M.S. THESIS

Classification for Multivariate Binary Data based on Association Rule

연관 규칙에 기반한 다변량 이진 데이터 분류 문제의 해결

BY

Myungjun Kim

FEBRUARY 2023

DEPARTMENT OF STATISTICS
COLLEGE OF NATURAL SCIENCES
SEOUL NATIONAL UNIVERSITY

Classification for Multivariate Binary Data based on Association Rule

연관 규칙에 기반한 다변량 이진 데이터 분류 문제의 해결

지도교수 PARK JUN YONG

이 논문을 이학석사 학위논문으로 제출함

2022년 10월

서울대학교 대학원
통계학과
김 명 준

김명준의 이학석사 학위논문을 인준함

2022년 12월

위 원 장	박 태 성	(인)
부위원장	PARK JUN YONG	(인)
위 원	이 권 상	(인)

Abstract

High-dimensional data refers to data which contains a lot of variables more than or equal to the number of observations. When dealing with high-dimensional data, it is necessary to select variables with high importance for further analysis, and association rule can be a useful method when data is binary. Association rule is one of the data mining techniques that extracts meaningful relationships from data. In this thesis, association rule will be used to analyze microbial DNA fingerprint data. To this end, this thesis uses association rule as a classifier and compares it with several machine learning models. Also, this thesis proposes a variable selection algorithm based on association rule. By comparing association rule with other variable selection methods, it was found that association rule is a useful technique to solve classification problems for multivariate binary data.

Keywords: Classification, Association rule, High-dimensional data, Multivariate binary data

Student Number: 2021-26231

Contents

List of Figures

List of Tables

Chapter 1

Introduction

High-dimensional data refers to data which contains a lot of variables more than or equal to the number of observations. When dealing with high-dimensional data, it is necessary to select variables with high importance for further analysis, such as classification or regression. To this end, some variable selection processes, such as forward stepwise selection and best subset selection, can be used.

If the data is binary data containing only values of 0 and 1, association rule can be a useful method for variable selection. Association rule is one of the data mining techniques that extract meaningful relationships from data, and is used in many fields such as market basket analysis (Agrawal 1993) and bioinformatics. In many cases, a classification problem for high-dimensional binary data needs to be solved. For example, to find out the treatment a sample received, microbial fingerprint data from soil samples will be used. In such cases, association rule is useful in that it can extract the relationship between variables and treatment.

In this thesis, association rule and several machine learning models will be used to analyze microbial DNA fingerprint data (Wilbur et al. 2002) and identify the treatment each sample received. First, association rule will be used as a classifier to compare performance with machine learning models. Next, association rule will be used as a variable selection algorithm rather than a classifier, and then the classification problem will be solved using the selected variable. Furthermore, comparison to the method proposed by Wilbur et al. will be performed to check the usefulness of association rule in classification problems for multivariate binary data.

Chapter 2 outlines the association rule and machine learning models to be used in the analysis. Chapter 3 introduces the explanation of microbial DNA fingerprint data, model evaluation metrics, and analysis results. Chapter 4 provides a conclusion, and the appendix contains the `Python` and R codes used in the analysis.

Chapter 2

Classification Methods

2.1 Association Rule

2.1.1 Association Rule

Association rule mining is a data mining technique to discover meaningful relations among variables in a dataset. Due to its descriptive nature, association rule mining has become an important tool in various domains such as market basket analysis (Agrawal 1993) and bioinformatics.

Let $\mathcal{I} = \{I_1, \ldots, I_n\}$ be a set of all variables in the dataset D. D consists of observations $T \subseteq \mathcal{I}$ and each observation can be represented by a binary vector: $T_k = 1$ if T contains I_k and 0 otherwise. T is often called an itemset. An association rule (Agrawal 1993) is the form of

$$X \Rightarrow Y$$

where $X \subseteq \mathcal{I}, Y \subseteq \mathcal{I}$ and $X \cap Y = \emptyset$. The right-hand-side of a rule is called antecedent of the rule and the left-hand-side of a rule is called consequent of

the rule.

To measure the reliability of an association rule, support and confidence (Agrawal 1993) will be used. Support of a rule measures the fraction of observations that contain all variables of interest. One can eliminate rules having low support.

$$\text{support}(X \Rightarrow Y) = \frac{n(X \cup Y)}{|D|}$$

Confidence of a rule is defined to be the percentage of observations including all variables of the rule among observations that carry all variables in the antecedent of the rule. Confidence measures the reliability of the inference made by a rule.

$$\text{confidence}(X \Rightarrow Y) = \frac{n(X \cup Y)}{n(X)} = \frac{\text{support}(X \cup Y)}{\text{support}(X)}$$

The process of generating association rules is as follows: Given the dataset, fix the support threshold and confidence threshold denoted by $minsupp$ and $minconf$, respectively. Then

1. Generate all itemsets whose support is greater than $minsupp$. These itemsets are called frequent itemsets.

2. From each itemset, generate all association rules that pass $minconf$.

Once we find the frequent itemsets in the first step, finding the solution to the second step is straightforward. However, if the data is large, the first step is computationally infeasible because calculating the support of an itemset needs to scan all the data. Also, the number of possible itemsets increases exponentially as the data becomes larger. For example, there are 2^m possible itemsets when the data has m variables. To resolve the problems, some algorithms such as the Apriori algorithm (Agrawal, Srikant 1994) were introduced.

2.1.2 Classification based on Association Rule

Association rule mining finds all rules that pass $minsupp$ and $minconf$. Classification is a process of identifying observations or matching objects to pre-determined categories. The consequent (or target) of an association rule is not pre-determined while there is only pre-determined target in classification. In the view of association rule mining, classification problem aims to discover rules whose consequent are restricted to the class label of response variable. An association rule whose consequent is restricted to the classification class label is called class association rule (CAR) (Liu et al. 1998), and associative classification (Liu et al. 1998) focuses on mining CARs. CBA (Classification Based on Association) algorithm (Liu et al. 1998) is one of the algorithm for associative classification.

