The Mechanisms of Action (MoA) Prediction Problem

Ali Alghaithi

9/22/2020

1: Introduction

Background, Significance ,and Research question(s)

The Mechanisms of Action (MoA) Prediction is a competition hosted in Kaggle by the Laboratory for Innovation Science at Harvard (LISH). This challenge was developed to advance drug development within improvements to MoA prediction algorithms. It is designed to help to advance the algorithm that classifies drugs based on their biological activity. With all the great technologies around us today, the drug discovery process has evolved a lot and building a more potent targeted model based on an understanding of the underlying biological mechanism of disease in biology. Biological activity of a given molecule, scientists assign a label referred to as mechanism-of-action or MoA for short. Pharmaceutical drug development intends to classify proteins connected to a particular disease and then produce molecules to target those proteins. The MoA of a molecule encodes its biological activity. This dataset describes the responses of 100 different types of human cells to various drugs. Those response patterns will be used to classify the MoA response.

In this project, the approach is to treat a sample of human cells with the drug and then analyze the cellular responses with algorithms that seek for the relationship to known patterns in large genomic databases, such as libraries of gene expression or cell viability patterns of drugs with known MoAs. In summary, the research question this project is trying to answer is " Will using gene expression data and cell viability data can improve drog development to predict multiple targets of the Mechanism of Action (MoA) response(s) of different samples?"

2: Data

2.1: Source of data or data collection

The dataset that will be used is a combination of gene expression and cell viability data. The data have been collected using the newest technologies measures concurrently human cells' responses to drugs of 100 different cell types. The data has MoA annotations for around 5,000 drugs in this dataset. The provided data is presented as a train and test files. For the training predictors, we are given two files, such as the training predictors (train_features.csv) and the targets (train_targets_scored.csv). The rows for these files represent specific treatment. We are using the training predictors to build a model to predict the train_targets_scored.csv class probabilities for the test file test_features.csv. The (train_targets_nonscored.csv) is an optional file that we can use to help with analysis but not needed to be predicted.

2.2: Overview of raw data and Variable introduction

Train Data

After exploring the data from the bellow table we see that sig_id is the unique primary key of the sample. In total, the train data has almost 900 columns. The variables that have the term "g-" encode gene expression data (there are 772 of those). on the other hand, the term "c-" (100 in total) shows cell viability data. In addition, the data also has 3 "cp_" features where the cp_type variable indicates the sample treatment, while cp_time and cp_dose encode the duration and dosage of the treatment. The bellow table only shows the first 4 rows and the first 6 columns of the train data with an additional to column number 850 and 860.

```
## # A tibble: 4 x 8
##   sig_id       cp_type cp_time cp_dose   `g-0`  `g-1` `c-73` `c-83`
##   <chr>        <chr>     <dbl> <chr>     <dbl>  <dbl>  <dbl>  <dbl>
## 1 id_000644bb2 trt_cp       24 D1       1.06    0.558  0.959  0.129
## 2 id_000779bfc trt_cp       72 D1       0.0743  0.409  0.593  0.340
## 3 id_000a6266a trt_cp       48 D1       0.628   0.582  0.136 -0.627
## 4 id_0015fd391 trt_cp       48 D1      -0.514  -0.249 -1.61  -1.16
```

```
## [1] 23814
```

```
## [1] 876
```

```
## [1] 772
```

```
## [1] 100
```

Targets Data

The Targets data has 23814 rows and 207 columns. The sig_id variable the unique primary key of the sample. On the other hand, the other column represents the target response with different binary outputs. In summary, this data is the binary MoA targets that are scored. The bellow table only shows the first 3 rows and first 4 columns.

```
## # A tibble: 3 x 4
##   sig_id       `5-alpha_reductase_inhibit~ `11-beta-hsd1_inhibit~ acat_inhibitor
##   <chr>                              <dbl>                  <dbl>          <dbl>
## 1 id_000644bb2                           0                      0              0
## 2 id_000779bfc                           0                      0              0
## 3 id_000a6266a                           0                      0              0
```

Targets_non Data

In this optional data, we find that the additional non-scored targets contain about 400 classes which are almost double than our target data. it odes provide a lot of information. the only disadvantage of this knowledge that our train data not much connected to the classes we see in this dataset. the bellow table only shows the first 4 rows and the first 5 columns of the non-scored targets data.

```
## # A tibble: 4 x 4
##   sig_id       abc_transporter_expression_enhancer abl_inhibitor ace_inhibitor
##   <chr>                                      <dbl>         <dbl>         <dbl>
## 1 id_000644bb2                                   0             0             0
## 2 id_000779bfc                                   0             0             0
## 3 id_000a6266a                                   0             0             0
## 4 id_0015fd391                                   0             0             0
```

