Mechanisms of Action (MoA) Prediction

Table of Contents

1. Introduction

This document outlines the steps taken for MIE1628: Big Data Science final project during the Fall 2020 semester. The project was based on Kaggle's Mechanisms of Action (MoA) Prediction competition[1]. The objective was to determine the MoA of a drug based on a patient's gene expressions and cell viability. The problem at hand was formerly a supervised problem and a multi-label classification problem and we chose a binary relevance approach for it. The final target and the overall score of our models was evaluated based on the overall mean of BCE losses across samples and all labels.
The *Data Exploration* section provides an overview of the unique challenges of the MoA data including high dimensionality and imbalance distribution of labels in addition to the data exploration steps we have taken. The *Methodology* section provides an overview of the data-preprocessing techniques and machine learning methods we have used to face the unique challenges of MoA competition and the experiments we designed to put our approach to test. The *Evaluation* section provides an overview of the results of our experiments. The *Discussion* section provides some of the approaches we tried but we did not necessarily include in our pipeline. Finally, the *Conclusions* provides a critical look back on our team's work during the Fall 2002 semester and suggests some future steps for potentially improving the current results.

2. Data Exploration

The MoA train dataset contains 23815 samples and 875 features for each sample. From the 23815 samples, 1866 belong to the control group. From the 876 features, 3 of which are categorical (`cp_time`, `cp_dose`, `cp_type`) variables and the rest are continuous variables describing the cell and genomic data. Each sample has a unique signature id and the dataset does not have any missing values. The samples belonging to the control group are indicated as such in both the train and the test set. Meanwhile, they do not have any label activations across the dataset. We therefore decided to disregard the samples from the control group all together in our analysis and machine learning pipeline.

The MoA train dataset is accompanied by a targets dataset which consists of 206 unique binary labels. The distribution of data in the targets dataset is highly imbalanced. Among the 23815 cases, there are merely between 1 to at most 832 activations for each label (See Figure 2.1). This imbalance in the target data and the high dimensionality of the training set (876 features) posed the two most serious challenges in designing our machine learning pipeline. We have dealt with these challenges with an original approach and a challenger approach. Each of these approaches will be explained in detail in the Methodology section.

[1] Link to the Kaggle competition: https://www.kaggle.com/c/lish-moa/

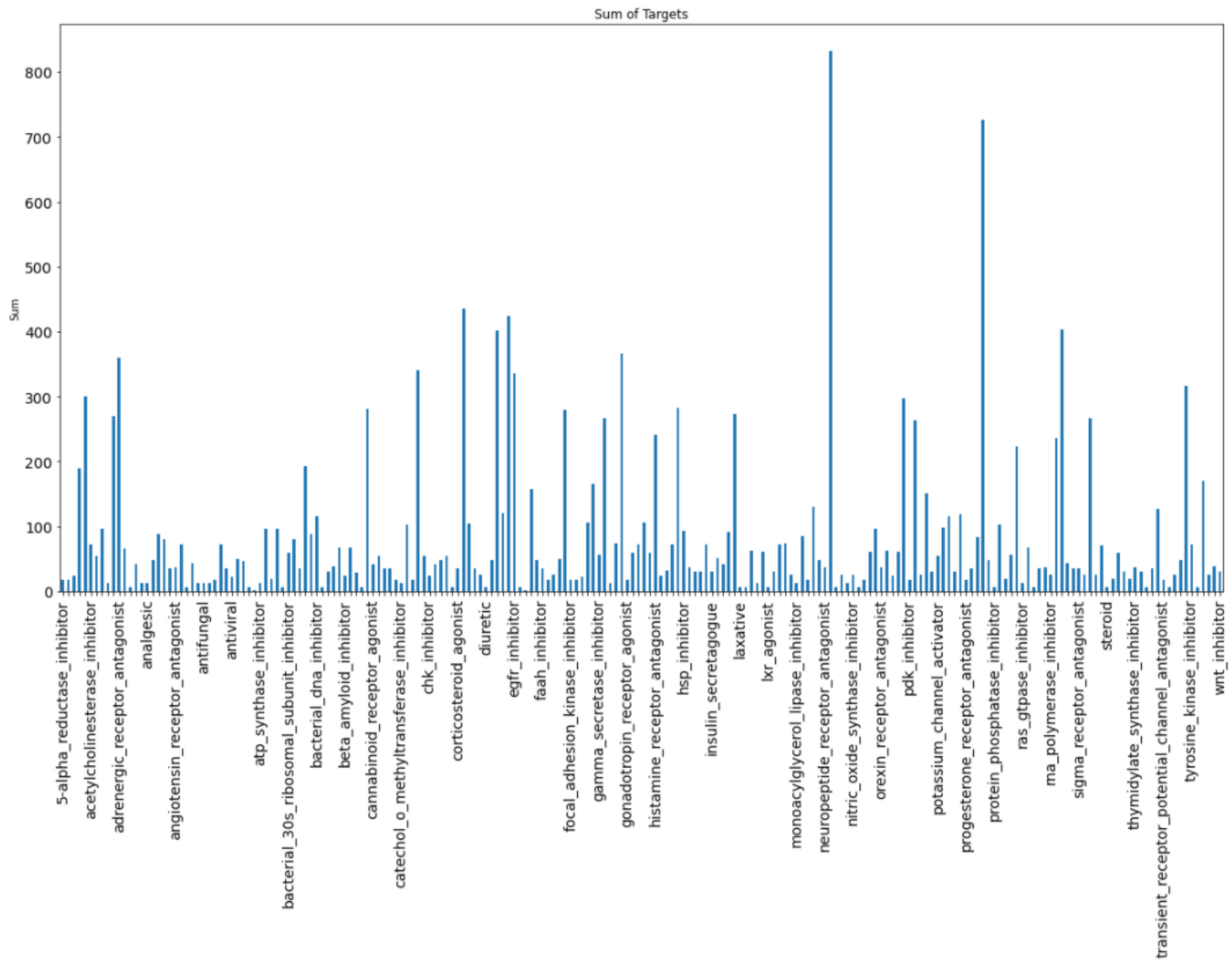

Figure 2.1: Count of each label's total activations over all the samples

Looking at the correlations among the continuous features and discovered that some of the features are highly correlated (see Figure 2.2). Knowing the large number of highly correlated features, we could later use the information to perform feature selection and lower the dimensionality of the data. Furthermore, we investigated the distributions of the cell features (`c-*`) and genomic features (`g-*`) across numerous train samples individually (See Figure 2.3-4). Most patients followed a similar pattern to graphs seen in Figure 2.3(bottom left and right), though the minimum, maximum and cell viability mean would vary by patient. There were a few patients which were extremely different for cell viability, this can be seen in Figure 2.4(bottom right) where most of the data would be near the minimum and there is a flat-line at -10. The other interesting thing to note is that in Figure 2.4(bottom left), the data flat-lines at -10 and 10 leading our team to believe that there may be a cap for the maximum and minimum of the data.These observations led us to engineer new features based on the minimum, maximum, and the mean of cell features and genomic features of each sample. This will be discussed further in the Methodology section.

