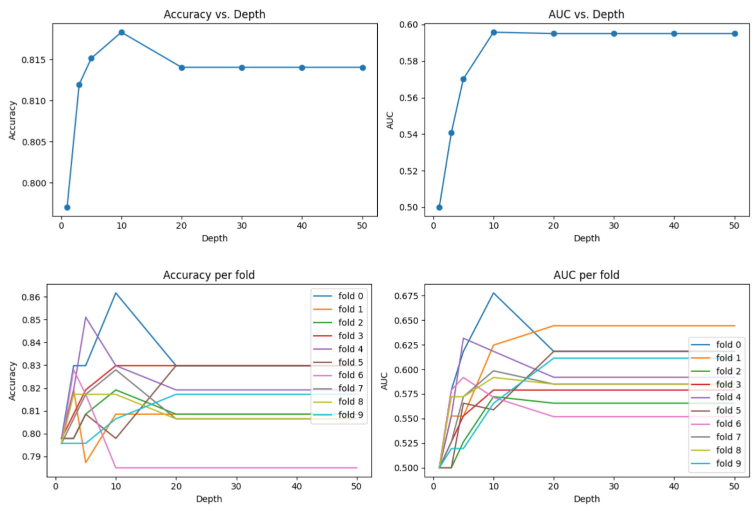


Figure 13: K-fold for RF at *gene50*

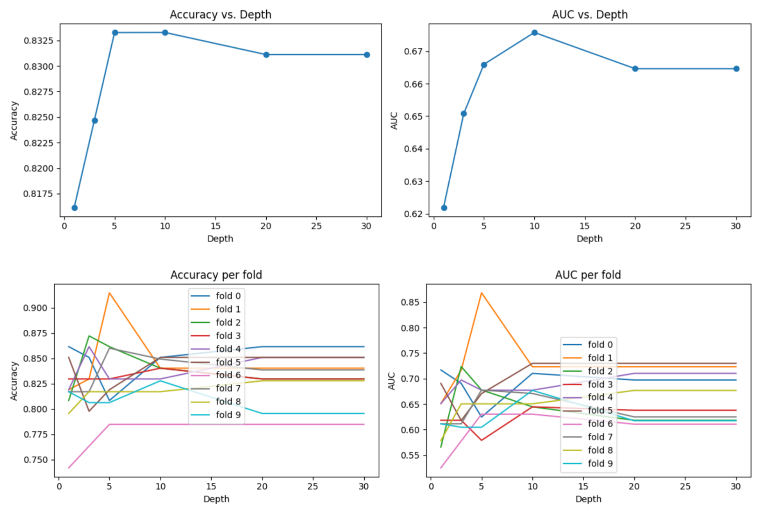


Figure 14: K-fold for XGB at *gene50*

Best Parameter	Gene10	Gene50	Gene100
Random Forest	20	10	20
XG Boosting	5	10	5

Figure 15: Charts for K-folds for all datasets

3.4 Dataset Selection

Out of the three datasets, we want to choose the one that best represents the original data and can generalize the model well. Based on validation results, *gene10* performs well on random forest with depth 20. With limited parameters and a deep depth, we believe *gene10* does not give sufficient information and is at risk of overfitting, so it cannot represent the entire data. Hence, we only compare the performances of *gene50, gene100*.

3.5 *gene50* vs. *gene100*

We use confusion matrices to visualize performances of *gene50* and *gene100*. We see these in the three plots[161718], with the matrix on left is *gene50*, and one the right is *gene100*. Clearly, we see *gene50* achieves better performances with less false positive predictions, and more true positive and true negative predictions. Overall, we decide to use *gene50* dataset.