Test Data

The below table only shows the first 4 rows and first 6 columns of the test data with an additional to column number 850 and 860. The test data contains the same features as the train data and has about 4000 rows compared to the 24000 rows of the training data. 4000 samples vs 200 targets is not a very high ratio.

```
## # A tibble: 4 x 8
##   sig_id       cp_type     cp_time cp_dose  `g-0`  `g-1` `c-73` `c-83`
##   <chr>        <chr>         <dbl> <chr>    <dbl>  <dbl>  <dbl>  <dbl>
## 1 id_0004d9e33 trt_cp           24 D1      -0.546  0.131 -0.415 -0.913
## 2 id_001897cda trt_cp           72 D1      -0.183  0.232 -1.35  -0.584
## 3 id_002429b5b ctl_vehicle      24 D1       0.185 -0.140  0.709 -0.462
## 4 id_00276f245 trt_cp           24 D2       0.483  0.196  0.231 -0.426
```

Data Quality

After checking for quality, the sig_id values are indeed unique in the training data. In addition, we also find that Train and Target sig_id values do not match.

2.3: Missing data and imputation methods

- There are not any missing values in the Train data as well as the Targets data

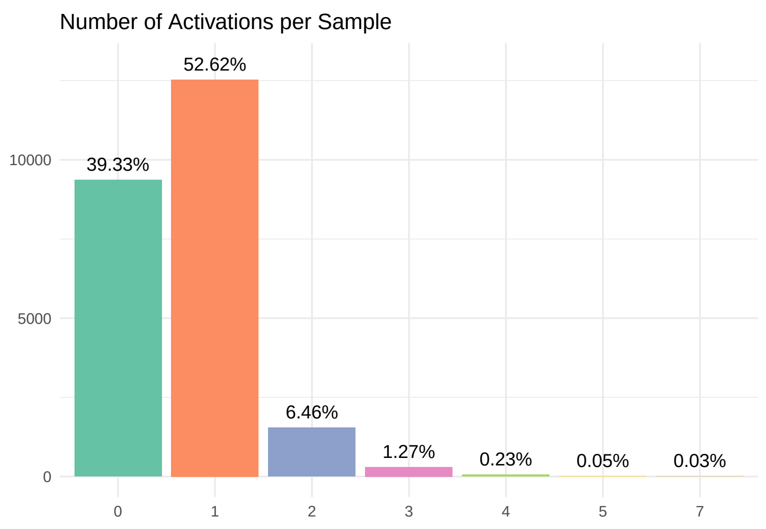
2.4: Evaluation

For every sig_id you will be predicting the probability that the sample had a positive response for each target. For N rows and M targets, you will be making $N\ddot{O}M$ predictions. Submissions are scored by the log loss:

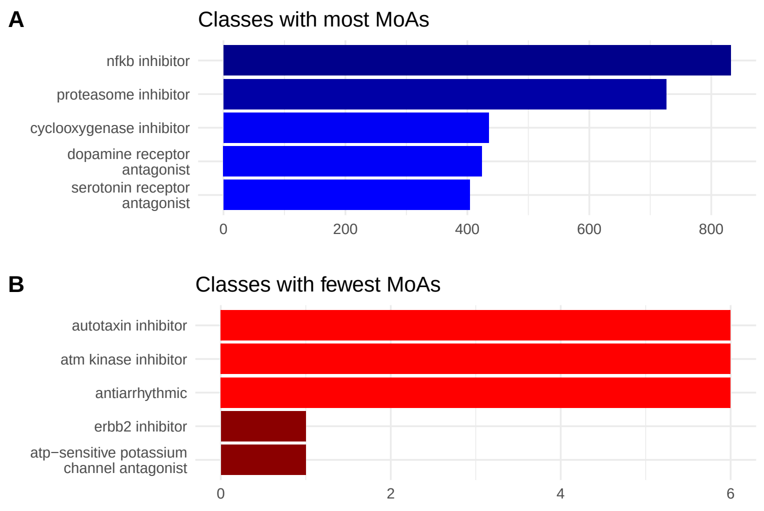
- Score $= -\frac{1}{M}\sum_{m=1}^{M}\frac{1}{N}\sum_{i=1}^{N}\left[y_{i,m}\log(\hat{y}_{i,m}) + (1-y_{i,m})\log(1-\hat{y}_{i,m})\right]$
- N is the number of sig_id observations in the test data $i = 1, \ldots, N$
- M is the number of scored MoA targets $m = 1, \ldots, M$
- $\hat{y}_{i,m}$ is the predicted probability of a positive MoA response for a sig_id
- $y_{i,m}$ is the ground truth, 1 for a positive response, 0 otherwise
- $log()$ is the natural (base e) logarithm