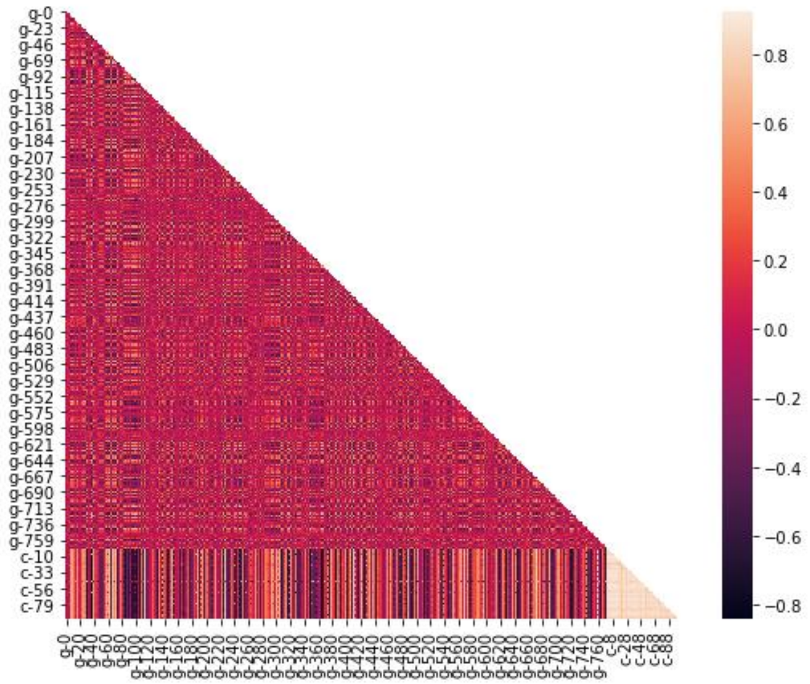

Figure 2.2: Correlation Heatmap gene and cell features

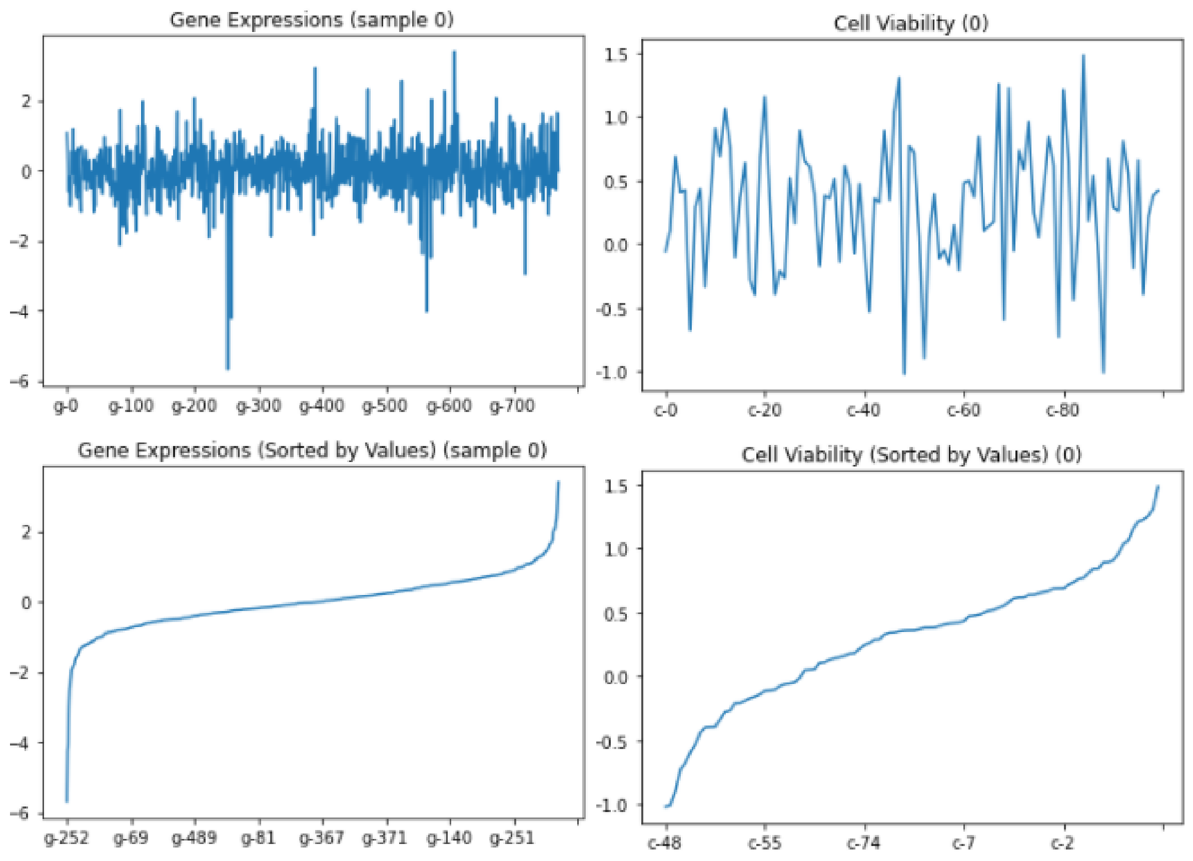

Figure 2.3: Line plot of the gene data (g-) and cell data (c-*) across the first train sample.*

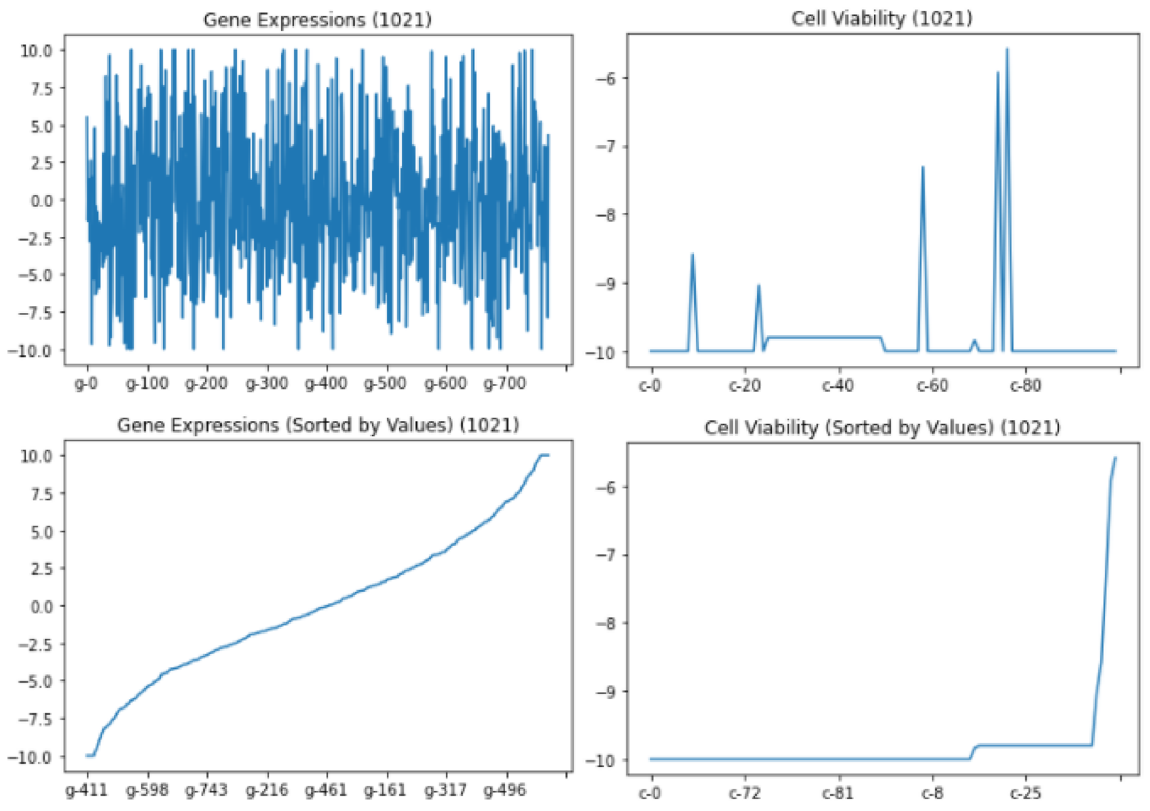

Figure 2.4: Line plot of the gene data (g-) and cell data (c-*) across the 1021st train sample.*

