UPPSALA
UNIVERSITET

PREDICTION OF DRUG INDICATION LIST BY MACHINE LEARNING

Submitted by
Bolin Wu

A thesis submitted to the Department of Statistics in partial fulfillment of the requirements for a two-year Master of Arts degree in Statistics in the Faculty of Social Sciences

Supervisor
Yukai Yang

Spring, 2021

ABSTRACT

The motivation of this thesis originates from the cooperation with Uppsala Monitoring Centre, a WHO collaborating centre for international drug monitoring. The research question is how to give a good summary of the drug indication list. This thesis proposes a regression tree, Random Forests and XGBoost, known as tree-based models to predict the drug indication summary based on its user statistics and pharmaceutical information. Besides, this thesis also compares the aforementioned tree-based models' prediction performance with the baseline models, which are basic linear regression and support vector regression SVR. The analysis shows SVR with RBF kernel and post-pruning tree are the best models to answer the research question.

Keywords: regression tree, random forests, XGBoost, drug indication, support vector regression

Contents

1 Introduction

1.1 Background

This thesis is written in cooperation with Uppsala Monitoring Centre (UMC). UMC is an independent, non-profit foundation as well as a WHO Collaborating Centre for International Drug Monitoring. UMC maintains the WHO global database called VigiBase. The primary purpose of VigiBase is to collect reports of suspected adverse drug reactions (ADRs) from all over the world. Some reports also provide drug indications, which are recorded valid reasons for someone to use a medication. For example, one indication of paracetamol is headache. Currently, the users of VigiBase are not using the reported indications in VigiBase in any systematic ways. Usually, when people are interested in a drug's indication, they would look up the drug's official label approved by a country's drug regulatory authority. However, as we may encounter in real life, doctors also give prescriptions for off-label indications based on their knowledge and experience. Therefore we would like to make good use of reported indications in VigiBase because they provide both officially labelled and off-label indication. From top to bottom, the Medical Dictionary for Regulatory Activities (MedDRA) hierarchy [1] consists of System Organ Class, High Level Group Term, High Level Term, Preferred Term and Lowest Level Term (LLT). In this thesis, we are interested in the indication at the preferred term (PT) level, and one of our intended users is the internal clinical staff. "Preferred Terms(PTs) is a distinct descriptor (single medical concept) for a symptom, sign, disease diagnosis, therapeutic indication, investigation, surgical or medical procedure, and medical social or family history characteristic"[2].

One problem we face when using reported indications is that each drug could have more than hundreds of PT indications. However, since some of the indications are rarely used or reporting errors, the user may only, for example, be interested in the top 20 or 30 indications. We aim to predict the percentile of indications to be included in a summary of reported indications for a drug. Moreover, we would like to explore what statistical model that is best suited to help answer our research question.

The outline of the paper is as follows. Section 2 introduces the implemented methodologies. Section 3 and Section 4 describes the data and the exact implementation and results of models. Besides, section 4 also includes the prediction results of a sampled test set. Section 5 gives a

[1] Reference link of MedDRA hierarchy: https://www.meddra.org/how-to-use/basics/hierarchy

[2] MedDRA hierarchy definition

discussion of the previous results.

1.2 Literature Review

In this thesis, we have 12 predictors that we select subjectively from VigiBase, and it is unknown which predictors have prediction power statistically. Therefore we choose the tree-based models because several empirical studies have shared that classification and regression tree (CART) has good properties like automatic search mechanism that predictors importance ranking, predictor value selection (Prasad, Iverson, and Liaw 2006) and no need for data transformation (Loh 2014). Lee et al. (2006) argue that CART outperforms traditional discriminant analysis like logistic regression and support vector machine (SVM) in the field of credit scoring.

The tree-based model has been a promising technique for numeric prediction. Since N.Morgan and Sonquist (1963) published the first regression tree algorithm in the literature, researchers have developed a bloom in this field. Breiman et al. (1984) theorized the classification and regression tree (CART) model and provided fundamental properties. Based on that, Bartlett et al. (1998) and Breiman (2001) proposed boosting and random forest respectively. These two methods are well-known ensemble learning techniques that play an instrumental role in regenerating people's interest in CART subject.

However, most of the research is based on big data size, and there is a lack of robust research on its relatively small data size performance. Moreover, labelling data can be pretty expensive in the pharmaceutical science field because of the need for experts and data privacy requirements, but finding potential relevant predictors is easier. Therefore, this paper compares the prediction performance of tree-based models and the baseline models when the input data have many predictors but small sample sizes.

2 Methodology

2.1 The Baseline Models

First we can start with introducing the linear regression model estimated by ordinary least square (OLS). We choose it as one of the baseline models because it is a basic model in statistics. Suppose the data consists of n observations and p predictors, then we can have an equation as follows:

$$y_i = \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} + \varepsilon_i \tag{1}$$

where y_i is the dependent variable, x_{ip} is the predictor, β_p is the coefficient and ϵ_i is the error term. We can also rewrite Equation (1) in matrix notation as:

$$y = X\beta + \varepsilon \tag{2}$$

where y and ε are $n \times 1$ vectors of the values of dependent variables and errors for each observation. X is an $n \times p$ matrix of predictors. By using OLS, β can be estimated as follows:

$$\hat{\beta} = (X^T X)^{-1} X^T y \tag{3}$$

Next, we proceed with introducing Support Vector Regression (SVR), which is SVM for regression. We choose SVR as the other baseline model because it is a standard method of machine learning toolbox and it has a good orientation towards industrial applications (Smola and Schölkopf 2004).

For introductory reasons, we begin by describing a simple linear function with only one predictor:

$$y_i = w_i x_i + \epsilon_i \tag{4}$$

where y_i is the dependent variable, x_i is the predictor, w_i is the coefficient and ϵ_i is the error term.

The object is to minimize the l2-norm of the coefficient:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}||w||^2 \\ \text{subject to} \quad & |y_i - w_i x_i| \leq \varepsilon \end{aligned} \tag{5}$$

In SVR model, we do not care about errors as long as they are less than ε which is known as the principal of maximal margin. However, given a specific constraint ε on errors in (5), we can not guarantee all the data points fall into the margin. For data points that are still fall outside the constraint, we need to take them into account by setting the slack variable ξ which denotes the deviation from the margin.

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}||w||^2 + C\sum_{i=1}^{n}|\xi_i| \\ \text{subject to} \quad & |y_i - w_i x_i| \leq \varepsilon + |\xi_i| \end{aligned} \tag{6}$$

The constant C and ε are two hyperparameters in the algorithm. As C increases, the tolerance for points outside of ε also increases. As ε decreases, the desired accuracy on training set is higher and the error margin is narrower. In practice we can tune the hyperparameters by grid searching and cross validation which we will show in the next empirical analysis section. Another note is that in SVR, the data is scaled by default to obtain a better prediction performance.

Moreover, SVR model uses a set of mathematical functions that are defined as the kernel functions. The purpose of kernel functions is to transform the input data into the required form, aiming for better prediction performance. Two common kernel function for numeric predictions are

- Linear kernel: $K(x, u) = x^T \cdot u$
- Gaussian radial basis function (RBF) : $K(x, u) = exp(-\frac{||x-u||^2}{\sigma^2})$

where x and u above denote all the pairs of data points. For details see Smola and Schölkopf (2004) and Awad and Khanna (2015).

2.2 Regression Tree

In Hastie, Tibshirani, and Friedman (2009), the CART model can be illustrated as in Figure 1. The general idea of the algorithm is to automatically find the splitting variables and split points to split the feature space into different regions. The procedure can be split into two phases: tree growing and tree pruning.

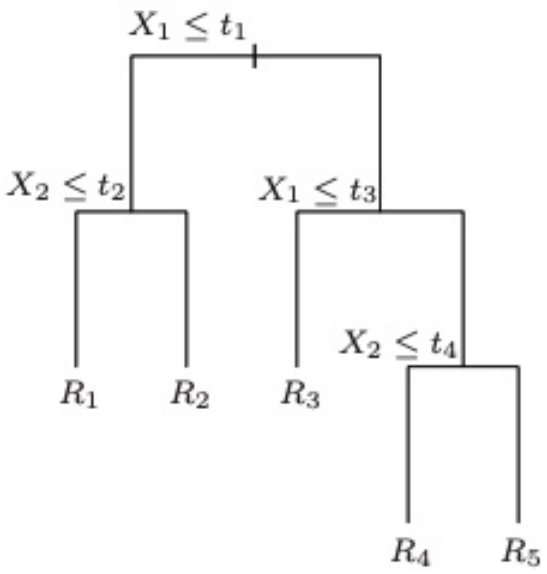


Figure 1: Illustration of the regression tree. Source: Hastie, Tibshirani, and Friedman (2009)

2.2.1 Tree Growing

According to Hastie, Tibshirani, and Friedman (2009), to grow the tree, we seek the splitting variable j and split point s that meet

$$\min_{j,s}[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2] \tag{7}$$

where y_i is the dependent variable, c_1 and c_2 are estimated by Equation (9). The object of Equation (7) is to minimize the node impurity, which is a measure of the homogeneity of the labels at the node.