2.5: Data re-structuring (data wrangling)

Since the challenge is a multi-label classification problem. Since the description from Kaggle that drug samples actually can have multiple MoA's. Data re-structuring is needed based on what way to approach this problem. it is very useful to check the distribution of how many target classes can be active at once.

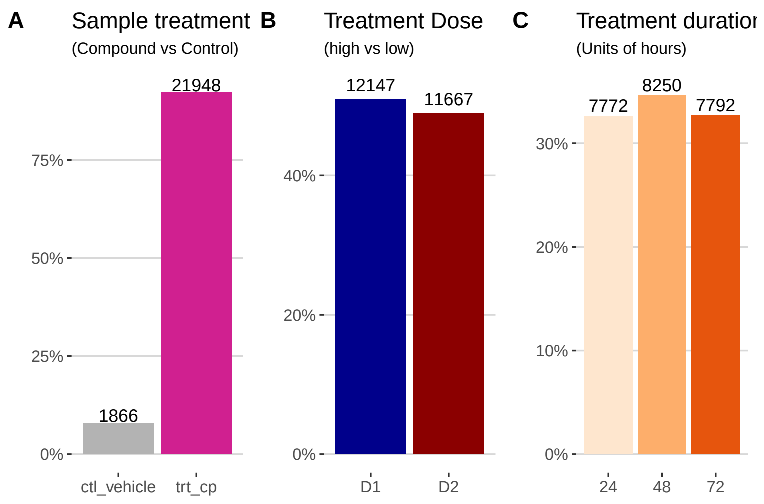


From the above plot, we can see that 39% of training samples have no MoA annotations at all. Almost 52% of samples have exactly 1 MoA annotation. In addition, 5% of cases have 2 MoAs, and the other cases are rarer. We might need to check more about what we have been seeing so far. The next plots should explain more and give us more insight.

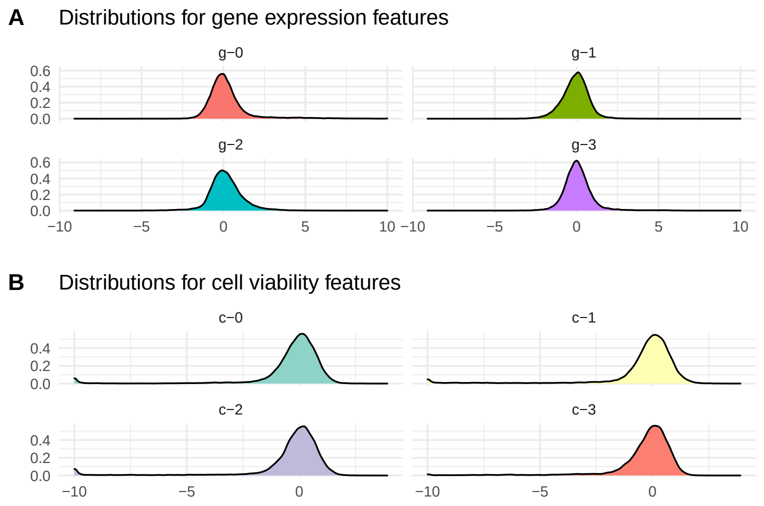


The rare case of only 1 positive MoA is measured for the two dark-red cases in plot B. The top-scoring classes can be seen in plot A. Inhibitors and antagonists are both inside both plots.

Now, we also have a look at the Train data and investigate the variables.



To understand how the sample was treated in terms of dose, duration, and whether it was a "real" treatment or control. The bellow plots can more explain the data using the ggplot2 library. According to the plots, most of the treatments are compound treatments and less of control perturbation treatments. The treatment dose variable is either D1(high) of D2(Low), and both responses are evenly balanced. For the treatment duration, we can also say all the responses are balanced.