3. Methodology

This section provides a detailed overview of our data processing and machine learning pipeline. Per previous discussion, the two main challenges in the MoA multi label classification problem were the imbalance in the labels data and the high dimensionality of data. We used stratified cross validation along upsampling the minority class and the downsampling of the majority class to mitigate the data imbalance challenge. As for the problem of the high dimensionality of data we first used the correlations among the features to perform an initial feature selection and then, we explored two separate avenues with an original approach using PCA for feature reduction and a challenger approach using Random Forest for further feature selection. We finally trained Logistic Regression, GBT, and Random Forest models with hyper-parameter tuning. The results of these experiments are discussed in the Evaluation section.

3.1 Data Preprocessing

The training features in the data consisted of two categorical variables (`cp_time` and `cp_dose`), 772 continuous variables for genomic features (`g-0` to `g-771`) and 100 continuous variables for cell features (`c-0` to `c-99`). We used label encoding and one hot encoding to prepare the categorical features for the modelling and training. We also used standardization for each of the continuous variables.

Finally, we calculated the correlations between the continuous features (872 features in total) and used a threshold of 0.6 to perform an initial feature reduction. This way, we were able to reduce the dimensionality of data by 342 and kept 530 features from the cell and genomic features.

3.2 Feature Engineering

During the team's data exploration phase, certain visualization assisted our team in determining what engineered features to select. We engineered 15 features based on the statistical summary of the data (row-wise mean, standard deviation, sum, min, and max) and an additional feature that was created based on KMean clustering.

3.3 Statistical Analysis

We created 10 features that describe the row statistics. The statistics are: minimum/maximum value for the gene data and cell data, sum of gene data, sum of cell data, mean of gene data, mean of cell data, standard deviation of gene data, standard deviation of cell data.

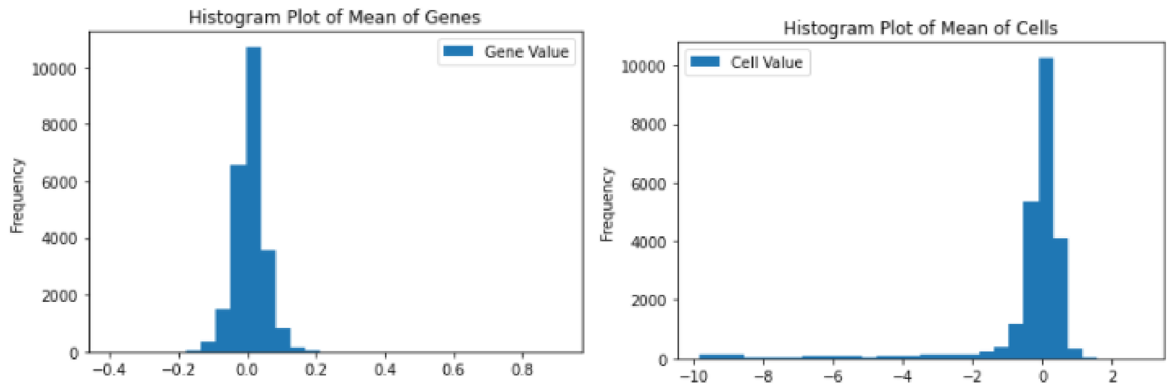

Figure 3.1: Histograms of the mean of gene data (g-) and cell data (c-*) across all train samples.*

Figure 3.1(left) is the engineered feature for the mean of gene data for each row. In Figure 3.1(left), the majority of the data is between -0.2 and 0.2. This agrees with our previous analysis that the gene features are mostly 0-mean. For the mean of cell data Figure 3.1(right), most of the data lies between -2 and 2 which is a much larger range. There is data located between -10 to -2 which can be useful when distinguishing certain ID's for certain targets as some drugs may only work when most of the cells are under a certain number..

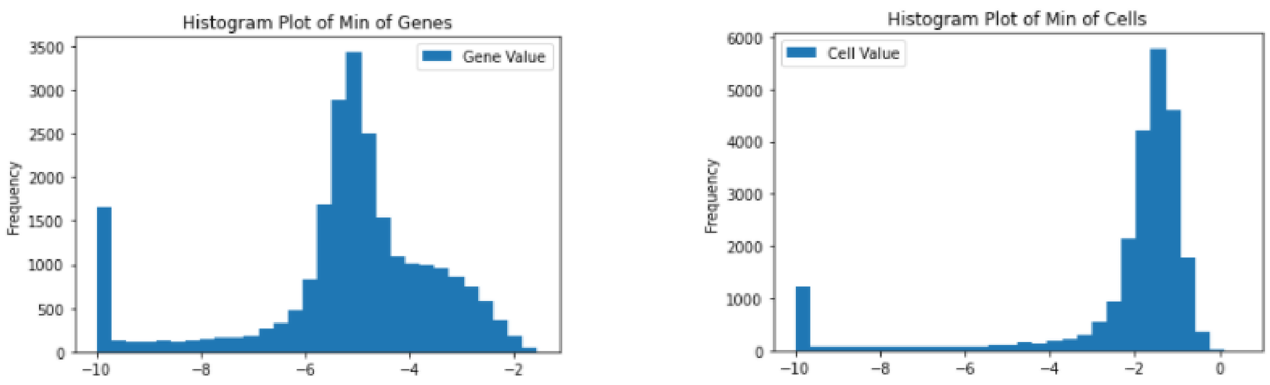

Figure 3.2: Histograms of the minimum value of gene data (g-) and cell data (c-*) across all train samples*

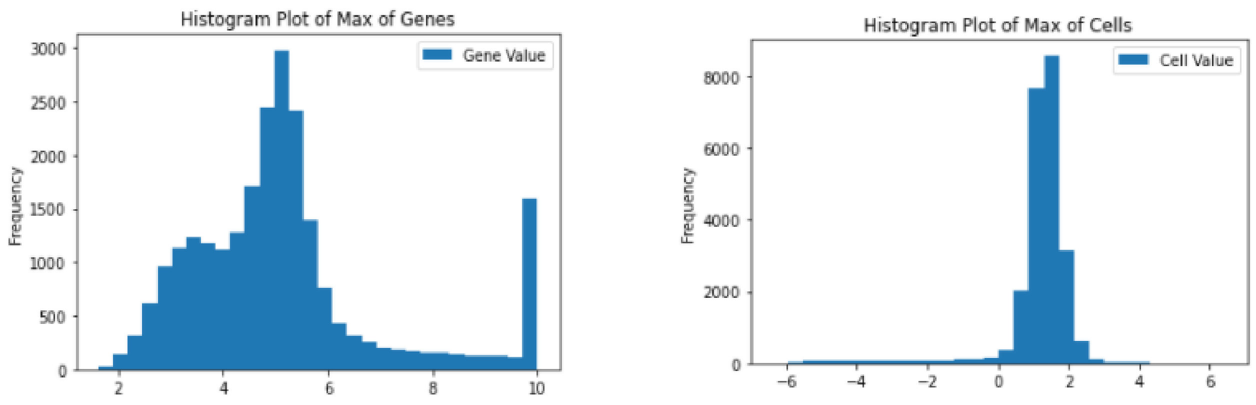

Figure 3.3: Histograms of the Maximum value of gene data (g-) and cell data (c-*) across all train samples*