The pairs of half-planes are defined by:

$$\begin{aligned} R_1(j,s) &= X|X_j \leq s \\ R_2(j,s) &= X|X_j > s \end{aligned} \tag{8}$$

The inner minimization with regard to j and s in Equation (7) is solved by :

$$\begin{aligned} \hat{c}_1 &= ave(y_i|x_i \in R_1(j,s)) \\ \hat{c}_2 &= ave(y_i|x_i \in R_2(j,s)) \end{aligned} \tag{9}$$

Essentially, the tree growing algorithm can be explained by the following four steps:

1. Let j grid over all the variables of the dataset. Let s grid over all the possible values of jth variable.

2. Allocate each observation according to the given j and s into two groups. And then calculate the mean value of each group, $\hat{c}_1$ and $\hat{c}_2$. Get the within group deviation.

3. Return the j and s that give the minimum node impurity. Then we get one split of the tree.

4. Iterate the step 1 - 3 until some condition is reached, e.g. minimum node size and maximum tree depth.

This process can be also called greedy algorithm, because we are griding over all the possible values and return the best split with the smallest within group deviation at each step.

2.2.2 Tree Pruning

After we have fully grown the tree, it may have an over-fitting problem. To generalize the tree better on the test set, we need to prune the tree. Tree pruning can be divided into pre-pruning and post-pruning. Pre-pruning is also known as early stopping criteria. As the name suggests, the criteria are set as parameter values while building the model. For example we can set the maximum depth of a tree, the minimum number of records that must exit in a node for a split to happen and the minimum number of records that can be present in a terminal node.

The strategy of postpruning is to grow a large tree T_0 and we define a subtree $T \in T_0$ to be any tree that can be obtained by pruning T_0. For every subtree T, we can get the cost complexity defined as follows (Hastie, Tibshirani, and Friedman 2009):

$$\sum_{m=1}^{|T|}(\sum_{x_i \in R_m} (y_i - \hat{c}_m)^2) + \alpha|T| \tag{10}$$

where |T| denotes the number of terminal nodes in T, R_m is the plane of node m derived by Equation (8), $\sum_{x_i \in R_m}(y_i - \hat{c}_m)^2$ denotes the sum of squared residuals within each node. The α is the complexity parameter estimated by cross validation. As α increases, more of the tree is pruned, which increases the total impurity of its leaves. See Breiman et al. (1984) for details. The purpose of postpruning is to find the final subtree $T_{\hat{\alpha}}$ that minimizes cost complexity, thus reducing overfitting problem.

2.3 Random Forests

Random Forests is an ensemble method that combines the simplicity of decision trees with flexibility resulting in an improvement in accuracy on test set. The algorithm is as below (Hastie, Tibshirani, and Friedman 2009):

1. For b = 1 to B: Draw a bootstrap sample Z^* of size N from the training data.
2. Create a decision tree using the bootstrapped dataset. The tree growing algorithm is similar to the one described in Section 2.2.1, but only use a random subset of p features at each step.
3. Output the ensemble of trees $\{T_b\}_1^B$.
4. Make a prediction at a new point x: $\hat{f}^B_{\text{random forest}}(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

For regression, the recommended number of feature to sample is P/3 where P is total number of variables in the dataset and the minimum node size is five (ibid.). The idea of Random Forests is to decrease the correlation between the trees. If we consider each tree to be an independent and identically distributed random variable with variance σ^2. The variance of B averaged trees is given by:

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2 \tag{11}$$

where ρ denotes the correlation between the trees. If we increase the B then the second term in expression (11) will vanish. The remaining part is the function of correlation between the trees and the variance. Since we only choose a subset of all the features when constructing the trees, the correlation between the trees is reduced, thus the averaged variance is reduced.

Another advantage of Random Forests is that it uses the predictive ability of all features rather than just a few of them. This usually improves the prediction performance on the test set.

2.4 XGBoost

XGBoost stands for "Extreme Gradient Boosting" which follows the principle of Gradient Boost. It is a powerful machine learning algorithm proposed by Chen and Guestrin (2016).

It earns great reputation in recent years because of its scalability, sophisticated design, computation speed as well as its outstanding prediction performance in many Kaggle [3] competitions. In order to introduce the mechanics of XGBoost we need to first review the concepts of Gradient Boost algorithm. In this paper, we will introduce the algorithms in a self-contained and principled way so that the explanations are clean and formal.

2.4.1 Gradient Boost

Intuitively speaking, Gradient Boost constructs a series of regression trees so that the latter tree is built based on the error made by the previous trees with scaling. And it iterates until it has made the number of trees that users ask for or additional trees fail to improve the fit.

Mathematically, the Gradient Boost algorithm (Friedman 2002) is as follows. Please note that all the variables are defined below the algorithm.

1. Input: Data $\{x_i, y_i\}_{i=1}^n$ and a differentiable loss function $L(y_i, F(x))$.

2. Initialize model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma)$$

3. Let M denote the total number of trees. For m =1 to M:

   (a) For i = 1,...,n compute:

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

   (b) Fit a regression tree to the r_{im} values and create planes R_{jm}

   (c) Let J_m denote the total number of leaves. For j = 1,...,J_m compute:

$$\gamma_{im} = \arg\min_{\gamma} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(X_i) + \gamma)$$

   (d) Update

$$F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$$

4. Output $F_M(x)$

[3] Kaggle is an online community of data scientists and machine learning practitioners

In Step 1, one popular loss function for regression is $1/2(y_i - F(x))^2$ where F(x) is the function that gives the predicted values. In Step 2, γ denotes the predicted value. We could either use gradient descent or first derivative to solve for $F_0(x)$. In Step 3 (a), if we use the loss function $1/2(y_i - F(x))^2$, then r_{im} values are the same as residuals of each sample. However, it is technically called pseudo residuals because if we use another loss function, e.g. $(y_i - F(x))^2$, then r_{im} denotes a process similar to calculating the residuals, but not exactly the same. In Step 3 (b), we use the regression tree to grow the tree. In Step 3 (c), we calculate the output value for each leaf. It is similar to the expression in Step 2, but one difference is that here we are taking the previous prediction into account. Another difference is that the summation only considers the samples in each leaf instead of all of the samples. In Step 3 (d), ν denotes the learning rate which is between 0 and 1. A smaller ν restricts the influence of each tree on the final prediction. The summation represents the addition of the output values $\gamma_{j,m}$ for all the leaves $R_{j,m}$ that x can be found in.

In summary, when Gradient Boost is used for regression with loss function to be $1/2(y_i - F(x))^2$, we start with a leaf that is the average value of the variable we want to predict. Then we estimate a tree based on the residuals. And we scale the tree's contribution to the final prediction with a learning rate. After that we include another tree based on new residuals. Finally, we keep including trees based on the error made by the previous trees until certain conditions are fulfilled.

2.4.2 XGBoost Principles

XGBoost is built based on the Gradient Boost algorithm. However, there are several differences in modeling details.

Firstly, XGBoost used a more regularized model formalization to control over-fitting (Chen and Guestrin 2016). The object function that we want to minimize in XGBoost is as follows:

$$\begin{aligned} L(\phi) &= \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k) \\ \text{where } \Omega(f_k) &= \gamma T + \frac{1}{2}\lambda ||w||^2 \end{aligned} \tag{12}$$

In Equation (12), we can see that the object function consists of two parts: a differentiable convex loss function l and the regularized term Ω. T is the number of terminal nodes in a tree, γ is a user defined penalty term which encourages pruning, w is the output value of a leaf, λ is a scalar of regularization penalty. The purpose of the equation is to find the optimal output

value w to minimize the object function $L(\phi)$. It can be solved by using second order Taylor polynomial. For details please see Chen and Guestrin (2016).

Another difference is that XGBoost uses its uniquely constructed tree instead of a regression tree. When growing XGBoost Trees for Regression, we calculate similarity scores and gain to determine how to split the data. And we make the splits up to the specified maximum depth. After that we prune the tree backwards by calculating the differences between gain values and a user defined tree complexity parameter,γ. The similarity score and gain of a leaf h_L are defined as follows:

$$\begin{aligned} \text{Similarity Score} &= \frac{1}{2}\frac{(\sum_{i\in h_L} g_i)^2}{\sum_{i\in h_L} h_i + \lambda} \\ \text{Gain} &= \text{Left}_{\text{Similarity Score}} + \text{Right}_{\text{Similarity Score}} \\ &\quad - \text{Root}_{\text{Similarity Score}} \\ \text{Gain} - \gamma &= \begin{cases} \text{positive number} & \text{then keep the branch} \\ \text{negative number} & \text{then prune the branch} \end{cases} \end{aligned} \tag{13}$$

Where g_i and h_i represents the first and second derivative of the loss function $l(\hat{y}_i, y_i)$ respectively. And the output value that gives the largest gain is set to be the split point.

2.5 Evaluation Method

In this thesis we choose two evaluation metrics: root of mean squared error (RMSE) and mean absolute error (MAE). The definitions are listed as follows.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{N}}$$

$$MAE = \frac{\sum_{i=1}^{N}|y_i - \hat{y}_i|}{N}$$

where y_i is the observed value, $\hat{y}_i$ is the predicted value and N is the total number of observations.

One difference between RMSE and MAE is that compared to MAE, RMSE does not treat each error the same. RMSE gives more weights to larger errors while MAE is less sensitive to outliers.