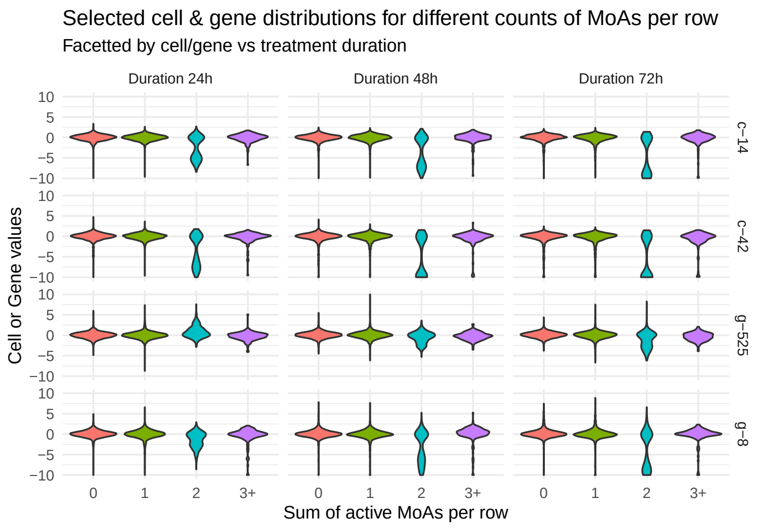


- Plot A: The variables with the term gene are labeled from "g-0" to "g-771," and they have numeric values. From the above plot where we are looking at the first 4 gene features where we see their distributions to be normal.
- Plot B: Similar to plot A, the cell viability features are unknown, labeled from "c-0" to "c-99"; 100 features. The distributions look normal as well.

However, we might need to check more and see why both distributions are very flat at both ends. Also, we are seeing that a small increase at -10 in plot B.

In conclusion, we are not sure how we need to instruct the data yet but we do have a great insight into our data. It is important to also investigate the interaction between predictors to see if we can find any correlations that might lead us to reduce some of the variables.

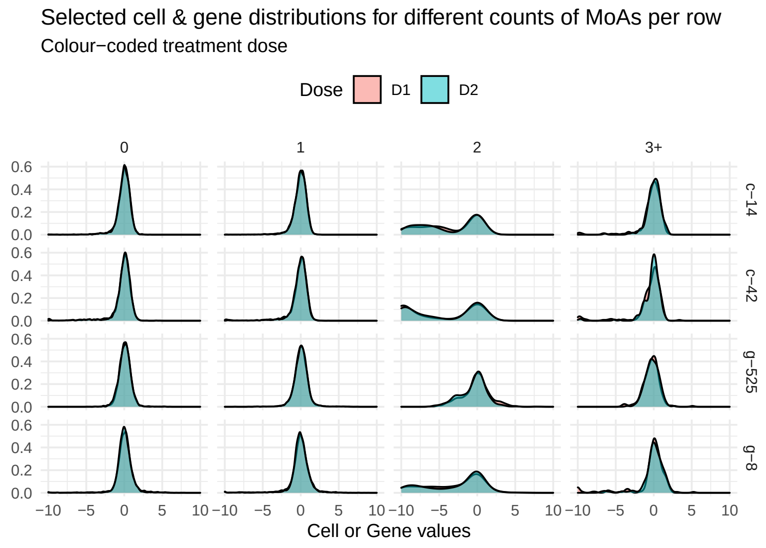
We must understand the relationship between predictor features and the target classes. The approach is to look at the interactions between targets and predictors in terms of their corresponding distributions.N ext, we'll check whether the number of MoAs in a row influences the cell or gene distributions. Here, for the sake of sample size, we lump together the instances with 3 or more MoAs into the group "3+". We're only looking at compound treatments; e.g. no control rows. For some visual variety, we'll be using violin plots (i.e. vertical, mirrored density plots):



There is a difference between those cases with 0 or 1 vs. 2 or more MoAs. We see that the 2 MoA category accounts for most of the negative tail in the cell features. We also see that the "3+" group looks more similar to the "0" and "1" group. Also, we note that the "3+" group has a tiny sample size in this data.

There are some slight differences between the cp_time duration for the "2" and "3+" groups. Both the "0" and the "1" group look pretty stable across the board.

Next, we can check for the treatment doses to see if we can see any relationships.



Almost all the distributions show perfect overlap, and there are not any differences. Again we see how "MoA = 2 is different from the others.