The plots from Figure 3.2 and 3.3 visualize the minimum and maximum with respect to both gene and cell data in each row. Based on Figure 3.2(left) we can see that the minimum peak lies near -5 for gene data while there is a big spike at -10. This is likely due to the minimum data cap at -10. In Figure 3.2(left), we can see the maximum peak lies near 5 while there is a spike at 10. This is likely due to the maximum capped at 10. Interestingly Figure 3.2(left) and 3.3(left) are nearly negatives of each other.

Figure 3.2(right) shows the feature engineered histogram plot of the minimum of cell data. The peak is near -1.5 and follows a general distribution path except there is a spike at negative. This is likely due to the minimum cap for data. The engineered feature, maximum of cell data Figure 3.3(right) is interesting as it has no spike at the maximum data cap and only has a general distribution with the peak near 1.5. The maximum data cap does not affect the cell data. These features can be useful as some drugs may require gene expression and cell viability to be over or under a certain number and the engineered features minimum and maximum display those numbers respectively.

3.4 Handling the Data Imbalance (Stratified Cross-validation)

The stratification code was adopted from a Github repository[2].

From our previous analysis of the training data, we saw the MoA target labels are highly unbalanced. There are cases where some target labels only have one activation over the entire training set (refer to Fig 2.2). An imbalanced dataset can pose problems for classification tasks. A potential cause for concern is poor predictability for the minority class, in our case, this problem translates to the models not being able to classify target activation. In response to this problem, we used stratified K-Fold cross validation. The idea is that we first do stratified K-fold split, then perform downsampling (without replacement) on the majority label class and upsampling (with replacement) on the minority label class for each fold and then return these stratified folds for training/validation and to search for the set of optimal hyperparameters.

3.5 Model Implementation

The team implemented four different machine learning pipelines in this project and the descriptions of these pipelines are outlined in this section.

The first two pipelines, which are based on logistic regression model and random forest model, are given below in Figure 3.4 and Figure 3.5 respectively.

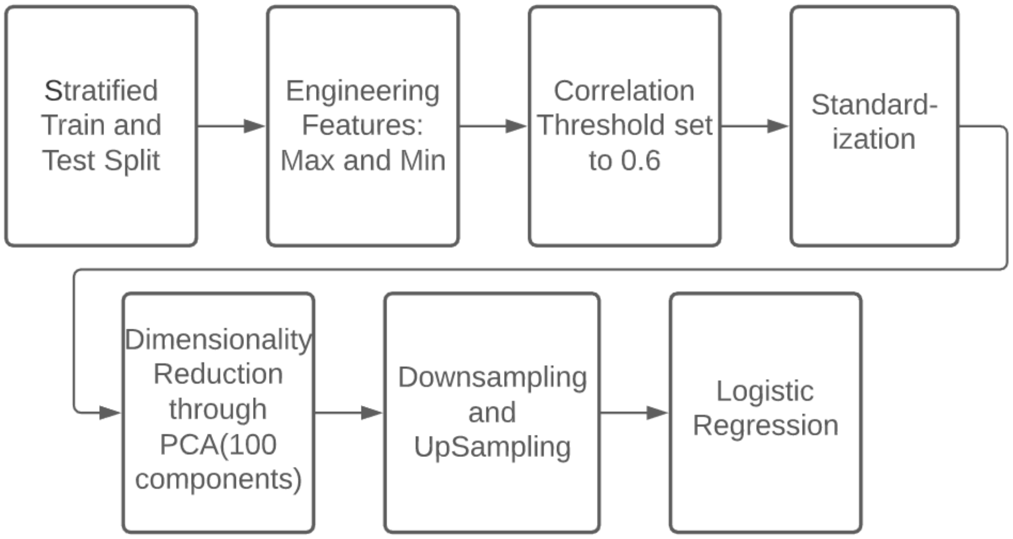
Stratified Train and Test Split
Engineering Features: Max and Min
Correlation Threshold set to 0.6
Standard-ization
Dimensionality Reduction through PCA(100 components)
Downsampling and UpSampling
Logistic Regression

Figure 3.4: Logistic Regression Model pipeline

[2] For more details about the implementation, see here: https://github.com/interviewstreet/spark-stratifier

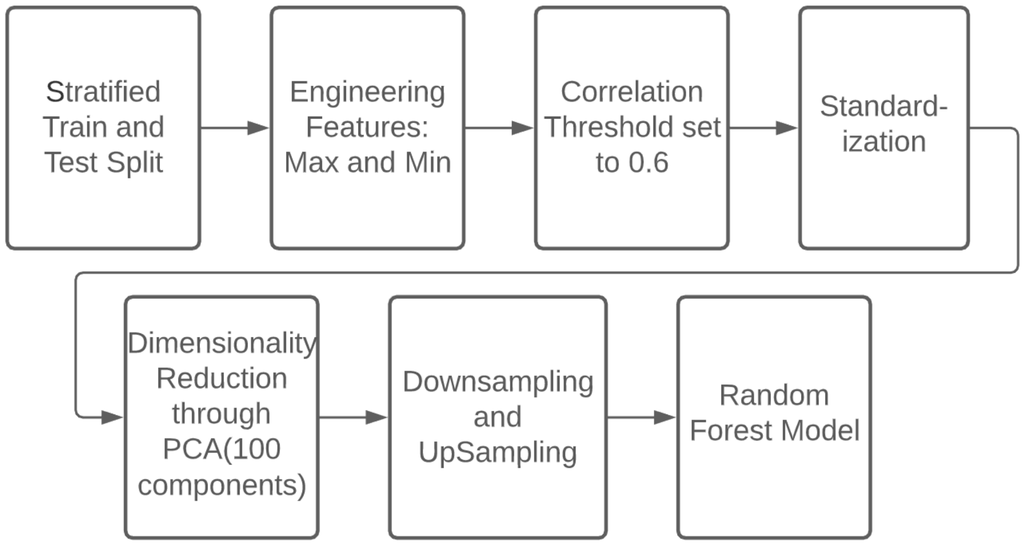
Stratified Train and Test Split
Engineering Features: Max and Min
Correlation Threshold set to 0.6
Standard-ization
Dimensionality Reduction through PCA(100 components)
Downsampling and UpSampling
Random Forest Model

Figure 3.5: Random Forest Model pipeline

In general, the procedures for data-preprocessing, feature engineering and feature transformation are identical for these two pipelines. The descriptions these steps are given below in the same order as they are implemented in the pipeline:

- **Stratified Train-Test Split:** to account for class imbalance. First every combination of target label vector was encoded, and then stratified train-test split (80-20) was done based on that encoded value. In the scenario, there would be samples for all possible different combinations of target labels in both train and test set in 80:20 ratio.
- **Feature Engineering:** maximum and minimum values of gene expressions and cell viability data for each sample were added as new features. The details have been previously provided in Section 3.2.
- **Correlation Analysis**: As shown in Figure 2.2, many of the features in the training data, especially the cell viability features, share high correlations with other features. This step, therefore, drops the features that share high correlation other cell viability and gene expressions features using (threshold = 0.6).
- **Standardization:** on gene expressions and cell viability features to make their distribution unit variance and remove bumps observed in cell viability data.
- **Dimensionality Reduction:** done via PCA to reduce the number of features to 100 (which accounted to around 65% cumulative variance) in order to save computation time
- **Down/Up Sampling:** to account for class imbalance. Samples with label 0 were downsampled (without replacement) by half while samples with label 1 were upsampled (with replacement) to have equal counts of both the classes in the train set.
- **Model implementation:** The transformed and vectoried features, along with the ground-truth labels, are then used to train a logistic regression model and a random forest model. The team selected the logistic regression model because of its wide popularity and that this is the most rudimentary model that can potentially work in many applications. The team also chose a tree-based model, Random Forest, since it may perform better than logistic regression for binary

classification in some scenarios[3]. Note that default hyperparameters were used for these two models, and these two models do not involve hyper-parameter tuning.

In addition to the logistic regression and random forest model that were described above, the team also tried a gradient boosting classifier, which is another tree-based model. If carefully tuned, gradient boosting may perform better than random forest[4]. However, given how the gradient boosting machine approach is designed, such a model may be subject to overfitting especially for an unbalanced dataset. Again, to solve the issue with the unbalanced dataset, the stratified cross-validation approach was applied, and it has been described in detail in Section 3.4. An overview of this pipeline is given in Figure 3.6 below.

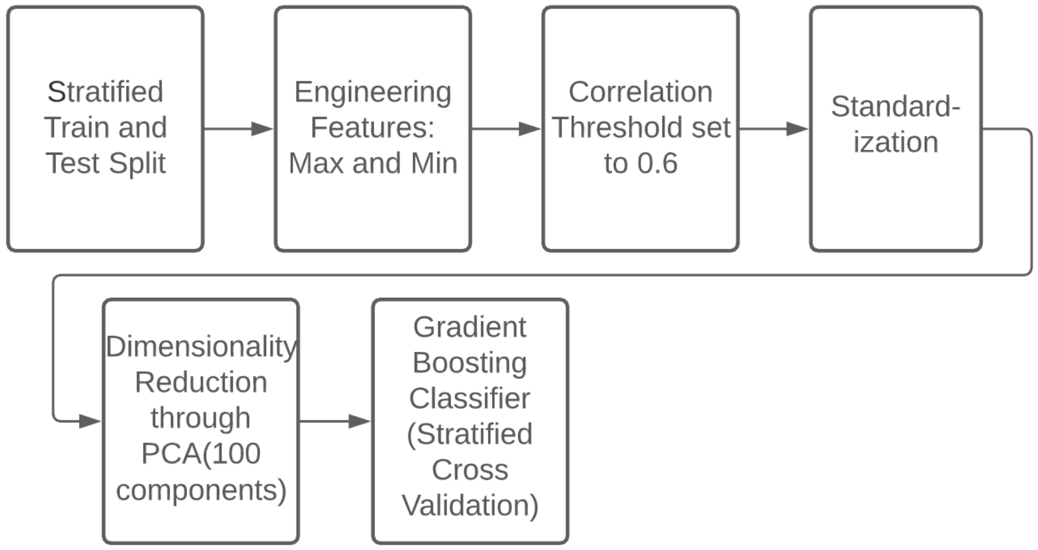
Stratified Train and Test Split
Engineering Features: Max and Min
Correlation Threshold set to 0.6
Standard-ization
Dimensionality Reduction through PCA(100 components)
Gradient Boosting Classifier (Stratified Cross Validation)

Figure 3.6: Gradient Boosted Trees Model pipeline

Note that the first five steps involving feature engineering and transformation (i.e. from stratified train-test split to PCA) are identical to the first two pipelines. Table 3.1 below summarizes the list of hyper-parameters used in the grid-search approach.

Table 3.1: Hyperparameter values for GBT

	Selected values
Maximum number of iterations	{20, 30}
Maximum tree depth	{5, 10}

[3] Kirasich, Kaitlin, Trace Smith, and Bivin Sadler. "Random Forest vs Logistic Regression: Binary Classification for Heterogeneous Datasets." SMU Data Science Review 1.3 (2018): 9.
[4] Posted by Stephanie Glen on July 28, (2019). Decision Tree vs Random Forest vs Gradient Boosting Machines: Explained Simply. Retrieved December 17, 2020, from https://www.datasciencecentral.com/profiles/blogs/decision-tree-vs-random-forest-vs-boosted-trees-explained

Note that the default parameter for the maximum number of iterations is 20 and 5 for the maximum tree depth. A higher number of iterations can decrease the training set error but may also have the potential to overfit. Higer tree depth may also increase the overfitting. Due to constraint in the computational resources, we were unable to further increase the resolutions of the hyperparameter grid. We believe that these value pairs should provide us with strong insights into the underlying trends and additional hyperparameter tuning is subjected to future research.

Finally, the team proposed a challenger model that incorporates both random forest model and logistic model. An overview of the pipeline is given in Figure 3.7 below.

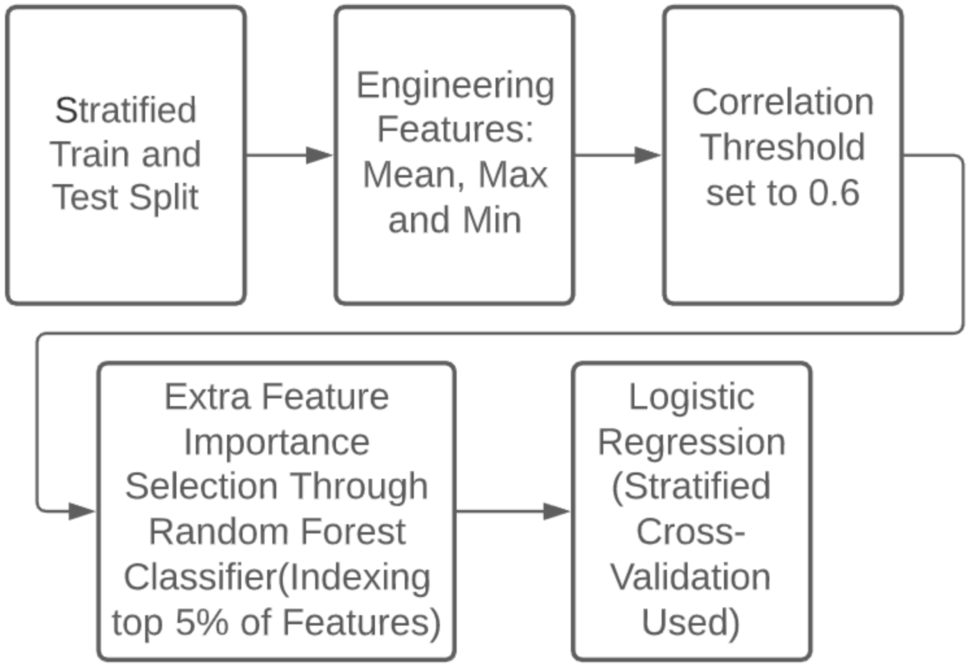
Stratified Train and Test Split
Engineering Features: Mean, Max and Min
Correlation Threshold set to 0.6
Extra Feature Importance Selection Through Random Forest Classifier(Indexing top 5% of Features)
Logistic Regression (Stratified Cross-Validation Used)