When we split the dataset into training set and test set by random sampling, we may face the problem of variability of evaluation results on the test set due to the randomness. To make

the comparison of different models more robust, we will use k-fold cross-validation (CV) as follows:

1. Split the observations randomly into k groups.
2. For j = 1 to k:
   (a) Let the observations in group j be the test set and estimate the model on the remaining k-1 groups.
   (b) Make the predictions for the observations in group j.
   (c) Calculate sample $\mathrm{RMSE_j}$ and $\mathrm{MAE_j}$ with the calculated predictions and true values in group j.
3. Compute the overall k-fold CV RMSE : $\sqrt{\frac{\sum_{i=1}^{N}(y_i-\hat{y_i})^2}{N}}$ and MAE: $\frac{\sum_{i=1}^{N}|y_i-\hat{y_i}|}{N}$.

Considering the computation power, we choose 10-fold CV (k = 10) in this thesis. We will evaluate prediction performance of different models by comparing their overall RMSE and MAE as well as the sample $\mathrm{RMSE_j}$ and $\mathrm{MAE_j}$.

3 Data

We choose 12 predictors to predict the length of the medical indication list. To get the labelled data, firstly we find the indications of top 60 most common drugs in the VigiBase. After that we get indication mapped to the MedDRA LLT. Then we use the MedDRA hierarchy to group each drug on PT level and count the entry of each indication's record on PT level in all the reports, sorting in descending order. Finally a medical doctor labels the data by annotating the cutting index of each drug. The cutting index is a threshold that every indication above it should be considered as an interested indication. The bigger the cutting index is, the more indications should be included in the summary of a drug and vice versa. An example of the exported indication is Table 1. Please note that due to the sensitive nature of the data, the numbers in the table are simulated.

Table 1: An Example of Indication List of Acetylsalicylic Acid

Index	Number of Entry	PT Level Indication
1	21590	Prophylaxis
2	15322	Cardiac disorder
2	5690	Pain
...	...	...
710	4	Obesity

The data set consists of 12 predictors and one label:

- n_indications: The number of distinct reported indications of the drug.
- avg_age: The average age of patients who take a specific drug.
- avg_weight: The average weight of patients who take a specific drug.
- age_range: The age range of patients who take a specific drug. It is calculated by maximum age minus minimum age.
- n_country: The number of distinct countries from which reports for a drug were entered in VigiBase.
- n_route: The number of reported paths of administration of a drug.
- n_dosage_number: The number of distinct structured dose number of the drug. We will give an example below.
- n_dosage_unit: The distinct number of structured dose units of the drug. For example if we say 2 mg in one dose, then "2" is the dose number and "mg" is the dose unit.
- n_ATC: The number of distinct ATC [4] number of a drug. The ATC number classifies an active drug substance into anatomical, therapeutic, pharmacological and chemical subgroups.
- n_body: The number of parts of body that a drug can be used to. It is identified by the first level of ATC.

[4]Reference link of ATC: https://www.whocc.no/atc/structure_and_principles/

- n_co_reported_drugs: The total number of co-reported drugs of the drug.
- n_null_uni_reports: The number of reports without dosage information of the drug.
- percentile: The cutting index of a drug's indication divided by the total number of rows of its indication list. The cutting index is labeled by a medical doctor manually.

The percentile is what we would like to predict for each drug. The larger percentile is, the larger proportion of its original indications list would be included for the summary of a drug and vice versa.
In this thesis, because of limited resources of labelling data, the sample size is 60.

4 Empirical Analysis

We mainly use R to prepare the data as well as build the models. For data pre-processing, we use "tidyverse" library. To train the regression tree model, Random Forests and XGBoost, we use "rpart" ,"randomForest", and "xgboost" packages, respectively. And in the following analysis, all the grid searchings of optimal parameters use 10-fold cross-validation.

Moreover, since there are ten estimated models in total in 10-fold CV so that it will be too long to list all of their results in the thesis. Therefore the following model results, for example, percentile prediction, tree model visualization and feature importance, are based on the first 10-fold CV sample with the number of observations to be fifty-four and six in the training set and test set respectively. The drug names of the six sampled test data are celecoxib, diazepam, fentanyl, interferon beta-1a, iron, and lorazepam.

4.1 The Baseline Models

Since the goal of the linear regression model in this thesis is to make prediction instead of inference, the statistical hypothesis tests are not our main concern. Therefore we will not examine the significance of variables and hypothesis test for each 10-fold CV iteration. The 10-fold CV RMSE and MAE for the linear regression model are 0.0689 and 0.0520, respectively.

In terms of SVR, as mentioned previously in Section 2.1, we need to find the optimal hyperparameters C and ε. The recommended search range of C and σ is the exponentially growing sequence. (Hsu, Chang, and Lin 2003). And when the kernel is RBF, we also need to tune the parameter σ. We will use "e1071" package in R. And the parameter tuning can be

done by the "tune()" function in this package, which uses 10-fold cross-validation by default. One note is that in this package, the parameter σ is measured by the argument "gamma". The grid range region is (0.001,0.01,0.1,1,10,100) for C, (0.01,0.01,0.1,1,10,100) for gamma and (0.01,0.1,1) for ε. For each loop in 10-fold CV, we find the optimal parameters, estimate SVR and calculate the prediction values. The results are listed below:

Table 2: 10-fold CV Results of SVR

Kernel Function	RMSE	MAE
Linear	0.0703	0.0543
RBF	0.0504	0.0622

One set of the predicted percentile given by the baseline models is shown in Figure 2. Given the sample data, the linear regression is good at predicting diazepam and fentanyl. The SVR with RBF kernel is better at predicting the percentile of celecoxib and interferon beta-1a. The SVR with linear kernel makes a good prediction for lorazepam. However, none of the baseline models gives a good prediction for iron.

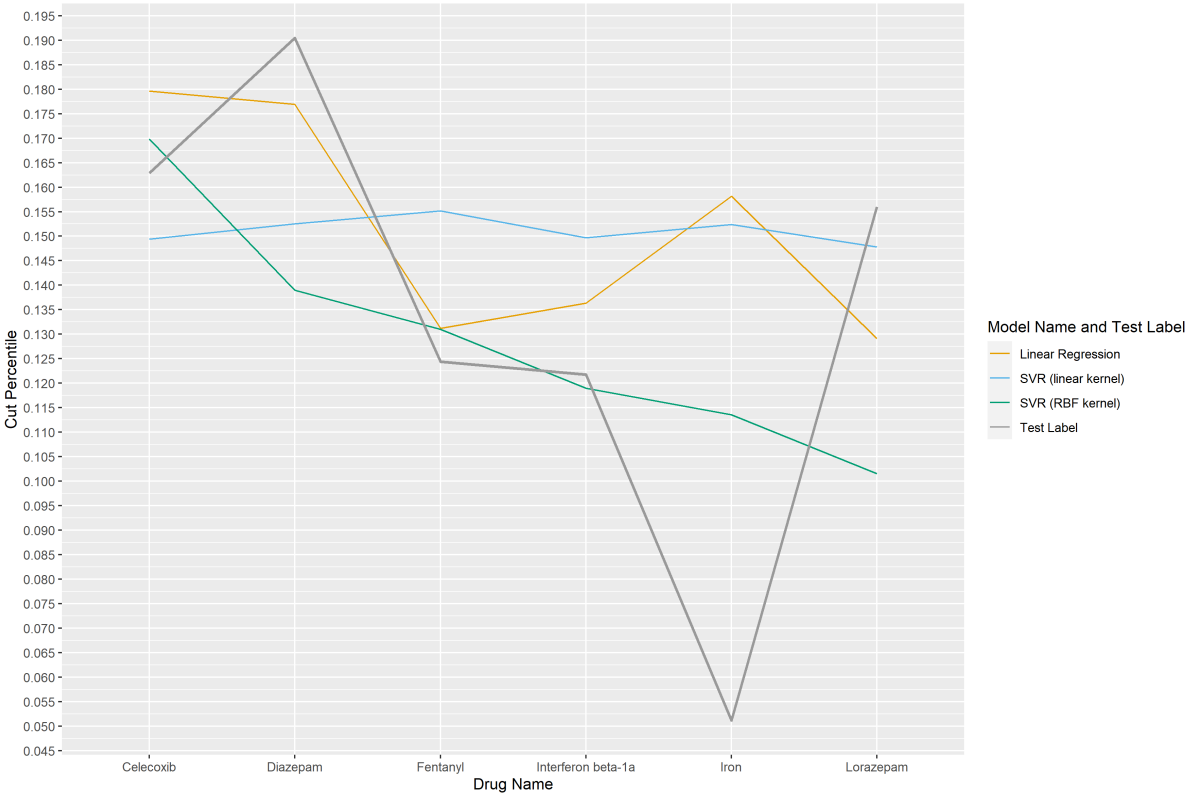


Figure 2: Prediction Results of Baseline Models

4.2 Regression Tree

In the regression tree model, we do not need to tune the parameters for the base tree. We let the base tree grow fully with a minimum number of observations in any terminal node to be two. For the post-pruning tree, the complexity parameter is derived from the base tree's complexity parameter table. The parameter tuning of the pre-pruning tree needs to be set up manually, which we will explain below.