2.6: New variable creation (optional)

Agina as we are dealing with a multilabel classification problem with many predictors, we for sure have to reduce the number of our variables first and then creat any necessary variable if needed.

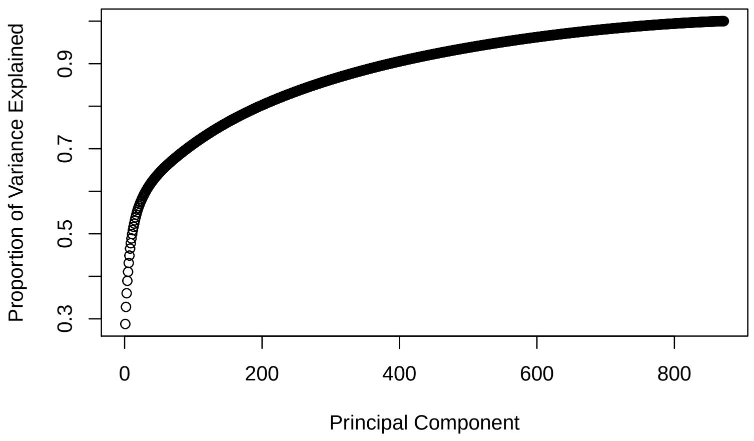
2.7: Overview of cleaned data (e.g. screenshot)

The data is well cleaned, and that would save us a lot of time. For the moment, we would need to do some preprocessing to get the data in the right shape for the model we are planning to use. For example, we join the targets for the training data.

Methods:

Principal Component Analysis (PCA)

The PCA is a dimensionality reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set. PCA has been applied to the train data set that only includes gene and cell expression which is about 871 features. In the next step, we have the proportion of variance explained by each component which we will need to decide the number of components. We calculated that the first seven components explain most of the variance, however, for a more visual approach, we plot the explained variance on a line graph. Here we plot the ratio of variance explained by each component using a line graph. This PCA chart helps us to decide the number of principal components to be taken for the modeling algorithm. During the Final submission to Kaggle, we decided to choose the first 200 since it explained almost 80% of the variance. For compression, we also tried 50 PCAs to compare.



mlr-package: Machine Learning in R

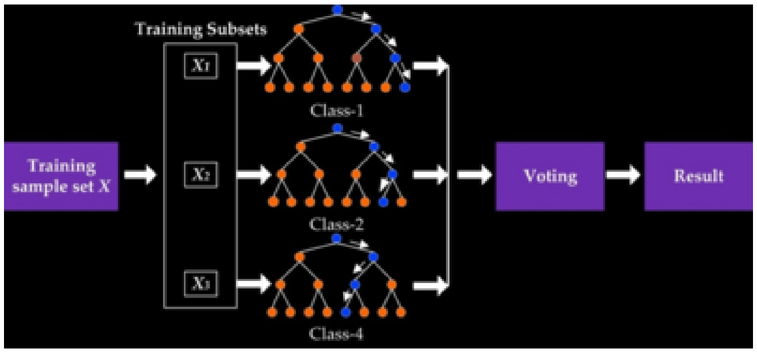
Multilabel classification is a classification problem where multiple target labels can be assigned to each observation instead of only one like in multiclass classification. we decided to use the mlr pakage to help dealing with the multilabel classification task.

Two different modeling approaches exist for multilabel classification.

- Problem transformation methods try to transform the multilabel classification into binary or multiclass classification problems.
- Algorithm adaptation methods adapt multiclass algorithms so they can be applied directly to the problem.

In multilabel classification each object can belong to more than one category at the same time. To be able to use makeMultilabelTask function from mlr, The target columns should be logical vectors that indicate which class labels are present. The names of the target columns are taken as class labels and need to be passed to the target argument of makeMultilabelTask.

Scine we decied to use the algorithm adaptation methods, one of the available algorithm adaptation methods in R are the multivariate random forest in the randomForestSRC package. The folloing plot shows how the randomforest in this case would look like.



randomForestSRC: Fast Unified Random Forests for Survival, Regression, and Classification (RF-SRC) In our spical senario we have classification, the y-outcomes associated with an individual i can be denoted by $\mathbf{Y}_i = ((Y_{i,1}, \ldots, Y_{i,r})$ where $Y_{i,j}$ is real or categorical for each $Y_{i,j}$. In our case, $Y_{i,j}$ is categorical for all $j \in \{1, \ldots, r\}$ the model is a multivariate classification forest. In this case the split rule statistic is a composite of r split rule statistics based on Weighted Gini Index Splitting. No normalization is necessary since the j-specific statistics represent an impurity measure which is invariant with respect to scale.