Figure 3.7: Challenger Model pipeline

As it can be seen, the first few steps of this pipeline, including stratified train-test split, feature engineering, and drop highly-correlated features. However, in place of PCA with standardization, this pipeline used a random forest classifier, trained on the training dataset, to determine the feature importance of the remaining features after dropping the highly-correlated features. This step was made possible by the fact that tree-based algorithms are able to distinguish which features to be splitted on in order to improve the purity at each level. Thus, by pruning the tree at a particular level, we are able to identify the subset of the most important features[5]. The subset of features were then used to train a logistic regression model, which was also subjected to hyperparameter tuning using stratified cross-validation. We hypothesized that the reduced number of features would boost the prediction capability of the logistic regression model. For the hyperparameter tuning, we used grid search to identify the best combinations of regularization parameters and elastic net parameters. The lists of selected values for these two parameters are provided in Table 3.2.

[5] Albon, C. (2017, December 20). Feature Selection Using Random Forest. Retrieved December 17, 2020, from https://chrisalbon.com/machine_learning/trees_and_forests/feature_selection_using_random_forest/

Table 3.2: Hyperparameter values for RF+Logit

	Selected values
Regularization (λ)	{0.01, 0.1}
Elastic net (α)	{0, 1}[6]

Note that the default parameters for the untuned model are $\lambda = 0$ (no regularization) and $a = 0$. However, as noted in the above table, higher regularization values were used as they may likely reduce overfitting. Again, we were unable to have additional hyperparameters tuned due to the limitation in computational resources.

4. Results

For each of the suggested models (Logistic Regression, GBT, Random Forest) we trained a binary classification model for each scored label in the MoA dataset. The criteria for each of these models was Binary Cross Entropy (BCE) loss. The final score for the overall project and the multilabel classification problem was the mean of the BCE losses across all labels. Figure 4.1-4.4 shows the log-loss for the respective model, presented in the same order as were described in Section 3.5. For the sake of visual clarity, only the first 50 labels are shown for all of these figures. A complete list of the BCE for all of the models, tuned and untuned, is provided in the GitHub repository[7].

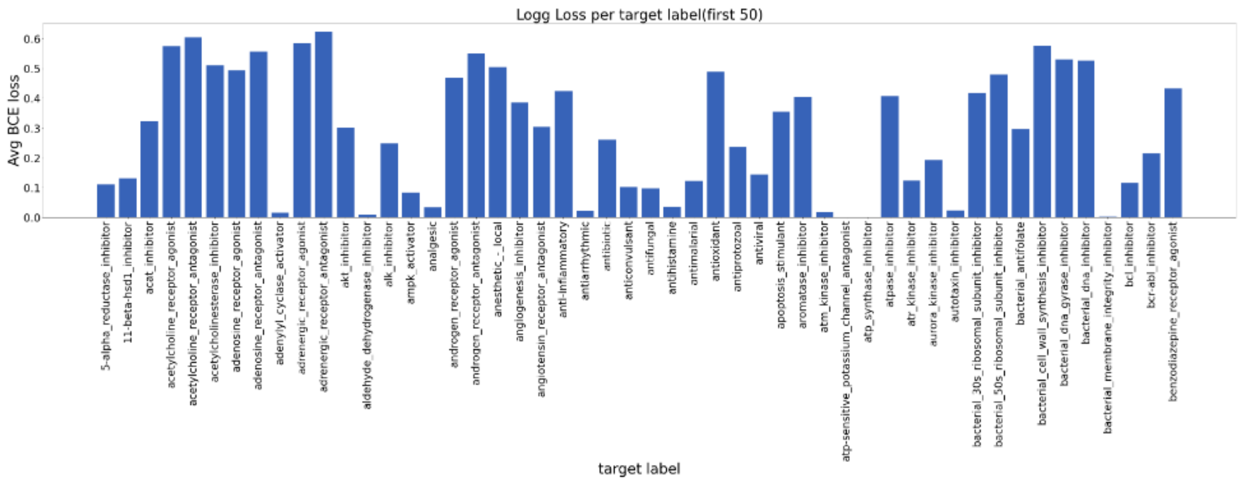

Figure: 4.1: Logistic regression model log-loss for the first 100 labels - Untuned