2.2 L_1 Regularized Logistic Regression

Logistic regression is a widely used technique in statistical learning when a response variable is categorical. For a binary response variable $Y \in \{0, 1\}$ and p-dimensional explanatory variable $\mathbf{X}$, logistic regression models the relationship of features and the conditional probability of a class given $\mathbf{x}$ as follows:

$$\text{logit}[P(Y = 1|\mathbf{X} = \mathbf{x}; \beta_0, \boldsymbol{\beta})] = \beta_0 + \boldsymbol{\beta}^T\mathbf{x}, \quad \boldsymbol{\beta} \in \mathbb{R}^p,$$

or equivalently

$$P(Y = 1|\mathbf{X} = \mathbf{x}; \beta_0, \boldsymbol{\beta}) = \frac{\exp(\beta_0 + \boldsymbol{\beta}^T\mathbf{x})}{1 + \exp(\beta_0 + \boldsymbol{\beta}^T\mathbf{x})}.$$

The logit transformation allows us to no longer restrict the range of response variable on $[0, 1]$. For a categorical response $Y \in \{1, \ldots, K\}$, logistic regression

can be written as

$$p_k(\mathbf{x};\theta) \equiv P(Y=k|\mathbf{X}=\mathbf{x};\theta) = \frac{\exp(\beta_{0k}+\boldsymbol{\beta}_k^T\mathbf{x})}{1+\sum_{l=1}^{K}\exp(\beta_{0l}+\boldsymbol{\beta}_l^T\mathbf{x})}, \quad k=1,\ldots,K-1.$$

Here $\theta = \{\beta_{01},\ldots,\beta_{0(K-1)},\boldsymbol{\beta}_1^T,\ldots,\boldsymbol{\beta}_{K-1}^T\}$ are the parameters of logistic regression. For a classification problem, we use $k^* = \text{argmax}_k\, p_k(\mathbf{x};\theta)$ as a class prediction for a new observation $\mathbf{x}$.

$$\min_{\theta}\sum_{i=1}^{N} -\log p_{g_i}(\mathbf{x}_i;\theta)$$

Logistic regression models are fit by maximum likelihood estimation. Without any restrictions, it is an unconstrained convex optimization problem, so it can be easily solved by convex optimization methods such as iteratively reweighted least squares (IRLS) (Green 1984).

L_1 regularization used in the lasso (Tibshirani 1996) can be used for variable selection or avoiding overfitting. For (binary) logistic regression, we would minimize penalized objective function:

$$\min_{\beta_0,\boldsymbol{\beta}}\sum_{i=1}^{N} -\log p_{g_i}(\mathbf{x}_i;\theta) + \lambda\|\boldsymbol{\beta}\|_1.$$

This problem is called L_1 regularized logistic regression. It is equivalent to logistic regression with a Laplace prior (Tibshirani 1996).

2.3 Random Forest

2.3.1 Decision Tree

A decision tree (Breiman 1984) is a simple and easily interpretable algorithm for classification and regression problems. Among input variables, it determines the optimal splitting variable and the splitting rule for each node. For a continuous splitting variable x, the corresponding rule is $\{x \geq c\}$ in general. If x is greater

than or equal to c, it is assigned to the left child node, or vice versa. For a categorical variable, the node is divided into two parts. For example, if x has the range of $\{1, 2, 3\}$, one possible rule is $x \in \{1, 3\}$. Using impurity measure, such as Gini index, entropy index or χ^2 statistics, a decision tree grows until the sum of impurity of the child nodes is less than that of the parent node.

We can write the decision tree as

$$f(\mathbf{x}) = \sum_{t \in T} c_t I(\mathbf{x} \in R_t), \quad \mathbf{x} \in \mathbb{R}^p$$

where T is the set of terminal nodes. $R_t = I(x_1 \in R_{t1}, \ldots, x_p \in R_{tp})$ is the splitting rule of node t and R_{tk} is a subset of the domain of x_k. If the sum of squares $\sum(y_i - f(x_i))^2$ is our regression criterion, the optimal predictive value c_t at node t is given by

$$\hat{c}_t = \frac{1}{|R_t|} \sum_{x_i \in R_t} y_i,$$

which is the average value of the response corresponding to input in node t. For a K classification problem, we compute the proportion of class k at node t, i.e.,

$$\hat{p}_{tk} = \frac{1}{|R_t|} \sum_{x_i \in R_t} I(y_i = k),$$

and use $k^* = \text{argmax}_k \, \hat{p}_{tk}$ for prediction.

2.3.2 Random Forest

A decision tree tends to yield a very different result even when the data changes slightly (high variance). Since this instability of decision trees originates from the hierarchical structure, it still remains after pruning the branches of trees. Bagging or bootstrap aggregating (Breiman 1996) reduces the variance by averaging noisy but unbiased learners. To see this, let $\mathcal{L} = \{(\mathbf{x}_i, y_i), i = 1, \ldots, N\}$ be a learning data and the response is continuous. If we adopt the sum of squares

Algorithm 1 Random Forest for Classification (Hastie et al. 2009)

1. Given training data $\mathcal{L} = \{(\mathbf{x}_i, y_i), \mathbf{x}_i \in \mathbb{R}^p, i = 1, \ldots, n\}$, fix $m \leq p$ and the number of trees B. Typically, $m = \sqrt{p}$ or $p/3$.

2. For $b = 1$ to B:

   (a) Draw a bootstrap sample $\mathcal{L}^{(b)}$ from $\mathcal{L}$ with replacement n times.

   (b) Select m input variables randomly among p variables.

   (c) Fit a decision tree $T^{(b)}(\mathbf{x})$ using the boostrapped sample $\mathcal{L}^{(b)}$.