Weighted Gini index splitting

Suppose the proposed split for the root node is of the form $x <= c$ and $x > c$ for a continuous x variable x, and a split value of c. The impurity of the node is defined as

$$\phi(\mathbf{p}) = \sum_{j=1}^{J} p_j(1-p_j) = 1 - \sum_{j=1}^{J} p_j^2.$$

The Gini index for a split c on x is

$$\theta(x,c) = \frac{n_l}{n}\phi(\mathbf{p}_l) + \frac{n_r}{n}\phi(\mathbf{p}_r),$$

where, as before, the subscripts l and r indicate left and right daughter node membership respectively, and nl and nr are the number of cases in the daughters such that (n=nl+nr). The goal is to find x and c to minimize

$$\theta(x,c) = \frac{n_l}{n}\left(1 - \sum_{j=1}^{J}\left(\frac{n_{j,l}}{n_l}\right)^2\right) + \frac{n_r}{n}\left(1 - \sum_{j=1}^{J}\left(\frac{n_{j,r}}{n_r}\right)^2\right), \tag{1}$$

where nj,l and nj,r are the number of cases of class j in the left and right daughter, respectively, such that (nj=nj,l+nj,r). This is equivalent to maximizing

$$\theta^*(x,c) = \frac{1}{n}\sum_{j=1}^{J}\frac{n_{j,l}^2}{n_l} + \frac{1}{n}\sum_{j=1}^{J}\frac{(n_j - n_{j,l})^2}{n - n_l}, \tag{2}$$

Terminal node estimators

- The predicted value for a terminal node h are the class proportions in the node. Let individual i have feature Xi. and reside in terminal node h. Let n_j equal the number of bootstrap cases of class j in the node. Then the proportion for the j^{th} class is given by

$$\hat{p}_{j,h} = \frac{1}{n_h} \sum_{\mathbf{X}_i \in h} I\{Y_i = j\}. \tag{3}$$

- To produce the OOB predicted value for individual i this is averaged over all $ntree$ trees. Let $\hat{p}_{j,b}(Xi)$ denote the predicted value for the j^{th} proportion for tree b in $1, ..., ntree$. As before, $I_{i,b} = 1$ if individual i is an OOB individual for b in $1, ..., ntree$, otherwise set $I_{i,b} = 0$. The OOB predicted value for the j^{th} proportion for individual i is

$$\hat{p}^*_{e,j}(\mathbf{X}_i) = \frac{\sum_{b=1}^{\mathtt{ntree}} I_{i,b}\hat{p}_{j,b}(\mathbf{X}_i)}{\sum_{b=1}^{\mathtt{tree}} I_{i,b}}.$$

Prediction error

- There is no prediction error implemented in this scenario

Model result

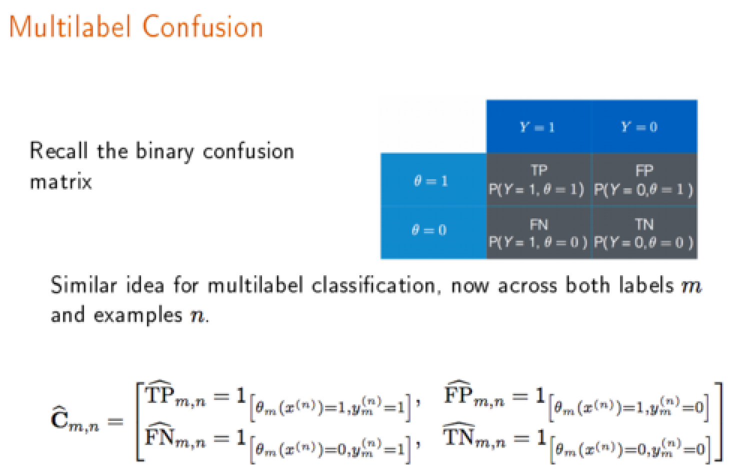
- The results are obtained by using a random forest algorithm. It is only considered the 200 Cpmpnana obtained from the PCA. For computation time, we did not consider does treatment, duration, and. We also n ly used 50% of the given trained data set. The reason s that this is a code competition that requires the code to run less than 9 hours. One submission to Kaggle takes 18 hours almost since Kaggle will need to run the code twice. The size of the data made it difficult to try other models and use sampling. However, this approach's result scored 0.03680 in the public leaderboard and 0.02923 on the private leaderboard. The private leaderboard is calculated with approximately 80% of the test data. Note: the actual submitted predicted probabilities are replaced with $max(min(p, 1 - 10^{-15}), 10^{-15})$ in Kaggle compared to the valuation method mentioned above. A smaller log loss is better. This simple approach was better than 713 teams, and it passed over 376 teams on the private leaderboard. Home wvwer after trying 50 PCAs instewad of 200, the molde performed less, which resulted in a score of 0.04770 in the public leaderboard and 0.055230 on the private leaderboard. The decline was caused by having less explanation of the variance from the data.