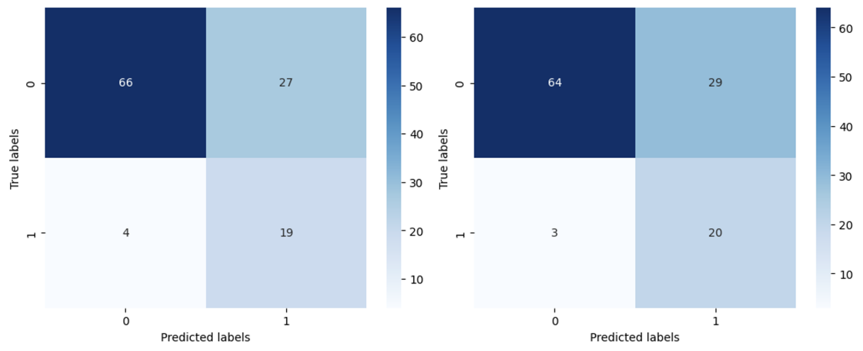


Figure 16: Confusion Matrix on Random Forest

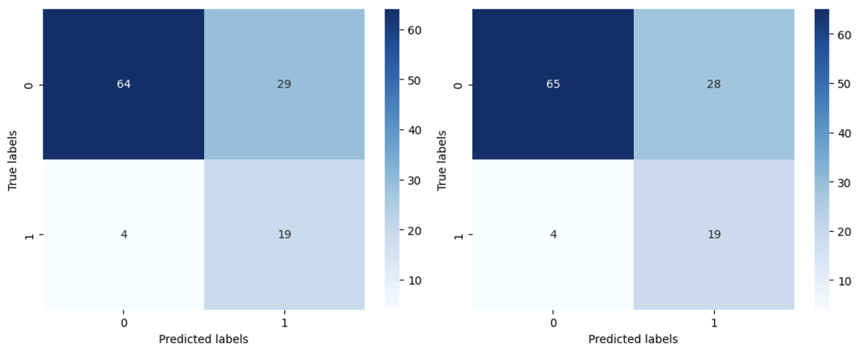


Figure 17: Confusion Matrix on XG Boosting

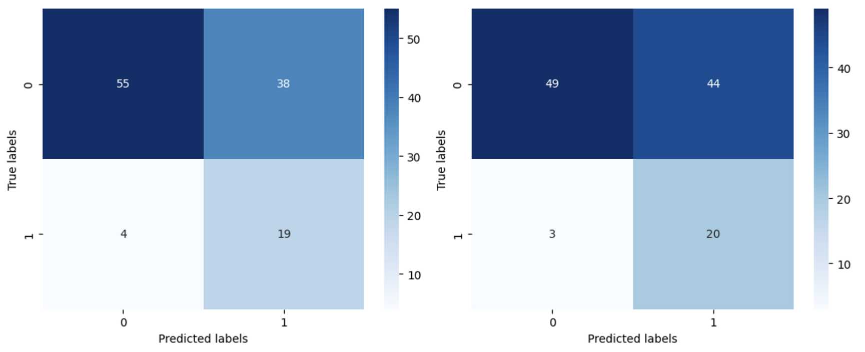


Figure 18: Confusion Matrix on Ada Boosting

4 Voting Algorithm

The voting algorithm or classifier is an algorithm that considers the prediction of several models. There are typically two types of voting, 'hard voting' and 'soft voting'.

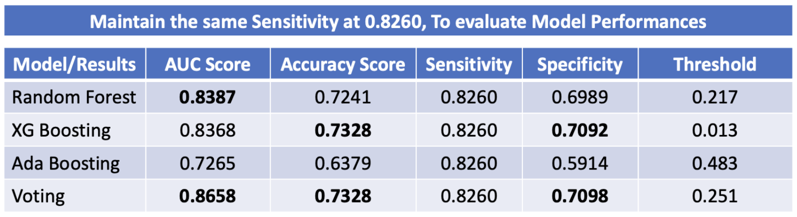
Hard voting is where you take in the prediction results of the ensemble of models (i.e. models you choose for voting) and take the majority class as your result. For example, if we have 3 models doing the binary classification task, then we'd have 3 models in the voting classifier. Assume their predictions are 1,0,1. Then, the majority vote is 1.

Soft voting, however, considers the probability of the ensemble of models. It outputs the mean of probability predicted. Looking at the previous example, assume the 3 models output 0.5, 0.7, and 0.2 probabilities, respectively. The voting classifier will produce the mean $(0.2+0.5+0.7)/3 = 0.46667$ as the output. We used the arithmetic mean in this example, but other metrics can also be used.

The voting algorithm is a good option when a single classifier shows some bias towards some specific factors. It helps the researcher to derive a model that is more robust. Moreover, the voting classifier could potentially have better performance than any other models in the ensemble. In fact, it is one of the most commonly used methods in data science competitions. Those champions always utilized the results from different models for their final models. Typically, it gives them a better chance to win.

5 Results

Now, we apply Random Forest, XG Boosting, Ada Boosting, and Voting on *gene50* to predict labels for y. To evaluate classifiers' performances, we choose the threshold that set the sensitivity of each model at 0.8, and compare the model's auc score, accuracy score, and specificity. As in Figure[19], it is clear that **Voting** outperforms all other classifiers on an overall basis. Its robustness is reasonable since voting as an ensemble learning method is a combination of many models. Hence, we pick **Voting Algorithm** as our final classifier. The threshold column represents the tuned threshold predictions needed to surpass for the model to predict as positive chemoResponse.

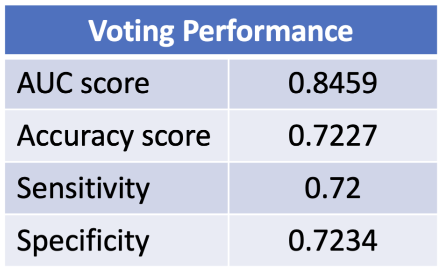


Maintain the same Sensitivity at 0.8260, To evaluate Model Performances					
Model/Results	AUC Score	Accuracy Score	Sensitivity	Specificity	Threshold
Random Forest	**0.8387**	0.7241	0.8260	0.6989	0.217
XG Boosting	0.8368	**0.7328**	0.8260	**0.7092**	0.013
Ada Boosting	0.7265	0.6379	0.8260	0.5914	0.483
Voting	**0.8658**	**0.7328**	0.8260	**0.7098**	0.251

Figure 19: Final Result

5.1 Performance on Dev Sets

Now that we have pick our classifier, we would like to test its performance on a previously unseen datasets, the Development set. The result, confusion matrix and roc curve are as follow: [202122].