3. Output the ensemble of $\{T^{(b)}(\mathbf{x})\}_1^B$:

   (a) For regression, output $\hat{f}(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B} T^{(b)}(\mathbf{x})$.

   (b) For classification, let N_k be the number of trees whose class prediction is k. Then output $k^* = \text{argmax}_k\, N_k$ (majority vote).

as the optimization criterion, the average prediction error of the aggregated predictor $f_A(\mathbf{x}) = \mathbb{E}_{\mathcal{L}} f(\mathbf{x}, \mathcal{L})$ is

$$\mathbb{E}_{Y,\mathbf{X}}(Y - f_A(\mathbf{X}))^2.$$

Also, the average prediction error of a predictor $f(\mathbf{x}, \mathcal{L})$ is

$$\mathbb{E}_{Y,\mathbf{X}}\mathbb{E}_{\mathcal{L}}(Y - f(\mathbf{X}, \mathcal{L}))^2.$$

Using the Jensen's inequality, we can see that

$$\mathbb{E}_{Y,\mathbf{X}}\mathbb{E}_{\mathcal{L}}(Y - f(\mathbf{X}, \mathcal{L}))^2 \geq \mathbb{E}_{Y,\mathbf{X}}\ (Y - f_A(\mathbf{X}))^2,$$

which shows the improvement of the aggregated predictor. Note that the difference between the two errors depends on the variance of $f(\mathbf{x}, \mathcal{L})$ over $\mathcal{L}$. That

is, the aggregated predictor $f_A(\mathbf{x})$ improves $f(\mathbf{x}, \mathcal{L})$ much more when the difference between $\mathbb{E}_{\mathcal{L}} f(\mathbf{x}, \mathcal{L})^2$ and $[\mathbb{E}_{\mathcal{L}} f(\mathbf{x}, \mathcal{L})]^2$ is large. On the other hand, if $f(\mathbf{x}, \mathcal{L})$ is stable with respect to $\mathcal{L}$, aggregation will not help. In this sense, a decision tree is a good base learner of bagging.

Random forest (Breiman 2001) is an extension of bagging based on decision tree. Unlike a decision tree, random forest only selects $m \leq p$ input variables to de-correlate B decision trees. Without selecting variables, trees might be highly correlated when there exists a variable which distinguishes the data well. When $m = p$, random forest is equivalent to bagging.

Observations which did not appear in the bootstrapped sample are called out-of-bag (OOB) samples. One can validate a model using OOB samples with no price. For an OOB sample $(\mathbf{x}_i, y_i)$, let $O_i = \{b : (\mathbf{x}_i, y_i) \notin \mathcal{L}^{(b)}\}$ and compute

$$\hat{f}_{\text{OOB}}(\mathbf{x}_i) = \frac{1}{|O_i|} \sum_{b \in O_i} T^{(b)}(\mathbf{x}_i).$$

Then the OOB error estimate is given by

$$\text{OOB error} = \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, \hat{f}_{\text{OOB}}(\mathbf{x}_i)).$$

It is known that an OOB error estimate is approximately the same as a leave-one-out cross validation error. Although only a subset of decision trees is used for computing OOB scores, it can be a good alternative for validation when the sample size is not large. The probability of not being included in the bootstrapped sample is

$$\left(\frac{n-1}{n}\right)^n \approx e^{-1} \approx 0.368.$$

About 37% of training data are available for each decision tree and therefore one can use OOB samples for validating random forest or tuning hyperparameter.

2.4 Boosting

2.4.1 AdaBoost

Freud and Schapire (1997) first proposed boosting algorithm called AdaBoost (Adaptive Boosting). Boosting algorithm combines weak learners to improve performance. Unlike bagging which fits the classifiers in parallel, boosting sequentially trains the classifiers giving higher weights to the currently misclassified observations. Then the weighted sum of every classifier becomes the final classifier.

AdaBoost can be understood to be a forward stagewise additive modeling using the exponential loss function

$$L(y, f(x)) = \exp(-yf(x))$$

(Friedman et al. 2000). That is, if $f_m(x)$ is our current model, AdaBoost finds (β_{m+1}, G_{m+1}) by

$$(\beta_{m+1}, G_{m+1}) = \underset{\beta, G}{\operatorname{argmin}} \sum_{i=1}^{N} \exp[-y_i(f_m(x_i) + \beta G(x_i))],$$

and then update

$$f_{m+1}(x) = f_m(x) + \beta_m G_m(x).$$

2.4.2 Gradient Boosting

Gradient boosting proposed by Friedman (2001) extended AdaBoost for general loss functions such as squared error loss or logistic loss. The main idea of gradient boosting is to reduce the errors of the previous model by building a new model using the gradients of the previous model. For squared error loss, the negative gradients called pseudo-residuals are computed by

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} = 2(y_i - F_{m-1}(x_i)), \quad i = 1, \ldots, N.$$

Algorithm 2 AdaBoost.M1 for Classification (Hastie et al. 2009)

1. Start with weights $w_i = 1/N, i = 1, \ldots, N$.

2. For $m = 1$ to M:

   (a) Fit the classifier $G_m(x)$ to the training data using weights w_i.

   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}$$

   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.

   (d) Update $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))], i = 1, \ldots, N$.

3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

However, the negative gradients r_{im} are only defined at the data points, so they cannot be generalized to new observations. As an alternative, given the basis function h, gradient boosting finds $\mathbf{h}_m = \{h(x_i; \gamma_m)\}_{i=1}^{N}$ as close as possible to the negative gradients $\mathbf{r}_m = \{r_{im}\}_{i=1}^{N}$:

$$\gamma_m = \underset{\gamma,\beta}{\text{argmin}} \sum_{i=1}^{N} \{r_{im} - \beta h(x_i; \gamma)\}^2.$$

Then the optimal step length ρ_m is calculated

$$\rho_m = \underset{\rho}{\text{argmin}} \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \rho h(x_i; \gamma_m)),$$

and the current solution is updated

$$F_m(x) = F_{m-1}(x) + \rho_m h(x; \gamma_m).$$

The step length ρ_m plays a role as a shrinkage factor (or learning rate). It controls the rate at which the loss function is minimized. Taking lots of small

steps in the right direction results in better predictions with a testing dataset, i.e., lower variance (Friedman 2001).