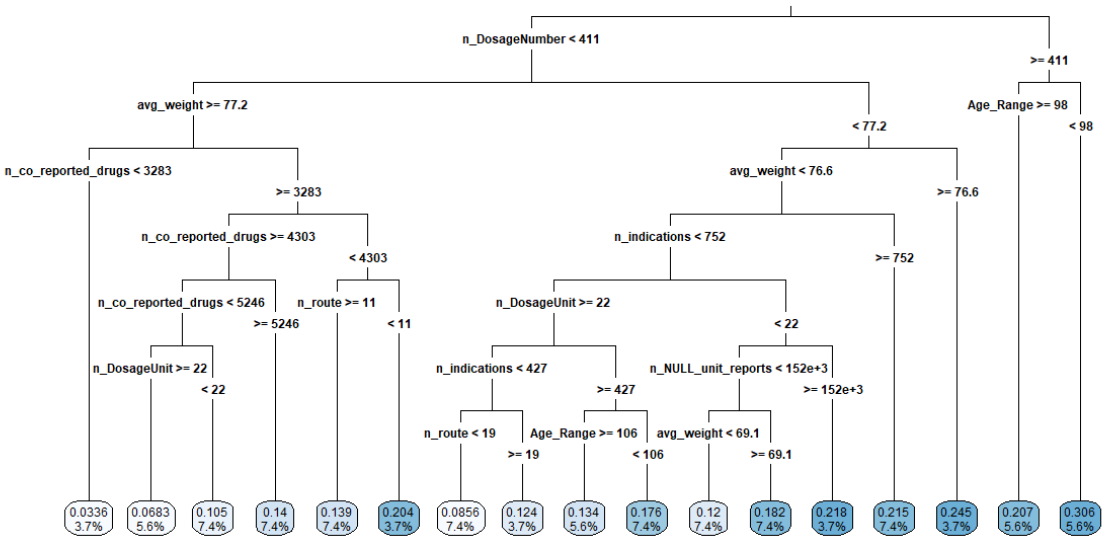


Figure 3: An Estimated Fully Grown Base Tree

Figure 3 is a visualization of the base tree. The number in each circle denotes the predicted value in its node; the percentage means the ratio of observations falls into that node. We can see that the base tree is deep with the depth to be seven and may have an over-fitting problem. Figure 4 shows the estimated relative errors with different complexity parameters. The post-pruning strategy is to choose the best complexity parameter that gives the smallest relative error in Figure 4. The relative error is estimated by cross-validation, and we view it as an approximation of RMSE of the test set. An example of a post-pruning tree with the best complexity parameter, which is 0.16 in this case, is shown in Figure 5.

In Figure 5 the post-pruning tree has a much shallower depth which may help reduce the over-fitting problem.

For the pre-pruning tree, we need to determine the three main arguments. The first is the minimum number of observations in a node for a split to be attempted (minsplit). The second

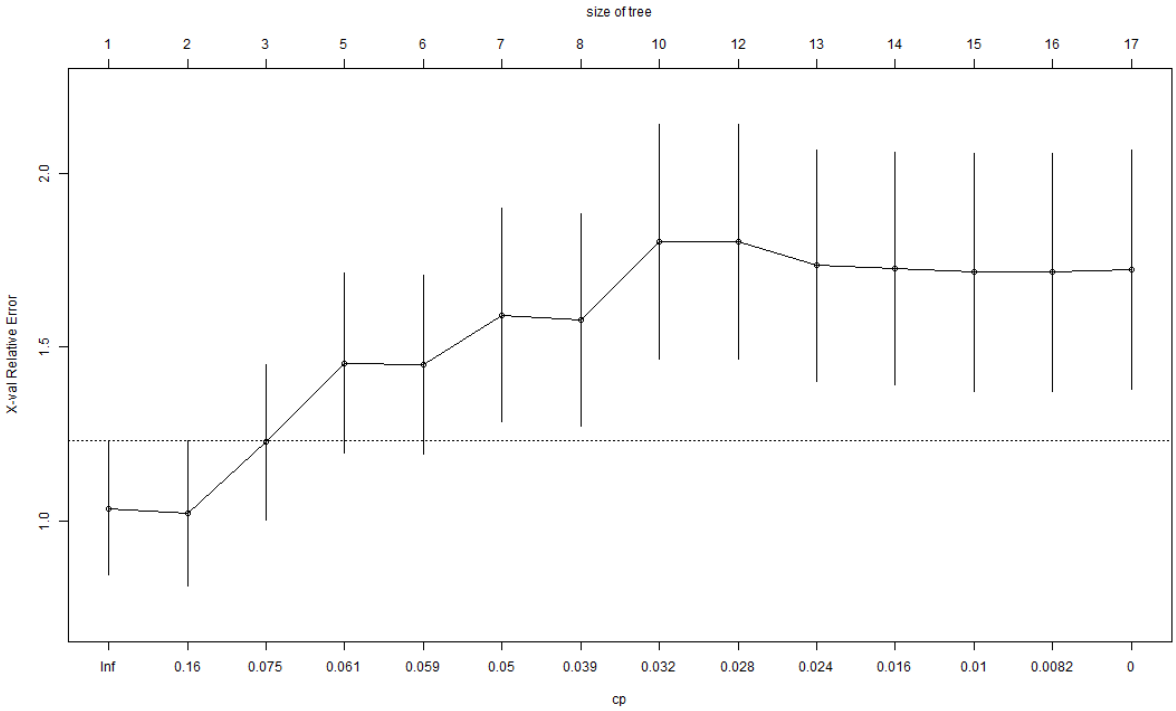


Figure 4: The Complexity Parameters of the Estimated Base Tree

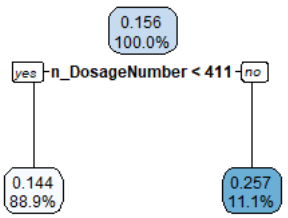


Figure 5: The Estimated Post-pruning Tree, cp = 0.16

is the minimum number of observations in any terminal node (minbucket). The third is the maximum depth of any node of the final tree (maxdepth). The grid searching information and corresponding 10-fold CV RMSE is listed in Table 3 and Table 4. One example of a pre-pruning tree with the best cross-validated parameters is shown in Figure 6 which is shallower than the

base tree as expected.

Table 3: Grid Searching Setup of Prepruning Parameters

Parameter	Range	Number of Combinations	Time Consumption per CV Iteration
minsplit	(6, 9, 12, 21)	80	5.76 seconds
minbucket	(2,3,4,7)		
maxdepth	(1, 3, 5, 7, 9)		

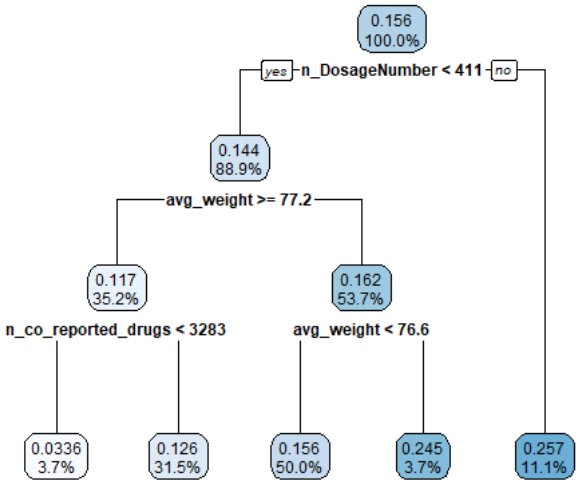


Figure 6: The Estimated Pre-pruning Tree, minsplit = 10,minbucket = 2, maxdepth = 3

After we grasp the estimation of each regression tree, we can make a comparison of 10-fold CV RMSE of each model. The results are listed in Table 4. we can see that the post-pruning tree gives the best prediction performance while the base tree model to be the worst.

Table 4: 10-fold CV Results of Regression Tree

Type	RMSE	MAE
Base Tree	0.0736	0.0590
Pre-pruning Tree	0.0663	0.0528
Post-pruning Tree	0.0673	0.0520

4.3 Random Forests

There are two important parameters in the Random Forests algorithm: The number of trees used in the forest (ntree) and the number of variables randomly sampled as candidates at each split (mtry). The grid searching information is listed in Table 5. Compared with the regression trees' prediction performance, Random Forests gives better results with the 10-fold CV RMSE and MAE on the test set to be 0.0648 and 0.0529, respectively.

Table 5: Grid Searching Setup of Random Forests Parameters

Parameter	Range	Number of Combinations	Time Consumption per CV Iteration
ntree	(1,11,...,191)	160	20.90
mtry	(1,2,...,8)		seconds

Furthermore, the Random Forests can produce the feature importance of each variable. It is useful when we want to investigate the contribution of each predictor to our model. Figure 7 shows two measures of feature importance of different predictors. "%IncMSE" is the increase in mean squared error of predictions as a result of variable j being permuted. "IncNodePurity" relates to the node impurity difference before and after the split, which is summed over all splits for that variable, over all trees. We can see that the number of distinct structured dose number of the drug is the essential features which is consistent as seen in the post-pruning tree. The average weight of patients, the number of reports without dosage information and the number of distinct indications and the number of co-reported drugs share similar prediction importance. The distinct ATC number is the least important predictor.