Current Model Summary

- Learner Summary

```
## Learner multilabel.randomForestSRC from package randomForestSRC
## Type: multilabel
## Name: Random Forest; Short name: rfsrc
## Class: multilabel.randomForestSRC
## Properties: missings,numerics,factors,prob,weights
## Predict-Type: prob
## Hyperparameters: na.action=na.impute
```

- Model Summary
- Multilabel Confusion Matrix



Multilabel Confusion

Recall the binary confusion matrix

	$Y=1$	$Y=0$
$\theta=1$	TP $P(Y=1,\theta=1)$	FP $P(Y=0,\theta=1)$
$\theta=0$	FN $P(Y=1,\theta=0)$	TN $P(Y=0,\theta=0)$

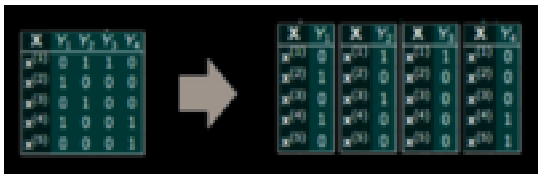
Similar idea for multilabel classification, now across both labels m and examples n.

$$\widehat{\mathbf{C}}_{m,n} = \begin{bmatrix} \widehat{\mathrm{TP}}_{m,n} = \mathbf{1}_{\left[\theta_m(x^{(n)})=1, y_m^{(n)}=1\right]}, & \widehat{\mathrm{FP}}_{m,n} = \mathbf{1}_{\left[\theta_m(x^{(n)})=1, y_m^{(n)}=0\right]} \\ \widehat{\mathrm{FN}}_{m,n} = \mathbf{1}_{\left[\theta_m(x^{(n)})=0, y_m^{(n)}=1\right]}, & \widehat{\mathrm{TN}}_{m,n} = \mathbf{1}_{\left[\theta_m(x^{(n)})=0, y_m^{(n)}=0\right]} \end{bmatrix}$$

```
##                          Sample size: 24
##             Frequency of class labels: NA, 24
##                      Number of trees: 1000
##             Forest terminal node size: 1
##         Average no. of terminal nodes: 9.292
## No. of variables tried at each split: 15
##                Total no. of variables: 200
##                Total no. of responses: 206
##           User has requested response: target_5.alpha_reductase_inhibitor
##         Resampling used to grow trees: swor
##      Resample size used to grow trees: 15
##                              Analysis: mRF-C
##                                Family: class+
##                        Splitting rule: mv.gini *random*
##         Number of random split points: 10
##                Normalized brier score: 0
##                                   AUC: NaN
##                            Error rate: 0, NA, 0
##
## Confusion matrix:
##
##           predicted
##   observed TRUE FALSE class.error
##      TRUE    0     0         NaN
##      FALSE   0    24           0
##
##  Overall error rate: 0%
```