2.4.3 XGBoost

Chen and Guestrin (2016) introduced XGBoost (eXtreme Gradient Boosting) to compute efficiently and resolve the overfitting problem of gradient boosting. XGBoost achieves the goal by minimizing the following regularized objective function.

$$\sum_{i=1}^{N} L(\hat{y}_i, y_i) + \sum_{k} \Omega(f_k) \quad \text{where} \quad \Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$$

Here L is a differentiable convex loss function and the second term Ω represents the complexity of the model. T is the number of terminal nodes in the tree and w is the weight for each terminal node. XGBoost has been showing good performance in many areas and it is still widely used because of its useful features such as efficient computation and early stopping.

Chapter 3

Analysis and Results

3.1 Data Description

Characteristic profiles of microbial communities (or community DNA fingerprints) have been used to examine the ecology of microbial systems, particularly those in soil. These community DNA fingerprints can be represented by binary vectors whose dimensions might be very high because some soil samples contain tens of thousands of bacteria (Torsvik et al. 1996). Some methodologies were introduced to investigate community DNA fingerprints. For example, Wilbur et al. (2002) proposed variable selection methods for multivariate binary data.

In this thesis, the microbial community fingerprints data from Wilbur et al. (2002) will be used. While the number of bacterial types has been estimated to be on the order of 10,000, there are only $d = 84$ microbial communities that were identified across $n = 89$ observations. All observations are obtained from four agronomic treatments. According to the tillage practice (plow or no-till) and the rotation practice (monoculture or rotation), all samples were classified

Treatment	X_1	X_2	X_3	X_4	X_5	$\cdots$	X_{82}	X_{83}	X_{84}
1	0	0	1	0	0	$\cdots$	0	1	0
2	1	0	0	0	0	$\cdots$	1	0	1
3	1	1	1	0	1	$\cdots$	0	1	1
4	1	1	0	0	1	$\cdots$	0	0	0
$\vdots$			$\vdots$			$\vdots$		$\vdots$	

Table 3.1: The microbial community fingerprints data (Wilbur et al. 2002). All input variables are binary. Treatment is target variable.

into four categories. The distribution of the samples across the four treatment groups is $n_1 = 23, n_2 = n_3 = n_4 = 22$, where the treatments are (1) corn grown in monoculture in plowed soil, (2) corn grown in monoculture in undisturbed (no-till) soil, (3) corn grown in rotation with soybean in plowed soil, and (4) corn grown in rotation with soybean in undisturbed (no-till) soil. Our purpose is to construct classifiers from the methods explained in the previous chapter and compare the performance of each model.

3.2 Evaluation Metrics

To compare the performance of models, three classification evaluation metrics will be used. The first one is accuracy score. Accuracy score is the fraction of correctly classified samples in the test data.

$$\text{F1 score} = \frac{2}{\text{Precision}^{-1} + \text{Recall}^{-1}}$$

The second evaluation metric is F1 score, which is the harmonic mean of precision and recall. Precision is the fraction of relevant instances among the

		Predicted Class	
		Positive (PP)	Negative (PN)
Actual Class	Positive (P)	True Positive (TP)	False Negative (FN)
	Negative (N)	False Positive (FP)	True Negative (TN)

Table 3.2: A confusion matrix is a table with two rows and two columns that reports the number of true positives, false negatives, false positives, and true negatives. Precision is defined as TP / (TP + FP) and recall (or sensitivity, true positive rate (TPR)) is defined as TP / (TP + FN). Also, the false positive rate (FPR) is defined as FP / (FP + TN).

retrieved instances, while recall is the fraction of relevant instances that were retrieved. F1 score measures the accuracy of a test and has a value between 0 and 1. The closer the value is to 1, the more accurate it is. F1 score can be used to find an equal balance between precision and recall, which is extremely useful when a dataset is imbalanced. For multi-class classification problem, F1 score is calculated by averaging F1 score of all classes.

The last evaluation metric is AUC-ROC curve (Area Under the Curve-Receiver Operating Characteristic curve). A ROC curve illustrates the relationship of the true positive rate (sensitivity) and the false positive rate (1− specificity) as the decision threshold of a binary classifier is varied. If a classifier has poor performance, its ROC curve becomes closer to a straight line (AUC = 0.5), so the AUC can be a good evaluation metric. That is, a good classifier has an AUC value close to 1. For the multi-class classification problem, AUC of each class is calculated by a one-vs-rest strategy, and the average of all AUCs becomes the AUC of a classifier.

Classifier	Parameters
L_1 Logistic Regression	`tol`(tolerance for stopping), `C`(regularization strength)

Random Forest	`max_depth`(the maximum depth), `n_estimators`(the number of trees)
	`max_features`
	(the number of features to consider when looking for the best split),
	`min_samples_leaf`
	(the minimum number of samples required to be at a leaf node),
	`min_samples_split`
	(the minimum number of samples required to split an internal node)

XGBoost	`gamma`(minimum loss reduction), `subsample`(subsample ratio)
	`learning_rate`, `max_depth`,
	`min_child_weight`
	(minimum sum of instance weight needed in a child),
	`colsample_bytree`
	(subsample ratio of columns when constructing each tree)

CBA	`supp`(minimum support), `conf`(minimum confidence)