[6] $\alpha = 0$ for Ridge regression and $\alpha = 1$ for Lasso regression

[7] https://github.com/faraz2023/MIE1628MoA/blob/submission/MBE.csv

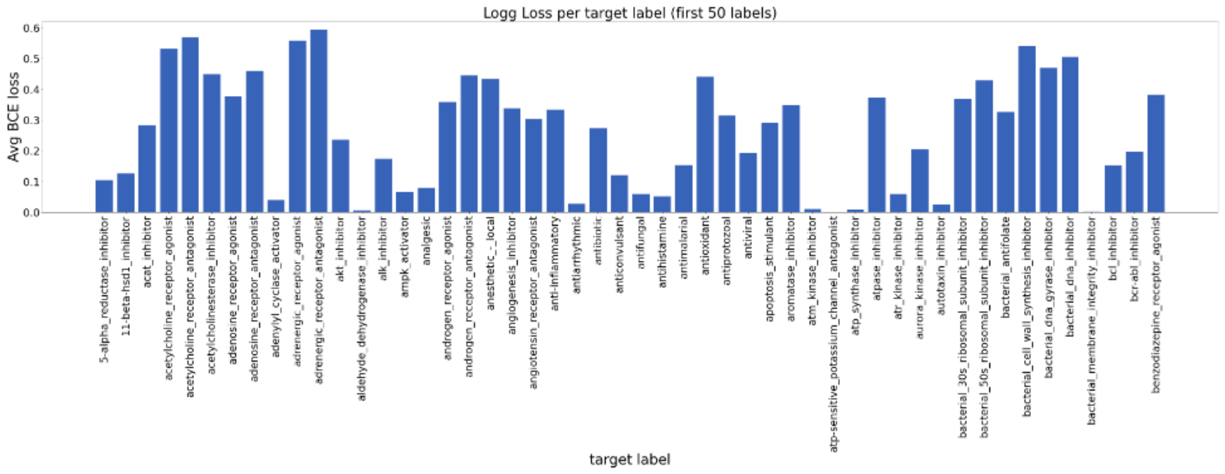

Figure: 4.2: Random forest model log-loss for the first 100 labels - Untuned

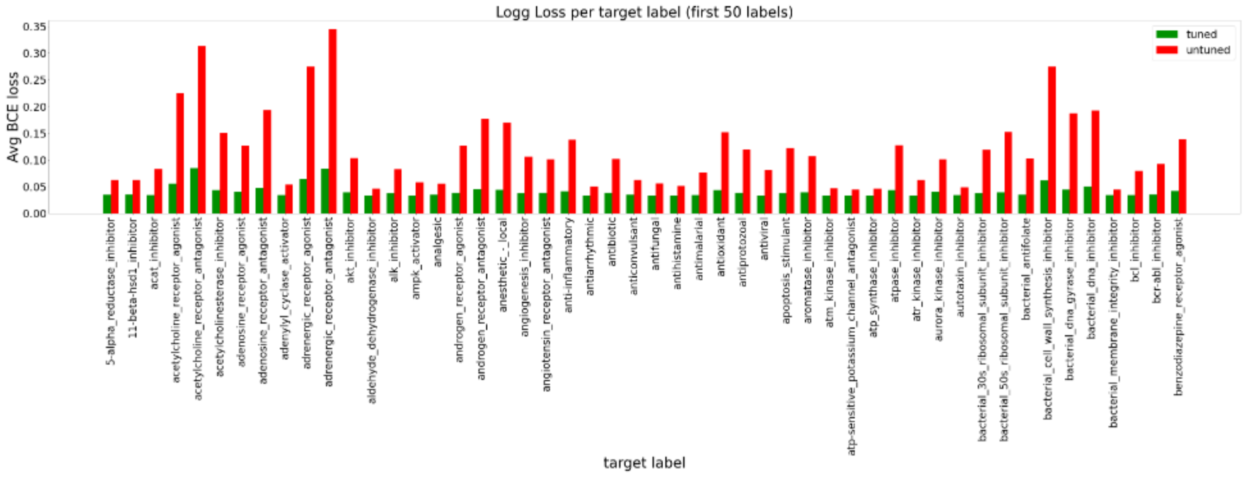

Figure: 4.3: GBT model log-loss for the first 100 labels - Untuned vs Tuned

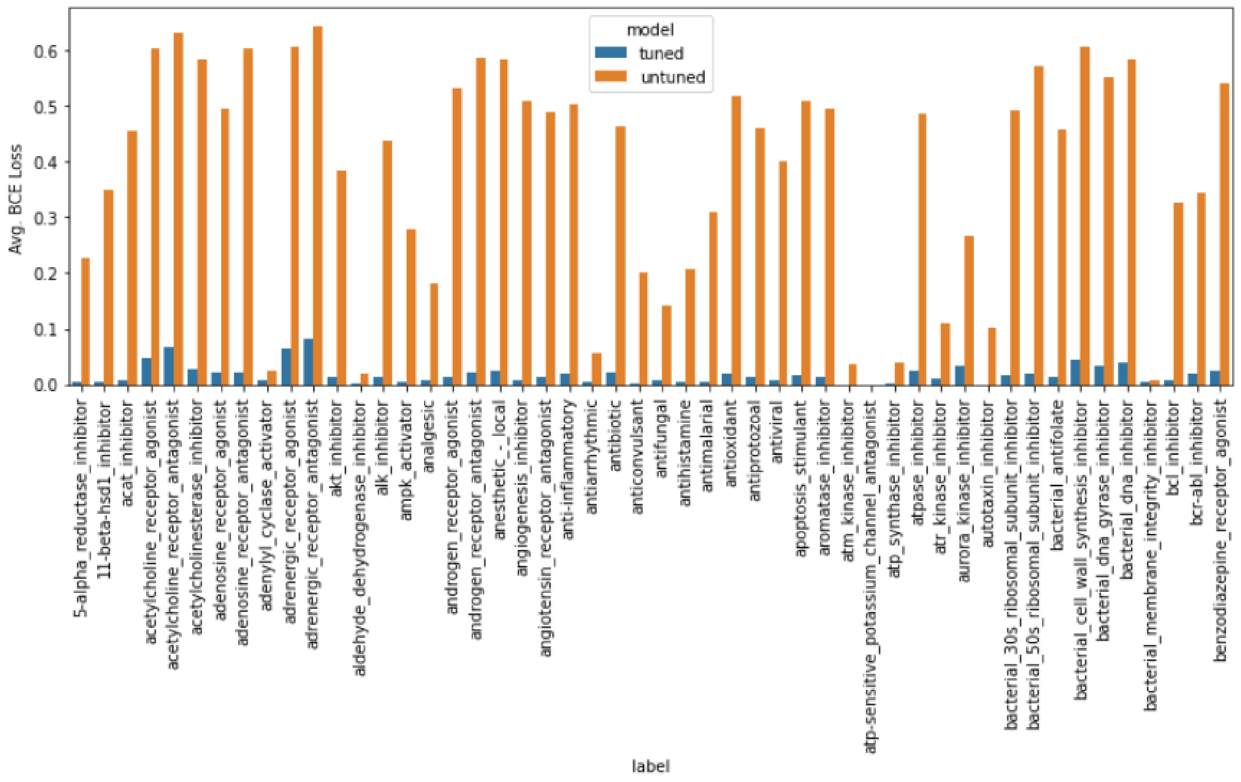

Figure: 4.4: Challenger model log-loss for the first 100 labels - Untuned vs Tuned

Apart from the above visualizations, Table 4.1 summarizes the final scores for each of the four models we implemented and a comparison between the tune and the untuned model where applicable.

Table 4.1: Comparison of Scores across different models

Model		Average BCE losses	Computation Time
Logistic Regression		0.26	~ 0.85 min/label[8]
Random Forest		0.25	~ 1 mins/label[8]
Gradient Boosting Machine	Untuned	0.11	~ 2 mins/label[8]
	Tuned	0.04	~ 5 mins/label[8]
Challenger	Untuned	0.37	~10 mins/label[9]

[8] Intel i7-7500 @ 2.70GHz with 8 GB memory

model	Tuned	0.02	~25 mins/label[9]

In summary, both the logistic regression model and the random forest model received similar mean BCE results. However, the untuned gradient boosting machine offered about 50% in the average BCE value. This result indicated that, by ensembling weak models to reduce loss in a sequential manner, gradient boosting performed better in this specific scenario involving an unbalanced dataset[10]. The tuned gradient boosting machine was able to additionally improve the accuracy, and the distribution of the optimal sets of parameters across all labels are given in Figure 4.5 below.

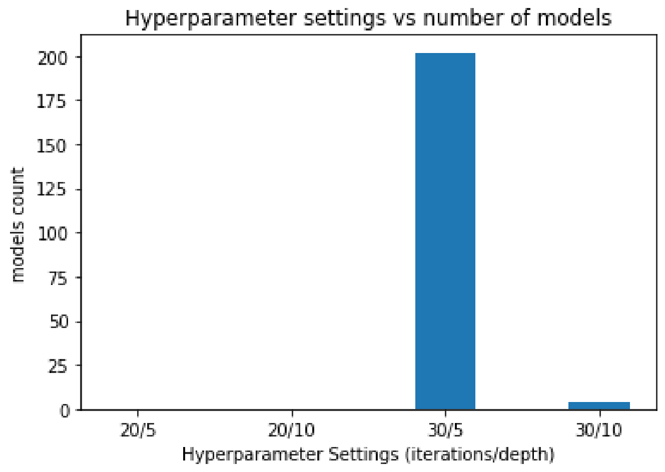

Figure 4.5: Hyperparameter settings vs model of models

According to the above results, having less depth (i.e. 5 as opposed to 10) was preferred when a deeper tree may result in overfitting. At the same time, a higher number of iteration (i.e. 30 as opposed to 20) was preferred in this case when increasing the max iteration allows the GBM to have more trees in the ensemble, thus allowing the GBM to better model the residual.