4.4 XGBoost

Since there are seven booster parameters in the function, it is nearly impossible to get a set of universal optimal parameters. Besides, our main concern is to reduce the test error. Therefore the tuning strategy is focusing on girding the parameters that prevent over-fitting: learning rate (η), complexity parameter (γ) and the sub-sample ratio of the training instance (subsample). The grid search range is listed below. Other arguments are default values, with the maximum number of iterations (nrounds) to be 100, the number of features supplied to a tree (colsample_bytree) to be 1, minimum number of instances required in a child node (min_child_weight)

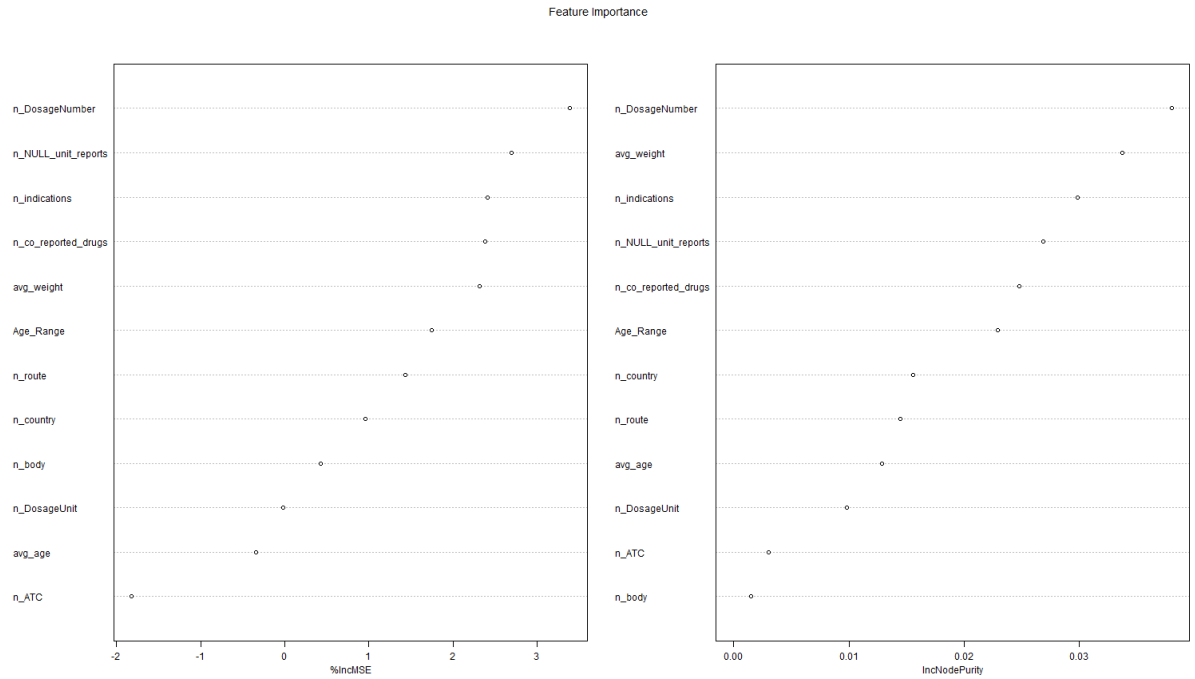


Figure 7: An Example of Estimated Feature Importance Chart from Random Forests

to be 1, and the maximum depth of tree (max_depth) to be 6. Here we do not tune the maximum depth because it is related to γ already.

Based on the information given in Table 6, we can see that for one set of training and test data, the grid searching takes around 9.53 minutes, therefore for the whole 10-fold CV procedure, it takes about 9.53×10 minutes to finish. Given the grid searching set up, the final 10-fold CV RMSE and MAE are 0.0731 and 0.0594 respectively.

Table 6: Grid Searching Setup of XGBoost Parameters

Parameter	Range	Number of Combinations	Time Consumption per CV Iteration
η	(0,0.05,0.10,...,0.3)	378	9.53 minutes
γ	(0,10,20,...,80)		
subsample ratio	(0,0.1,0.2,...,0.5)		

Similar to Random Forests, XGBoost estimates feature importance as well. Figure 8 shows that the average weight of patients is the most important predictor. However, the other variables give much less contribution compared with the feature importance result of Random Forests. Therefore, given the sample set, the predictor importance of XGBoost is less balanced than the one of Random Forests. This may be one of reasons why XGboost does not give a better

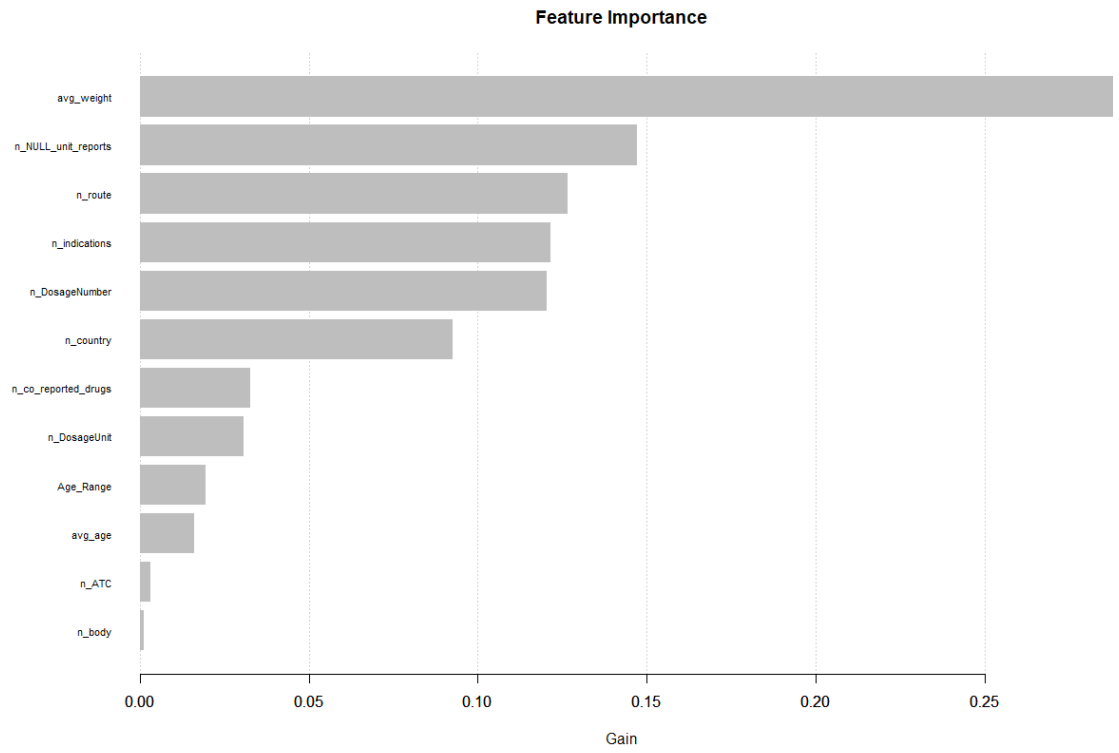


Figure 8: An Example of Estimated Feature Importance Chart from XGBoost

overall prediction performance.

Now let us proceed to compare the prediction results of the tree-based models. In Figure 9 there are two noticeable points. Firstly, the post-pruning and pre-pruning tree produce the same predictions for all of the six test samples. The reason could be that both trees are shallow, and the six samples happened to fall in the same leaves. The second is similar to baseline models, none of the tree-based models gives an ideal prediction for iron.

At last, we can compare the prediction performance of the models mentioned above.The Figure 10a in appendix tells us their performance overall and Figure 10b shows their performance for each of the CV samples.

In terms of MAE, SVR with RBF kernel is the best model on average and it has the smallest range difference of sample MAE. The post-pruning tree and pre-pruning tree can give the best possible predictions since their minimum sample MAE values are the lowest. However, the post-pruning tree is better than pre-pruning tree because it has smaller range difference. The base tree is the worst model because it has a high overall MAE value, and its maximum sample MAE is the highest.