Table 3.3: Parameters for each model

3.3 Classification

In this section, classification results are presented according to four methods explained in the previous chapter. Using `R` and `Python`, libraries for each method were used; `arulesCBA` library in `R` for classification based on association rule (CBA), `sklearn` library in `Python` for logistic regression and random forest and `xgboost` library in `Python` for XGBoost. Hyperparameter tuning was performed prior to model fitting (Table 3.3). In the case of XGBoost, hyperparameter tuning was performed sequentially to find parameters efficiently. For other methods, optimal parameters are found by a grid search strategy. Using a 5-fold CV strategy, 80% of the total data was used as training data and 20% of the total data was used as test data. Once again, the training data is divided into training data(80%) and validation data(20%) for model fitting and hyperpa-

Classifier	Accuracy score	F1 score	AUC
L_1 Logistic Regression	0.905556(0.91)	0.888982(0.91)	0.978853(0.97)
Random Forest	0.922222(0.93)	0.916034(0.93)	0.985731(0.97)
XGBoost	0.898148(0.88)	0.890594(0.87)	0.980576(0.98)
CBA	0.776238(0.78)	0.770505(0.78)	0.857437(0.85)

Table 3.4: Comparison of classification evaluation metrics of four models.

rameter tuning, respectively. The evaluation metrics were calculated with the test data.

The process was repeated 30 times to calculate the average value of each metric (Table 3.4). All three metrics showed similar trends, and random forest showed the best overall performance. The logistic regression is meaningful in that its performance lags behind that of random forests, but its computation speed was much faster than other models. XGBoost was faster than random forest, but less accurate than logistic regression and random forest. On the other hand, the classifier based on association rules showed worse performance than other models. The results were similar even in the case of using the Leave-One-Out-CV(LOOCV) strategy.

3.4 Variable Selection via Association Rule

In the previous section, association rule was used independently for classification, and performance was poor. In this section, I would like to solve the classification problem by using the association rule in the variable screening process, not as a classifier. The variable screening process for data with an insufficient number of observations and many variables is expected to help improve classifier performance. The variable selection process is conducted using

Algorithm 3 Variable Selection via Association Rule

1. Set the candidate for the hyperparameter of CBA classifier, `supp` and `conf`.
2. For each `supp` and `conf`, fit a classifier with LOOCV strategy.
3. Choose `supp` and `conf` of the best model with respect to accuracy score.
4. Select the variables included in the association rules with the chosen `supp` and `conf`.

grid search and LOOCV strategy (Algorithm 3).

There were 13 variables(S_{AR}) selected through the above process, which was different from the two variable selection methods proposed by Wilbur et al. (2002); the first method selected three variables(S_1), and the second method selected 19 variables(S_2).

$$
\begin{aligned}
S_{AR} &= \{X_9, X_{12}, X_{13}, X_{19}, X_{32}, X_{34}, X_{36}, X_{39}, X_{45}, X_{48}, X_{54}, X_{55}, X_{84}\} \\
S_1 &= \{X_{13}, X_{34}, X_{54}\} \\
S_2 &= \{X_9, X_{12}, X_{13}, X_{14}, X_{19}, X_{32}, X_{34}, X_{36}, X_{39}, X_{40}, X_{43}, X_{45}, X_{46}, \\
&\qquad X_{48}, X_{49}, X_{53}, X_{54}, X_{55}, X_{84}\}
\end{aligned}
$$

The results of solving the classification problem using the selected variable are as follows (Table 3.5-7). It can be seen that the performance of S_1 is not good because there are too few variables included. When the variables selected through the association rule were used, the number of variables was smaller than that of the second method, but the performance was similar with the second method. Therefore, it can be interpreted that variables are selected more effectively among a large number of variables by association rule.

Classifier	S_1	S_2	S_{AR}
L_1 Logistic Regression	0.724074	0.868519	0.848148
Random Forest	0.737037	0.877778	0.840741
XGBoost	0.731481	0.864815	0.809259

Table 3.5: Accuracy score for selected variables

Classifier	S_1	S_2	S_{AR}
L_1 Logistic Regression	0.653317	0.857936	0.838857
Random Forest	0.685617	0.869949	0.834154
XGBoost	0.661207	0.854173	0.798846

Table 3.6: F1 score for selected variables

Classifier	S_1	S_2	S_{AR}
L_1 Logistic Regression	0.916676	0.978493	0.972254
Random Forest	0.918578	0.976708	0.976170
XGBoost	0.922974	0.982932	0.974288

Table 3.7: AUC for selected variables

In addition, the three sets have an inclusion relationship, i.e., $S_1 \subset S_{AR} \subset S_2$. In other words, the association rule tends to select variables with high importance between S_2 as can be seen from the feature importance obtained from random forest (Figure 3.1). Although the evaluation metrics increase as the number of variables increases, the difference is not large, so it can be interpreted that the associated rule efficiently selects the variable.

Unlike the previous section, the logistic regression model showed the best performance. It is meaningful in that the results are better than other machine learning models unlike before, even though it learned the fastest among the three models. On the other hand, XGBoost model performed poorly even though it took the longest time to learn.