Voting Performance	
AUC score	0.8459
Accuracy score	0.7227
Sensitivity	0.72
Specificity	0.7234

Figure 20: Tests on dev set

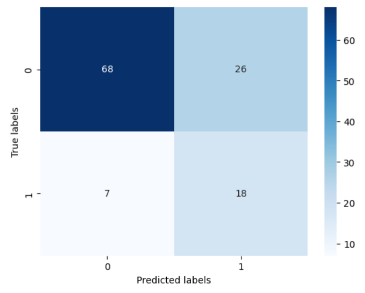


Figure 21: Confusion Matrix on dev set

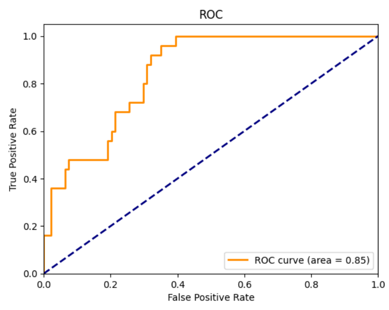


Figure 22: ROC curve on dev set

As expected, the final model performed worse in most respects on the development set relative to the training and testing set. Interestingly, the specificity of the model actually increased.

5.2 Robustness of Results

It is always important to characterize your models performance and ensure that it is not the result of random chance. Furthermore, we are exploring whether gene expression levels allow us to make accurate predictions, in which case we also may wish to compare with other models that do not include gene expression information.

5.2.1 Metadata Baseline

In order to identify whether our genomic data added predictive power, we trained a similar voting classifier with identical parameters for its Random Forest, XG Boost, and Ada Boost submodels and subject to the same constraints. However, this other voting classifier was only fed in the metadata for our genomic dataset. We observed that the metadata only model performed worse in all respects relative to the genomic-inclusive model.

We would characterize the differences between the metadata only model and the genomic inclusive model as significant, as when we trained 100 different metadata models under permutation testing, the genomic-inclusive model was many standard deviations beyond the best performing metadata only model.

5.2.2 Permutation Testing

We always wish to compare the results that were obtained with random chance to ensure they are robust. One method of doing so is to perform permutation testing. In permutation testing, one randomly permutes the class labels of the dataset many times, in our case 10,000, and compares their results metrics with those achieved by the random chance models. We observed that our AUC score, the metric chosen for evaluation, was the best out of all 10,000; passing a bonferoni correction of 0.05 when accounting for the number of hypothesis tested. We thus conclude that our model is not simply the result of random chance.

6 Conclusion

There are several lessons learned from this project. First and foremost may be that empirical data is always necessary. A priori, we may know that dimensionality reduction is expected to help our classification. However, we don't know exactly how much dimensionality reduction to do, eg if reducing to 50 or 100 dimensions will perform better. The overall state space of possible models across different dimensions is too large to fully explore, but we do need empirical data to at least reduce the range of possibilities.

Additionally, we note that increased complexity of a model does not always yield better performance. An additional model tested was Denoising Auto Encoders, but this more sophisticated model performed worse that many of our other models. While it may be valuable to include more sophisticated models, it is important to ensure that they fit the scenario you are working with. This could include hard constraints such as performance metrics, but also soft constraints such as the explainability of your model.

We also note that any data augmentation and preprocessing techniques must be carefully considered in the context of your problem and models. For instance, we initially included upsampling in our preprocessing. This method of data augmentation creates additional synthetic samples of your classes with relatively less data. While upsampling did increase our sensitivity, it actually decreased our specificity. If we care more about our specificity, we would choose not to upsample our data. Thus which data augmentation techniques we performed have to be carefully chosen in the context of our problem. It may be possible for instance to carefully tune the loss function to care more about sensitivity or specificity or less frequent class occurances.

Furthermore, we observe that the modelling assumptions of a given class of model should be taken into account prior to attempting to use the given kind of model. As noted, the state space of possible models is effectively infinite. Thus we wish to reduce it as much as possible using information such as modelling assumptions. If our gene expression data was not approximately normal, it would be much less likely that Gaussian Naive Bayes models would work.

Finally, we believe that it is important to always check the robustness of our results. While a result may appear to be "good", it is always important to clarify what good is. Are we performing significantly better than random chance, given the number of models tested? How do we compare to a baseline? These questions will always be relevant for any modelling done.

References

Albatayneh, N. A., Ghauth, K. I., and Chua, F.-F. (2018). Utilizing learners' negative ratings in semantic content-based recommender system for e-learning forum. *Journal of Educational Technology & Society*, 21(1):112–125.

Ashfaque, J. M. and Iqbal, A. (2019). Introduction to support vector machines and kernel methods. *publication at https://www. researchgate. net/publication/332370436*.