The tuned challenger model also delivered a significant reduction in log loss when compared to its untuned counterpart. This is expected since the untuned model has no regularization, which is a major contributor to overfitting. To demonstrate further, Figure 4.6 below shows the distribution of the optimal combination of parameters based on the hyperparameter tuning results across all labels.

[9] Intel i7-6700 @ 3.40 GHz with 48.0 GB memory
[10] Posted by Stephanie Glen on July 28, (2019). Decision Tree vs Random Forest vs Gradient Boosting Machines: Explained Simply. Retrieved December 17, 2020, from https://www.datasciencecentral.com/profiles/blogs/decision-tree-vs-random-forest-vs-boosted-trees-explained

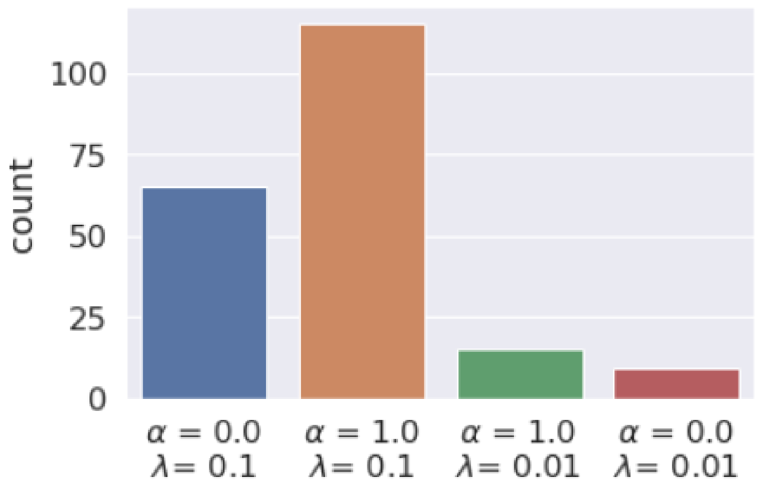

Figure 4.6: Optimal tuned hyperparameter for the challenger model

As expected, the majority of the tuned models perform well when a higher regulzariation is used ($\lambda = 0.1$). At the same time, a higher number of models perform better with Lasso regression. In its essence, Lasso regression can potentially further reduce the number of features by setting the coefficients of less relevant features to zero, which can further reduce the overfitting phenomenon.

5. Discussions

In addition to the steps that the team has attempted and are described above, we also made additional efforts in the domains of feature engineering or improving computational efficiency. However, these attempts were eventually left out of the final models due to the reasons that are discussed below:

- **Parallel training of models**
  Given that this project involves training 206 independent models, our team also examined the possibility of having these models trained in parallel by leveraging RDD capability offered by Spark, as opposed to running these models in sequence using for loops. The team had explored this option[11], but our attempts were unsuccessful thanks to the lack of ability to execute nested operations in Spark[12]. Specifically, Spark's machine learning pipeline itself is already a distributed framework, and thus it cannot be nested inside an RDD map function. Therefore, without relying on other external tools, using a loop for sequential training of each model and only relying on Spark's distributed machine learning framework appeared to be the only option.

- **Applying window function on genome sequence**
  With the understanding that the genome is expressed in sequence, the team also explored the option of transforming genome features into their moving average using window function. Such method of feature transformation has conventionally used in transforming time-series data.

[11] For more details about the implementation, see here: https://github.com/faraz2023/MIE1628MoA/blob/submission/misc/attampt-parallelize-using-map.ipynb
[12] According to a Stack Overflow post: https://stackoverflow.com/questions/43428297/run-ml-algorithm-inside-map-function-in-spark

Given the genome features are given in a row-wise fashion, the window function can thus not be used directly, because it requires a transpose of the dataframe to bring these genome sequences to each column. After applying the window function, we also need to perform yet another matrix transpose to bring back the original matrix structure (i.e. each row for each sample while each column for each feature)[13]. While this idea was initially proposed, we did not eventually adapt it to our final model. This decision was made mainly due to the concern that we were unable to guarantee that the gene features given in the dataset were indeed expressed in sequence, as opposed to being randomized; such information was not provided in the instructions of Kaggle competition.

- **Cluster Analysis for Feature Engineering**
  The team also tried feature engineering using K-Means clustering to signify relationships for the two categorical variables `cp_time` and `cp_dose`[14]. We believe the two features inherently characterize the samples into some groupings in a high dimensional space. To see if there was anything distinguishable, we tried to visualize `cp_time` and `cp_dose` with other genomic features in the dataset using 3D plots, however that did not reveal much information. We then tried KMeans clustering with different hyperparameters on a subset of the data and discovered many samples are characterized by different clusters based on the values of `cp_time` and `cp_dose`.

Additionally, due to the limitation in the computation resource, we were unable to further increase the value ranges of hyperparameter or increase resolution. Future exploitation of the model may include further increasing the regularization parameter to additionally reduce overfitting. Alternatively, heuristic hyperparameter tuning techniques may also be explored. Last but not the least, if the platform option is not constrained to Spark, other advanced machine learning packages may also be used. For instance, deep learning algorithms typically appear among the top solutions in the leaderboard of the competition. However, to date, Spark has yet implemented deep learning in the platform's machine learning package.

Our original approach consisted of feature reduction through PCA and then training a logistic regression model, a random forest model and a GBT model. The challenger approach, however, used random forest and feature importance for feature selection and then trained a logistic regression based on the remaining features. As for a comparison between the trained models, here we mention the pros and cons for each machine learning pipeline and model presented for this project:

- Logistic regression pipeline
  - Pros: short training time because it requires less passes
  - Cons: linear model but data not always linearly separable and hence low accuracy,, non-interpretability of features due to PCA, loss of information due to dimensionality reduction

[13] For more details about the implementation, see here: https://github.com/faraz2023/MIE1628MoA/blob/submission/misc/attampt-apply-window-function.ipynb

[14] For more details about the implementation, see here: https://github.com/faraz2023/MIE1628MoA/blob/submission/misc/attampt-k-mean-clustering-feat-engineering.ipynb

- Random forest pipeline
    - Pros: non-linear model; it can be highly scalable due to bootstrapping, low variance
    - Cons: non-interpretability of features due to PCA, loss of information due to dimensionality reduction
- GBT pipeline
    - Pros: high accuracy(due to ensemble of weak learners) when properly tuned, low bias
    - Cons: non-interpretability of features due to PCA, loss of information due to dimensionality reduction, large number of hyperparameters to tune, long training time
- Challenger model pipeline (random forest and logistic regression)
    - Pros: high accuracy when properly tuned; feature interpretability because it does not involve PCA transformation
    - Cons: long training time because it requires sequentially training of the random forest and logistic regression

Conclusion

We have developed several machine learning pipelines to classify mechanisms of activation based on cells and genes data using binary cross-entropy as a scoring metric. During data exploration we saw the data is mostly normal with some abnormalities, this encouraged us to engineer new features to describe the row-wise statistics of the cell and gene data. We have also used different forms of feature selection technique in each of the machine learning pipelines. Some techniques that were used are correlation analysis, feature importance, and PCA. These were implemented to remove insignificant features and reduce the dimensions of the dataset. Out of all the proposed models, our best scoring model pipeline uses a random forest for feature selection and fits logistic regression models on the important features for each target label.. This model returned a BCE score of 0.02 with hyperparameter tuning and 0.37 without tuning.