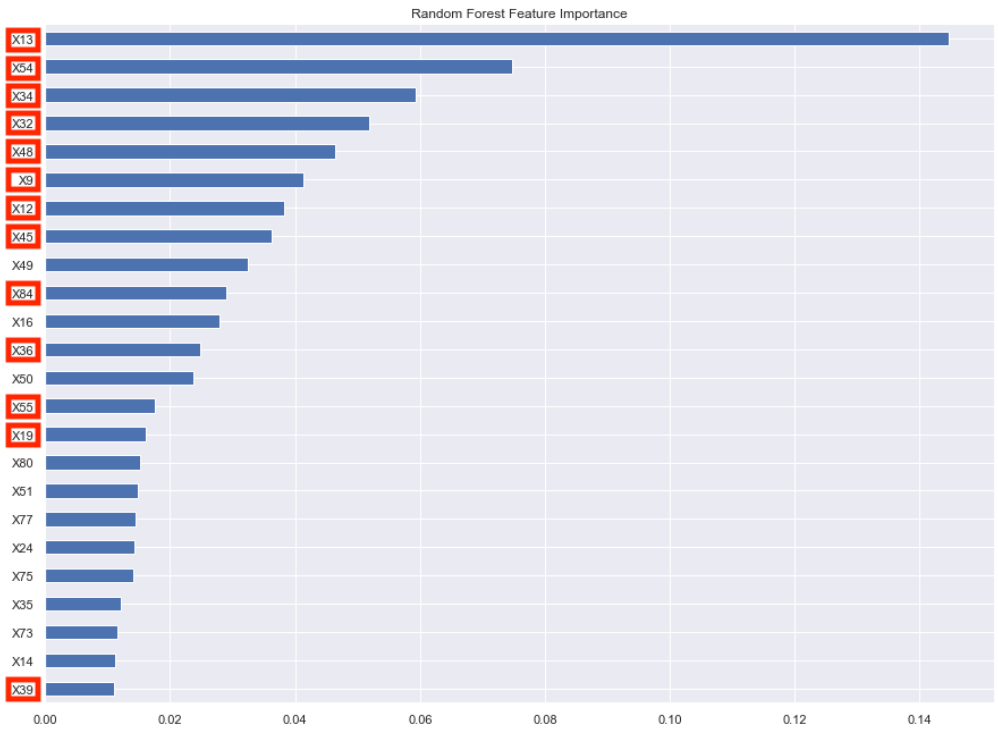

Figure 3.1: Feature importance obtained from the random forest. The highligthed variables were chosen by the association rule.

Chapter 4

Conclusion

The classification problem for multivariate binary data was solved using several machine learning models and association rule. When the association rule was used as a classifier, the performance was significantly worse than that of other machine learning models. The random forest showed the best performance for all evaluation metrics, and the logistic regression model has a lower performance than that, but has a great advantage in terms of computation time. There are only 89 observations in the data, so it did not take a long time for all models to fit the model, but the advantage of logistic regression is expected to stand out if the data size increases.

After using the association rule in the variable screening process, a smaller number of variables were selected, but the accuracy score is almost similar to that of the method proposed by Wilbur et al. It suggests that association rule can be a useful method when selecting variables of high variable importance from high-dimensional data. If the speed of the algorithm for finding association rules is improved, the process can be done more effectively. On the other hand,

the performance of the logistic regression model was the best in this case, and XGBoost still performed the worst.

High-dimensional data is widely covered in many fields such as bioinformatics and genetics, and there is a lot of binary data in such fields. For future research, excluding variables having low variable importance is needed. As seen in the previous result, it is expected that high-dimensional binary data can be effectively analyzed by selecting variables with high importance through the variable screening process using association rule.

Appendix A

Codes

A.1 R Code for Classification based on Association Rule

```
library(caret)
library(arulesCBA)
library(pROC)

n_trial <- 30

accuracy_cba <- rep(0, n_trial)
f1_cba <- rep(0, n_trial)
auc_cba <- rep(0, n_trial)

for (i in 1:n_trial){

    cv <- createFolds(data$trt, k = 5, list = T, returnTrain = F)

    train <- data[-cv$Fold1, ]
```

```
        X_test <- data[cv$Fold1, 2:85]
        y_test <- data[cv$Fold1, 1]

        clf <- CBA(trt ~ ., data = train, supp = 0.1, conf = 0.9, verbose =
        ↪  F)
        y_pred <- predict(clf, X_test)

        acc <- sum(y_test == y_pred) / length(y_test)

        confmat <- confusionMatrix(y_test, y_pred, mode = "everything")
        f1 <- ifelse(is.na(mean(confmat$byClass[, 7])),
                     mean(confmat$byClass[, 7], na.rm = T) * 3 / 4,
                     mean(confmat$byClass[, 7]))

        accuracy_cba[i] <- acc
        f1_cba[i] <- f1
        auc_cba[i] <- auc(multiclass.roc(y_test, as.numeric(y_pred)))
    }
```

A.2 Python Code for L_1 Regularized Logistic Regression

```
    from sklearn.model_selection import train_test_split, KFold,
    ↪  GridSearchCV

    from sklearn.preprocessing import label_binarize
    from sklearn.multiclass import OneVsRestClassifier
    from sklearn.metrics import accuracy_score, f1_score, roc_curve, auc

    from sklearn.linear_model import LogisticRegression

    accuracy_logistic = []
    f1_logistic = []
```

```
11          fprs_logistic = []
12          tprs_logistic = []
13          roc_aucs_logistic = []
14
15          for i in range(n_trial):
16              X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
                ↪  = test_size, random_state = i)
17
18              y_copy = label_binarize(y, classes=[1, 2, 3, 4])
19              X_train_copy, X_test_copy, y_train_copy, y_test_copy =
                ↪  train_test_split(X, y_copy, test_size = test_size, random_state
                ↪  = i)
20
21              logistic = LogisticRegression(penalty = "l1", solver = "liblinear")
22
23              param_grid_log = {
24                  "tol" : [1e-4, 1e-3, 0.01, 0.1],
25                  "C" : [1e-5, 1e-4, 1e-3, 0.01, 0.1, 1, 10, 100]
26              }
27
28              grid_search_log = GridSearchCV(estimator = logistic, param_grid =
                ↪  param_grid_log, cv = 4)
29              grid_search_log.fit(X_train, y_train)
30
31              best_grid_log = grid_search_log.best_estimator_
32              y_pred = best_grid_log.predict(X_test)
33
34              accuracy_logistic.append(accuracy_score(y_test, y_pred))
35              f1_logistic.append(f1_score(y_test, y_pred, average = "macro"))
36
37              classifier = OneVsRestClassifier(best_grid_log)
38              y_score = classifier.fit(X_train_copy,
                ↪  y_train_copy).decision_function(X_test_copy)
```

```
39
40              fpr = dict()
41              tpr = dict()
42              roc_auc = dict()
43
44              for i in range(n_classes):
45                  fpr[i], tpr[i], _ = roc_curve(y_test_copy[:, i], y_score[:, i])
46                  roc_auc[i] = auc(fpr[i], tpr[i])
47
48              fprs_logistic.append(fpr)
49              tprs_logistic.append(tpr)
50              roc_aucs_logistic.append(roc_auc)
```