When it comes to RMSE, Random Forests and SVR with RBF kernel are the two best

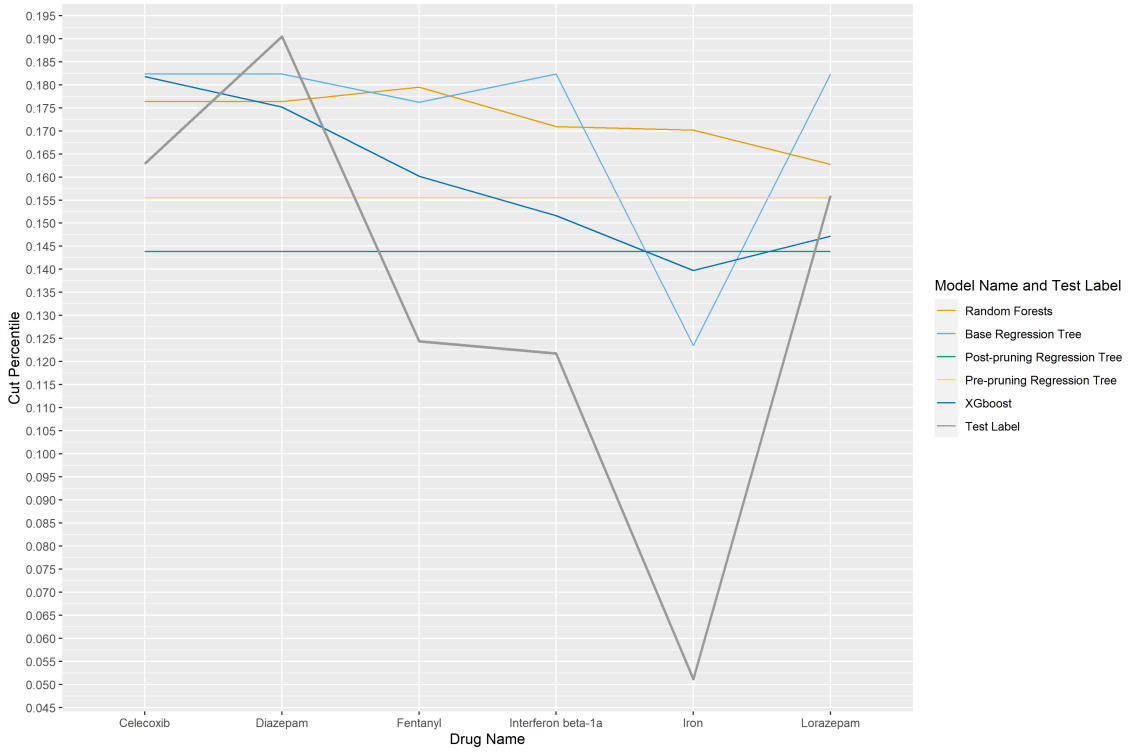


Figure 9: Prediction Results of Tree-based Models

model on average, and they have the smallest sample RMSE range difference. Similarly, the two pruning trees are still the ones that give the lowest minimum sample RMSE. The Random Forests has the lowest maximum sample RMSE.

When comparing the most complicated model XGBoost and the most basic linear regression, we find that XGBoost gives both higher overall RMSE and MAE as well as higher minimum and maximum sample RMSE and MAE. Therefore in our study, the XGBoost is worse than the linear regression.

5 Discussion

In this thesis, we predict the percentile of drug indication given the twelve predictors from VigiBase. It can be a helpful tool for retrieving interested indications of a drug to future VigiBase users, for example, internal clinical staff. By inputting a drug's 12 predictors that are mentioned in Section 3, the user can get a good summary of its indication list.

To have an overview of prediction performance, we compare different models from the perspective of MAE and RMSE. On one hand, if we only consider the overall prediction performance, then SVR with the RBF kernel is the best model to answer our research question

because it has the lowest values for both RMSE and MAE. The reason could be that, instead of focusing on minimizing the errors, SVR uses a soft margin which results in a good generalization on the test sets.

On the other hand, if we care about the prediction performance on different cross-validated samples, the post-pruning tree is the best model. Its minimum sample RMSE and MAE are the lowest, which means it has a possibility to make the best predictions for some drugs. Besides, compared with pre-pruning tree, post-pruning tree has a lower maximum RMSE and MAE, indicating that the worst prediction of post-pruning tree is better than one of the pre-pruning tree. The reason could be that for each training set, the post-pruning tree will calculate a new cross-validated complexity parameter to prune the base tree rather than following a fixed grid searching pattern. Therefore post-pruning tree has a better generalization than pre-pruning tree on the test set. Although the overall RMSE and MAE of post-pruning tree are not the best among the six models, we can find that the difference is acceptable. The differences between the post-pruning tree and SVR with RBF kernel are 0.005 and 0.002 for RMSE and MAE, respectively.

Moreover, we also find that the XGBoost model fails to outperform other models as we supposed. One reason could be that XGBoost is a complicated model with lots of parameters. We do not have enough computation power to grid search an extensive range of parameters to reduce the over-fitting problem when iterating the cross-validation. The other reason is that our dataset is not large and complicated enough to exploit the ability of XGBoost fully. In our case, the baseline models are better choices than XGBoost considering their similar performance but a considerable gap in computation time. However, XGboost, like Random Forests, gives the feature importance information which the baseline models do not provide. The average weight of patients (avg_weight) and the number of distinct structured dose number of the drug (n_DosageNumber) are the most essential features for XGBoost and Random Forests respectively. This is reasonable because if there is more flexibility to prescribe a drug's dosage, then a doctor is more likely to prescribe it to the patients. In addition, a drug given at a different dose may be used for different indications. For example, Acetylsalicylic acid at 75 mg is used as a blood thinning drug to prevent blood clots while the dose of 500 mg is used to treat pain and inflammation. Thus the drug may have more interested indications. And the average weight may implicitly contain other information. For example, if the average weight of patients is larger than 90 kg, we may assume that perhaps most of the patients are male or they are adults. The

drugs given to varying age groups are expected to have more indications. Therefore it could also be a good predictor. Another example is that a higher average weight would include more obese people and obesity is linked to increased risk of many diseases so that more interested indications should be included.

For future research, annotating more training data and including more features in the study would be a good idea since tree-based models, especially XGBoost, are excellent at handling large complex data set. Besides, in this thesis, the data set is from the sixty most common drugs. We need more annotation of the drugs with fewer reported cases in VigiBase so that the model will have better scalability. In addition, given the predicted percentile, we can consider applying the clustering method to the indications in the predicted percentile to derive a more concise final indication list. Furthermore, we can try to predict if an individual indication should be included in the list to have a more precise list. Last but not least, in this thesis, we do list-wise deletion for the records with missing value so that the models do not use any information on missing data. In the future, we can use imputation to handle the missing value.

Acknowledgement

Throughout the writing of this thesis, I have received great support and assistance.

I would like to thank Uppsala Monitoring Centre to provide the funding and position for this thesis project. It has been a wonderful experience to write my thesis here.

I want to thank my supervisor, Henric Taavola from Uppsala Monitoring Centre, for his guidance through each stage of the process. Your expertise was invaluable in formulating the research questions and critical steps. I really appreciate your time and patient support. I would also like to thank my other supervisor, Yukai Yang, from the statistics department, Uppsala University. Your professional suggestions pushed me to sharpen my thinking.

I want to thank my colleagues at Uppsala Monitoring Centre for their excellent collaboration. Christian Rausch, I want to thank you for your assistance in annotating the data. Without your help, I would not be able to perform my empirical analysis. Eva-Lisa Meldau, thank you for your insightful feedback which brought my work to a higher level. Jim Barrett, thank you for your time of sharing your machine learning analysis experience with me. Oskar Gauffin, thank you for your insights and explanations of the variables in the VigiBase.

Finally, I would like to thank my father. Because of your financial support, I can focus on completing my master thesis in Sweden instead of worrying about my finance, especially during the COVID time.

References

Awad, Mariette and Rahul Khanna (2015). "Support Vector Machines for Classification". In: *Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers*. Berkeley, CA: Apress, pp. 39–66. ISBN: 978-1-4302-5990-9.

Bartlett, Peter et al. (1998). "Boosting the margin: a new explanation for the effectiveness of voting methods". In: *The Annals of Statistics* 26.5, pp. 1651–1686.

Breiman, Leo (2001). "Random Forests". In: *Machine Learning* 45, pp. 5–32.

Breiman, Leo et al. (1984). *Classification and regression trees*. The Wadsworth & Brooks/Cole statistics/probability series. Monterey, CA: Wadsworth & Brooks/Cole Advanced Books & Software.

Chen, Tianqi and Carlos Guestrin (Aug. 2016). "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Friedman, Jerome (Feb. 2002). "Stochastic Gradient Boosting". In: *Computational Statistics & Data Analysis* 38, pp. 367–378.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2009). *The elements of statistical learning: data mining, inference and prediction*. 2nd ed. Springer.

Hsu, Chih-Wei, Chih-Chung Chang, and Chih-Jen Lin (2003). *A Practical Guide to Support Vector Classification*. Tech. rep. Department of Computer Science, National Taiwan University.

Lee, Tian-Shyug et al. (2006). "Mining the customer credit using classification and regression tree and multivariate adaptive regression splines". In: *Computational Statistics & Data Analysis* 50.4, pp. 1113–1130. ISSN: 0167-9473.

Loh, Wei-Yin (2014). "Fifty Years of Classification and Regression Trees". In: *International Statistical Review* 82, pp. 329–348.