Second appraches: Binary Relevance

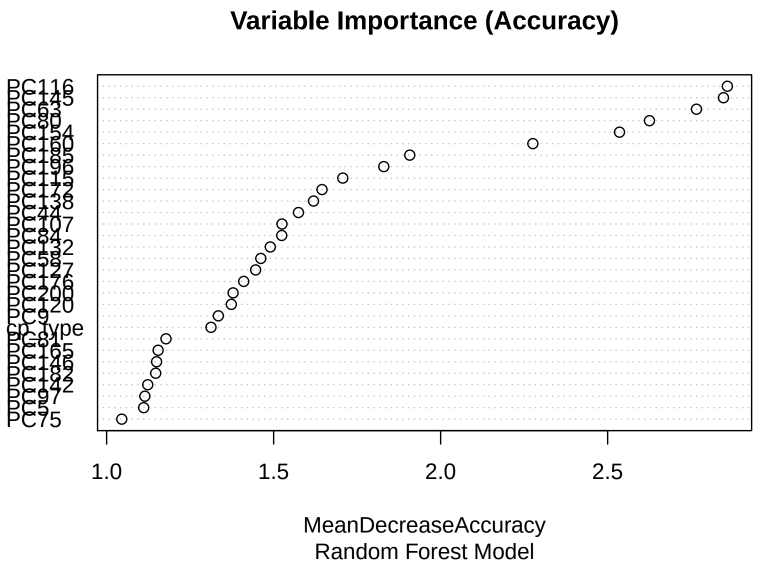
Binary relevance is arguably the most natural solution for learning from multi-label examples. It works by decomposing the multi-label learning task into several independent binary learning tasks (one per class label). In binary relevance, this problem is broken into 206 different single class classification problems. The figure below shows a short example how this method can be applied.



The bellow model summary from the random forest with adjusted salmpesize parameter for the unbalanced issue with the data. This model summary only for one of the labels of this data. Following the approach, there will be 206 models that will be needed.

```
##
## Call:
##  randomForest(formula = as.factor(the_train[, 204 + i]) ~ ., data = the_train[,      c(2:204)], ntre
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 14
##
##         OOB estimate of  error rate: 0.08%
## Confusion matrix:
##       0 1  class.error
## 0 23795 2 8.404421e-05
## 1    17 0 1.000000e+00
```

Dotchart of variable importance as measured by a Random Forest as mentioned above.



Conculation on the Binary Relevance,

from the simlistcy side , this apprache made it simple to modle the data. Hoever, this apprache resulted in pooor performance comparing to the apdatctibe method.it resulted in a score of 0.082234 in the public leaderboard and 0.089670 on the private leaderboard.

Summary

In general, multilabel classification problems are very challenging, and it is messy when the size of the data is big. The preparation methods to solve this type of problem mentioned in this report, we introduced two main concepts of multilabel classification problems. The report covers the Adapted Algorithm and Binary Relevance.

The Interesting findings from both methods are that for this specific problem, the Adapted Algorithm method performs better than the binary relevance method. Also, the more PCAs we include, the better model we have. However, dealing with these classification problems requires more computing power. The binary relevance method did not fit this data because of the way this problem was set to be solved. We find that it is important that we build a model that takes consideration of all the labels at once to reduce the error rate.

The further work of this project can be looped around finding new ways to reduce the variables and rows of the data. The use of cross-validation and other sampling techniques can help reduce the computing time and increase accuracy over all the labels. Future work could also take consideration of some of the variable that was not included in this project. In summary, working with multilabel classification problems can be very beneficial to learn more machine learning technics and what steps to take towered solving new challenges.

Citations

Deep Learning with Keras & TensorFlow in R | Multilayer Perceptron for Multiclass Classification

- Talk: https://www.youtube.com/watch?v=hd81EH1g1bE

www.kaggle.com recipe

- https://www.kaggle.com/kailex/moa-transfer-recipe-with-smoothing

mlr Tutorial

- Paper: https://arxiv.org/pdf/1609.06146.pdf

reducing

- https://www.datavedas.com/dimensionality-reduction-in-r/#:~:text=There%20are%20many%20modeling,Component%

Paper: Extreme Classification: A New Paradigm for Ranking & Recommendation

- Paper:http://manikvarma.org/pubs/agrawal13.pdf
- Talk: https://www.youtube.com/watch?v=1X71fTx1LKA

Efficient Multi-label Classification with Many Labels

- Paper:http://proceedings.mlr.press/v28/bi13.pdf

The utiml Package: Multi-label Classification in R

- https://journal.r-project.org/archive/2018/RJ-2018-041/RJ-2018-041.pdf

Working with Multilabel Datasets in R: The mldr Package

- Paper: https://journal.r-project.org/archive/2015/RJ-2015-027/RJ-2015-027.pdf

Extreme Multi-label Learning via Nearest Neighbor Graph Partitioning and Embedding

- Talk: https://www.microsoft.com/en-us/research/video/extreme-multi-label-learning-via-nearest-neighbor-graph-partitioning-embedding/

Theory and Specifications: Random Forests for Survival, Regression,and Classification

https://kogalur.github.io/randomForestSRC/theory.html#section8.4