A.3 Python Code for Random Forest

```
1           from sklearn.ensemble import RandomForestClassifier
2
3           accuracy_rf = []
4           f1_rf = []
5           fprs_rf = []
6           tprs_rf = []
7           roc_aucs_rf = []
8
9           for i in range(n_trial):
10              X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
                ↪  = test_size, random_state = i)
11
12              y_copy = label_binarize(y, classes=[1, 2, 3, 4])
13              X_train_copy, X_test_copy, y_train_copy, y_test_copy =
                ↪  train_test_split(X, y_copy, test_size = test_size, random_state
                ↪  = i)
14
15              rf = RandomForestClassifier()
```

```
16
17              param_grid_rf = {
18                  "max_depth" : [5, 10, 20],
19                  "max_features" : ["sqrt", "log2"],
20                  "min_samples_leaf" : [1, 2, 4],
21                  "min_samples_split" : [2, 5, 10],
22                  "n_estimators" : [20, 50, 80]
23              }
24
25              grid_search_rf = GridSearchCV(estimator = rf, param_grid =
                ↪  param_grid_rf, cv = 4)
26              grid_search_rf.fit(X_train, y_train)
27
28              best_grid_rf = grid_search_rf.best_estimator_
29              y_pred = best_grid_rf.predict(X_test)
30
31              accuracy_rf.append(accuracy_score(y_test, y_pred))
32              f1_rf.append(f1_score(y_test, y_pred, average = "macro"))
33
34              classifier = OneVsRestClassifier(best_grid_rf)
35              y_score = classifier.fit(X_train_copy,
                ↪  y_train_copy).predict_proba(X_test_copy)
36
37              fpr = dict()
38              tpr = dict()
39              roc_auc = dict()
40
41              for i in range(n_classes):
42                  fpr[i], tpr[i], _ = roc_curve(y_test_copy[:, i], y_score[:, i])
43                  roc_auc[i] = auc(fpr[i], tpr[i])
44
45              fprs_rf.append(fpr)
46              tprs_rf.append(tpr)
```

```
47              roc_aucs_rf.append(roc_auc)
```

A.4 Python Code for XGBoost

```
1           from xgboost import XGBClassifier
2
3           accuracy_xgb = []
4           f1_xgb = []
5           fprs_xgb = []
6           tprs_xgb = []
7           roc_aucs_xgb = []
8
9           for i in range(n_trial):
10              X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
                ↪  = test_size, random_state = i)
11
12              y_copy = label_binarize(y, classes=[1, 2, 3, 4])
13              X_train_copy, X_test_copy, y_train_copy, y_test_copy =
                ↪  train_test_split(X, y_copy, test_size = test_size, random_state
                ↪  = i)
14
15          # max_depth, min_child_weight tuning
16          xgb1 = XGBClassifier(
17              learning_rate = 0.1,
18              max_depth = 3,
19              min_child_weight = 5,
20              gamma = 0,
21              subsample = 0.8,
22              colsample_bytree = 0.8,
23              objective = "multi:softmax",
24              nthread = -1,
25              seed = 123)
26
```

```
param_grid_xgb1 = {
    "max_depth" : [3, 6, 9],
    "min_child_weight" : [1, 3, 5]
}

grid_search_xgb1 = GridSearchCV(estimator = xgb1, param_grid =
↪  param_grid_xgb1, cv = 4)
grid_search_xgb1.fit(X_train, y_train - 1)
best_param_1 = grid_search_xgb1.best_params_

# gamma tuning
xgb2 = XGBClassifier(
    learning_rate = 0.1,
    max_depth = best_param_1["max_depth"],
    min_child_weight = best_param_1["min_child_weight"],
    gamma = 0,
    subsample = 0.8,
    colsample_bytree = 0.8,
    objective = "multi:softmax",
    nthread = -1,
    seed = 123)

param_grid_xgb2 = {
    "gamma" : [i/10.0 for i in range(0, 5)]
}

grid_search_xgb2 = GridSearchCV(estimator = xgb2, param_grid =
↪  param_grid_xgb2, cv = 4)
grid_search_xgb2.fit(X_train, y_train - 1)
best_param_2 = grid_search_xgb2.best_params_

# subsample, colsample_bytree tuning
xgb3 = XGBClassifier(
```

```
58              learning_rate = 0.1,
59              max_depth = best_param_1["max_depth"],
60              min_child_weight = best_param_1["min_child_weight"],
61              gamma = best_param_2["gamma"],
62              subsample = 0.8,
63              colsample_bytree = 0.8,
64              objective = "multi:softmax",
65              nthread = -1,
66              seed = 123)
67
68          param_grid_xgb3 = {
69              "subsample" : [i/10.0 for i in range(6, 10)],
70              "colsample_bytree" : [i/10.0 for i in range(6, 10)]
71          }
72
73          grid_search_xgb3 = GridSearchCV(estimator = xgb3, param_grid =
            ↪  param_grid_xgb3, cv = 4)
74          grid_search_xgb3.fit(X_train, y_train - 1)
75          best_param_3 = grid_search_xgb3.best_params_
76
77          # learning_rate tuning
78          xgb4 = XGBClassifier(
79              learning_rate = 0.1,
80              max_depth = best_param_1["max_depth"],
81              min_child_weight = best_param_1["min_child_weight"],
82              gamma = best_param_2["gamma"],
83              subsample = best_param_3["subsample"],
84              colsample_bytree = best_param_3["colsample_bytree"],
85              objective = "multi:softmax",
86              nthread = -1,
87              seed = 123)
88
89          param_grid_xgb4 = {
```

```
        "learning_rate" : [0.01, 0.05, 0.1, 0.15, 0.3]
    }

    grid_search_xgb4 = GridSearchCV(estimator = xgb4, param_grid =
    ↪  param_grid_xgb4, cv = 4)
    grid_search_xgb4.fit(X_train, y_train - 1)
    best_param_4 = grid_search_xgb4.best_params_

    # final model
    xgb_final = XGBClassifier(
        learning_rate = best_param_4["learning_rate"],
        max_depth = best_param_1["max_depth"],
        min_child_weight = best_param_1["min_child_weight"],
        gamma = best_param_2["gamma"],
        subsample = best_param_3["subsample"],
        colsample_bytree = best_param_3["colsample_bytree"],
        num_class = 4,
        objective = "multi:softproba",
        nthread = -1,
        seed = 123)

    xgb_final.fit(X_train, y_train - 1)
    y_pred = xgb_final.predict(X_test)

    accuracy_xgb.append(accuracy_score(y_test - 1, y_pred))
    f1_xgb.append(f1_score(y_test - 1, y_pred, average = "macro"))

    classifier = OneVsRestClassifier(xgb_final)
    y_score = classifier.fit(X_train_copy,
    ↪  y_train_copy).predict_proba(X_test_copy)

    fpr = dict()
    tpr = dict()
```

```
121         roc_auc = dict()
122 
123         for i in range(n_classes):
124             fpr[i], tpr[i], _ = roc_curve(y_test_copy[:, i], y_score[:, i])
125             roc_auc[i] = auc(fpr[i], tpr[i])
126 
127         fprs_xgb.append(fpr)
128         tprs_xgb.append(tpr)
129         roc_aucs_xgb.append(roc_auc)
```