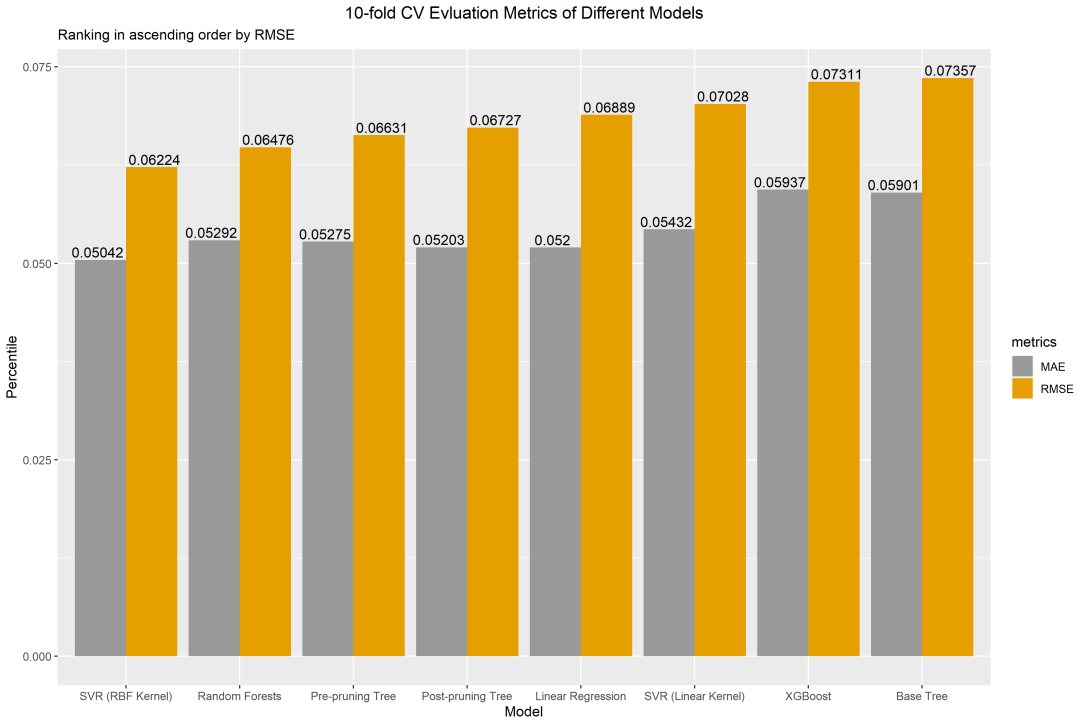
N.Morgan, James and John A. Sonquist (1963). "Problems in the Analysis of Survey Data, and a Proposal". In: *Journal of the American Statistical Association* 58, pp. 415–434.

Prasad, Anantha M., Louis R. Iverson, and Andy Liaw (2006). "Newer Classification and Regression Tree Techniques: Bagging and Random Forests for Ecological Prediction". In: *Ecosystems* 9, pp. 181–199.

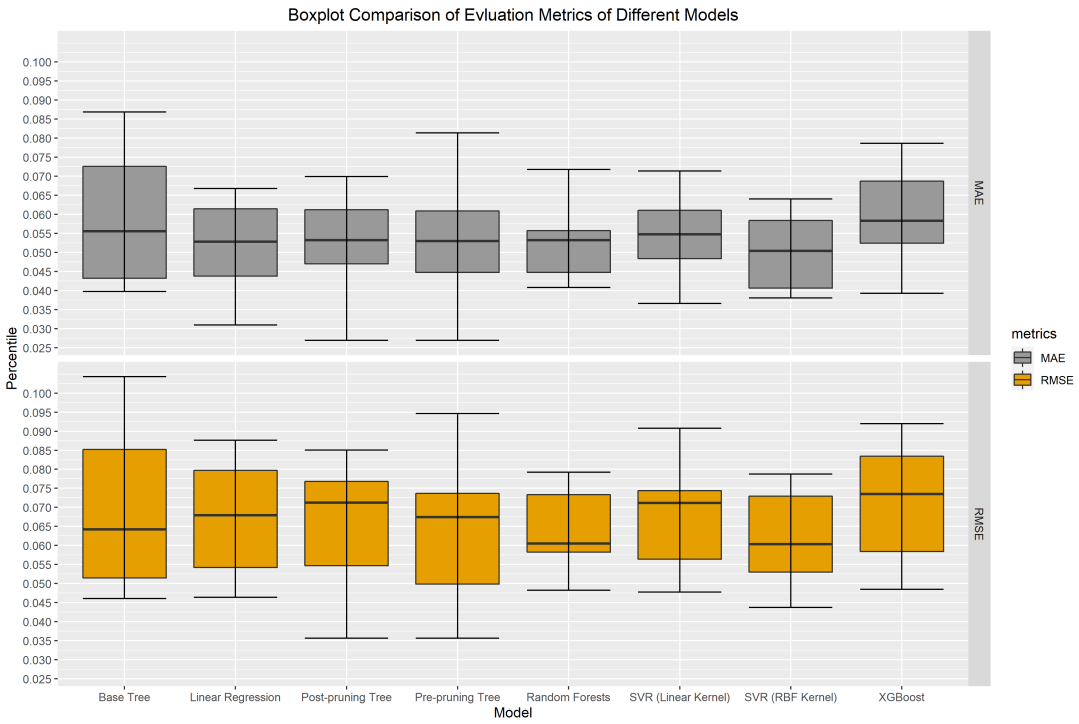
Smola, Alex J. and Bernhard Schölkopf (2004). "A tutorial on support vector regression". In: *Statistics and Computing* 14, pp. 199–222. ISSN: 1573-1375.

Appendix

Prediction evaluation figure



(a)



(b)

Figure 10: Comparison of 10-fold CV RMSE and MAE of Different Models

R code

```
library(tidyverse)
library(dplyr)
library(rpart)
library(rattle)
library(rpart.plot)
library(RColorBrewer)
library(randomForest)
library(xgboost)
library(e1071) # for SVR modelling
library(purrr) # for grid search data frame
library(ggplot2)

####---------------------- pre-setup ----------------------####
# get the percentile
df_pct = df_numeric %>%
  mutate(percentile = cut_index / real_n_indication) %>%
  select(-c(cut_index))
# 10 fold CV
n = nrow(df_pct)
k = 10
set.seed(2021)
folds = sample(rep(1:k,n/k),n, replace = F)
train_num = df_pct[folds != 1,]
test_num = df_pct[folds == 1,]

####---------------------- SVR ----------------------####

### kernal = linear

## an example of prediction

# use cross validation to find the best parameter
best_cost = tune(svm, percentile~., data = train_num, kernel = 'linear',
```

```
                    ranges = list(cost = 1*10^(-3:2),
                                  epsilon = c(0.01,0.1,1)))$best.parameters
svr_m = svm(percentile~., data = train_num,kernel = 'linear',
            cost = as.numeric(best_cost['cost']) ,
            epsilon = as.numeric(best_cost['epsilon']))
pred_svr_linear = predict(svr_m, test_num %>% select(-percentile) )


# CV for model comparison
svr_p = numeric()
rmse_svr_linear_i =c()
mae_svr_linear_i =c()
start_time_svr_linear <- Sys.time()
for (i in 1:k) {
  set.seed(1234)
  train = df_pct[folds != i,]
  test = df_pct[folds == i,]
  best_cost = tune(svm, percentile~., data = train, kernel = 'linear',
                   ranges = list(cost = 1*10^(-3:2),
                                 epsilon = c(0.01,0.1,1)))$best.parameters
  svr_m = svm(percentile~., data = train,kernel = 'linear',
              cost = as.numeric(best_cost['cost']) ,
              epsilon = as.numeric(best_cost['epsilon']))
  svr_p[folds == i] = predict( svr_m, select(test,-percentile))
  rmse_svr_linear_i[i] = caret::RMSE(test$percentile, svr_p[folds == i])
  mae_svr_linear_i[i] = caret::MAE(test$percentile, svr_p[folds == i])
}
end_time_svr_linear <- Sys.time()
grid_time_comsumption_svr_linear=end_time_svr_linear -
                                 start_time_svr_linear
ten_fold_RMSE_svr_linear = caret::RMSE(df_pct$percentile, svr_p)
ten_fold_MAE_svr_linear = caret::MAE(df_pct$percentile, svr_p)
ten_fold_RMSE_svr_linear