Bibliography

[1] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, page 207–216, 1993.

[2] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, page 487–499, 1994.

[3] Alan Agresti. *Categorical data analysis*. A Wiley-Interscience publication. Wiley, 1990.

[4] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[5] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[6] Leo Breiman, Jerome Friedman, Charles J. Stone, and R.A. Olshen. *Classification and Regression Trees*. Chapman and Hall/CRC, 1984.

[7] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *KDD*, pages 785–794. ACM, 2016.

[8] Alaa Al Deen and M. Nofal. Classification based on association-rule mining techniques : A general survey and empirical comparative evaluation. *Ubiquitous Computing and Communication Journal*, 2011.

[9] Peter K. Dunn and Gordon K. Smyth. *Generalized Linear Models With Examples in R*. Springer, 2018.

[10] Bradley Efron and Trevor Hastie. *Computer Age Statistical Inference: Algorithms, Evidence, and Data Science*. Cambridge University Press, 1st edition, 2016.

[11] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[12] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting. *The Annals of Statistics*, 28(2):337 – 407, 2000.

[13] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189 – 1232, 2001.

[14] Joshua Goodman. Exponential priors for maximum entropy models, 2004.

[15] P. J. Green. Iteratively reweighted least squares for maximum likelihood estimation, and some robust and resistant alternatives. *Journal of the Royal Statistical Society: Series B (Methodological)*, 46(2):149–170, 1984.

[16] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., 2001.

[17] Su-In Lee, Honglak Lee, Pieter Abbeel, and Andrew Y. Ng. Efficient $l1$ regularized logistic regression. In *AAAI*, pages 401–408, 2006.

[18] Bing Liu, Wynne Hsu, and Yiming Ma. Integrating classification and association rule mining. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, page 80–86. AAAI Press, 1998.

[19] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.

[20] Vigdis L. Torsvik, Frida Lise Daae, Ruth-Anne Sandaa, and Lise Ovreås. Novel techniques for analysing microbial diversity in natural and perturbed environments. *Journal of biotechnology*, 64:53–62, 1998.

[21] Chih-Fong Tsai and Mao-Yuan Chen. Variable selection by association rules for customer churn prediction of multimedia on demand. *Expert Systems with Applications*, 37(3):2006–2015, 2010.

[22] J. D. Wilbur, J. K. Ghosh, C. H. Nakatsu, S. M. Brouder, and R. W. Doerge. Variable selection in high-dimensional multivariate binary data with application to the analysis of microbial community dna fingerprints. *Biometrics*, 58(2):378–386, 2002.

[23] Peter M. Williams. Bayesian regularization and pruning using a laplace prior. *Neural Computation*, 7(1):117–143, 01 1995.

초록

고차원 데이터는 변수의 개수가 관측치의 수와 비슷하거나 그 이상으로 많은 데이터를 의미한다. 고차원 데이터를 다룰 때 추후 분석을 위해 중요도가 높은 변수를 선택하는 것은 필수적이며, 데이터가 이진 변수로만 이루어져 있는 경우 연관 규칙은 유용한 방법이 될 수 있다. 연관 규칙은 데이터로부터 유의미한 관계를 추출하는 데이터 마이닝 기법의 하나이다. 본 논문에서는 연관규칙을 활용하여 미생물 DNA 지문 데이터를 분석할 것이다. 이를 위해, 먼저 연관 규칙을 분류기로서 사용하고 여러 머신 러닝 모형과 그 성능을 비교한다. 더 나아가 연관 규칙에 기반한 변수 선택 방법을 제안하고, 이미 알려진 변수 선택 방법과 비교할 것이다. 이를 통해 다변량 이진 데이터의 분류 문제 해결에 있어서 연관 규칙이 유용함을 확인하는 것이 목표이다.

주요어: 분류, 연관 규칙, 고차원 데이터, 다변량 이진 데이터

학번: 2021-26231