```

```
68
69
70  # kernal = radial
71  # an example of prediction
72
73  best_cost = tune(svm, percentile~., data = train_num, kernel = 'radial',
74                   ranges = list(cost = c(0.01,0.1,1,10,100),
75                                 gamma = c(0.01,0.1,1,10,100),
76                                 epsilon = c(0.01,0.1,1))) $best.parameters
77
78  svr_m = svm(percentile~., data = train_num,kernel = 'linear',
79              cost = as.numeric(best_cost['cost']) ,
80              epsilon = as.numeric(best_cost['epsilon']))
81  pred_svr_RBF = predict(svr_m, test_num %>% select(-percentile) )
82  pred_svr_RBF
83
84
85  # CV for model comparison
86  svr_p = numeric()
87  rmse_svr_rbf_i = c()
88  mae_svr_rbf_i = c()
89  start_time_svr_rbf <- Sys.time()
90  for (i in 1:k) {
91    set.seed(1234)
92    train = df_pct[folds != i,]
93    test = df_pct[folds == i,]
94    best_cost = tune(svm, percentile~., data = train_num,
95                     kernel = 'radial',
96                     ranges = list(cost = 1*10^(-3:2),
97                                   epsilon = c(0.01,0.1,1),
98                                   gamma = c(0.01,0.1,1,10,100)))$best.parameters
99    svr_m = svm(percentile~., data = train,kernel = 'radial',
100             cost = as.numeric(best_cost['cost']) ,
101             epsilon = as.numeric(best_cost['epsilon']))
```

```
  svr_p[folds == i] = predict( svr_m, select(test,-percentile))
  rmse_svr_rbf_i[i] = caret::RMSE(test$percentile, svr_p[folds == i])
  mae_svr_rbf_i[i] = caret::MAE(test$percentile, svr_p[folds == i])
}
end_time_svr_rbf <- Sys.time()
grid_time_comsumption_svr_rbf_10CV = end_time_svr_rbf - start_time_svr_rbf

ten_fold_RMSE_svr_rbf = caret::RMSE(df_pct$percentile, svr_p)
ten_fold_MAE_svr_rbf = caret::MAE(df_pct$percentile, svr_p)
ten_fold_RMSE_svr_rbf



####---------------------- Linear Model ----------------------####
# An example
linear_model = lm(percentile~., data = train_num)
# prediction on the validation set
pred_lr = predict(linear_model, test_num )
pred_lr

# CV for model comparison
lr_p = numeric()
rmse_lr_i = c()
mae_lr_i = c()
for (i in 1:k) {
  set.seed(1234)
  train = df_pct[folds != i,]
  test = df_pct[folds == i,]
  lm = lm(percentile~., data = train)
  lr_p[folds == i] = predict( lm,test)
  rmse_lr_i[i] = caret::RMSE(test$percentile, lr_p[folds == i])
  mae_lr_i[i] = caret::MAE(test$percentile, lr_p[folds == i])
}
ten_fold_RMSE_lr = caret::RMSE(df_pct$percentile, lr_p)
```

```
ten_fold_MAE_lr = caret::MAE(df_pct$percentile, lr_p)
ten_fold_RMSE_lr
ten_fold_RMSE_lr == mean(rmse_lr_i)



####------------------------ Regression Tree ------------------------####
# an example of prediction
# base tree
reg_tree0 = rpart(percentile~., data = train_num, method = 'anova',
                  control = rpart.control(cp = 0,minbucket = 2))
# result

png('example_base_tree.png',width = 1189, height = 679, units = "px")
rpart.plot(reg_tree0, type = 3, digits = 3, fallen.leaves = TRUE)
dev.off()
printcp(reg_tree0)
png('example_cp_base_tree.png',width = 1078, height = 646, units = "px")
plotcp(reg_tree0)
dev.off()
# prediction
pred_rt_base <- predict(reg_tree0, test_num)

# pre-pruning tree with the CV best parameter
best_par = tune.rpart( percentile~., data = train_num,
                       minsplit = c( 6, 9, 12, 21),
                       minbucket = (c(2,3,4,7)),
                       maxdepth = seq(1,10, by = 2) )$best.parameters
pre_pruned_m = rpart(percentile~., data = train_num, method = 'anova',
                     control = rpart.control(minbucket = as.numeric(best_par['minbucket']),minsplit =
as.numeric(best_par['minsplit']),maxdepth = as.numeric(best_par['maxdepth']),cp = 0.01))
# pre-pruning with minbucket = 2,minsplit = 10,maxdepth = 3,cp = 0.01
```

```
pre_pruned_m = rpart(percentile~., data = train_num, method = 'anova',
                     control = rpart.control(minbucket = 2,
                                             minsplit = 10,
                                             maxdepth = 3,cp = 0.01))
pred_rt_prep <- predict(pre_pruned_m, test_num)

# example results

png('pre_pruning_2_10_0.01_3.png',width = 480, height = 480, units = "px")
rpart.plot(pre_pruned_m, type = 2, digits = 3, fallen.leaves = TRUE)
dev.off()
printcp(pre_pruned_m)
# plotcp(pre_pruned_m)

# post-pruning pree
cp_best = reg_tree0$cptable[which.min(reg_tree0$cptable[,"xerror"]),"CP"]
post_pruned_m = prune(reg_tree0,cp = cp_best,minbucket = 2 )# no max maxdepth
png('post_pruning.png')
rpart.plot(post_pruned_m, digits = 3, fallen.leaves = TRUE)
dev.off()
pred_rt_postp <- predict(post_pruned_m, test_num)
pred_rt_postp


# CV for model comparison
base_p = pre_pruned_p = post_pruned_p = numeric()
rmse_base_i = rmse_pre_pruned_i = rmse_post_pruned_i = numeric()
mae_base_i = mae_pre_pruned_i = mae_post_pruned_i = numeric()

start_time_rt <- Sys.time()
for (i in 1:k) {
  set.seed(1234)
  train = df_pct[folds != i,]
  test = df_pct[folds == i,]
```

```
203     # tree grow
204     base_m = rpart(percentile~., data = train, method = 'anova', control = rpart.control(cp = 0,minbucket = 2))
205     base_p[folds == i] <- predict(base_m, test)
206     rmse_base_i[i] = caret::RMSE(test$percentile, base_p[folds == i])
207     mae_base_i[i] = caret::MAE(test$percentile, base_p[folds == i])
208     # pre prune
209     best_par = tune.rpart( percentile~., data = train,
210                                 minsplit = c( 6, 9, 12, 21),
211                                 minbucket = (c(2,3,4,7)),
212                                 maxdepth = seq(1,10, by = 2) )$best.parameters
213     pre_pruned_m = rpart(percentile~., data = train, method = 'anova',
214                               control = rpart.control(minbucket = as.numeric(best_par['minbucket']),
215                                                       minsplit = as.numeric(best_par['minsplit']),
216                                                       maxdepth = as.numeric(best_par['maxdepth']),cp = 0.01))
217     pre_pruned_p[folds == i] <- predict(pre_pruned_m, test)
218     rmse_pre_pruned_i[i] = caret::RMSE(test$percentile, pre_pruned_p[folds == i])
219     mae_pre_pruned_i[i] = caret::MAE(test$percentile, pre_pruned_p[folds == i])
220     # post prune
221     cp_best = base_m$cptable[which.min(base_m$cptable[,"xerror"]),"CP"]
222     post_pruned_m = prune(base_m,cp = cp_best,minbucket = 2 )# no max maxdepth
223     post_pruned_p[folds == i] <- predict(post_pruned_m, test)
224     rmse_post_pruned_i[i] = caret::RMSE(test$percentile, post_pruned_p[folds == i])
225     mae_post_pruned_i[i] = caret::MAE(test$percentile, post_pruned_p[folds == i])
226 }
227 end_time_rt <- Sys.time()
228 grid_time_comsumption_rt_10CV = end_time_rt – start_time_rt
229 base_RMSE = caret::RMSE(df_pct$percentile, base_p)
230 base_MAE = caret::MAE(df_pct$percentile, base_p)
231 pre_pruned_RMSE = caret::RMSE(df_pct$percentile, pre_pruned_p)
232 pre_pruned_MAE = caret::MAE(df_pct$percentile, pre_pruned_p)
233 post_pruned_RMSE = caret::RMSE(df_pct$percentile, post_pruned_p)
234 post_pruned_MAE = caret::MAE(df_pct$percentile, post_pruned_p)
235 # retuen the averaged RMSE
236 ten_fold_rt = data.frame(base_RMSE, pre_pruned_RMSE, post_pruned_RMSE)
```

```
ten_fold_rt_mae = data.frame(base_MAE, pre_pruned_MAE, post_pruned_MAE)
ten_fold_rt


# Grid search set up of regression tree
gs_rt <– list( minsplit = c( 6, 9, 12, 21),
                  minbucket = (c(2,3,4,7)),
                  maxdepth = seq(1,10, by = 2)) %>%
  cross_df() # Convert to grid data frame
gs_rt

####–––––––––––––––––––––––– Random Forests ––––––––––––––––––––––––####
# an example of prediction
best_par = tune.randomForest(percentile~., data = train_num,
                                  mtry =seq(1,8),
                                  ntree = seq(1,200,by =10),
                                  importance = T) $best.parameters
rf_m = randomForest(percentile~., data = train_num,
                        mtry = as.numeric(best_par['mtry']),
                        ntree = as.numeric(best_par['ntree']),importance = T)
pred_rf = predict( rf_m, select(test_num,–percentile))
pred_rf

# feature importance

importance(rf_m)
png('example_feature_importance.png',width = 1508, height = 866, units = "px")
varImpPlot(rf_m,main = 'Feature Importance',)
dev.off()

# CV for model comparison
rf_p = numeric()
rmse_rf_i= numeric()
mae_rf_i= numeric()
```

```
start_time_rf = Sys.time()
for (i in 1:k) {
  set.seed(1234)
  train = df_pct[folds != i,]
  test = df_pct[folds == i,]
  best_par = tune.randomForest(percentile~., data = train,
                               mtry =seq(1,8), ntree = seq(1,200,by =10),
                               importance = T) $best.parameters
  rf_m = randomForest(percentile~., data = train,
                      mtry = as.numeric(best_par['mtry']),
                      ntree = as.numeric(best_par['ntree']),importance = T)
  rf_p[folds == i] = predict( rf_m, select(test,–percentile))
  rmse_rf_i[i] = caret::RMSE(test$percentile, rf_p[folds == i])
  mae_rf_i[i] = caret::MAE(test$percentile, rf_p[folds == i])
}
end_time_rf = Sys.time()
grid_time_comsumption_rf_10CV = end_time_rf – start_time_rf
ten_fold_RMSE_rf = caret::RMSE(df_pct$percentile, rf_p)
ten_fold_MAE_rf = caret::MAE(df_pct$percentile, rf_p)
ten_fold_RMSE_rf

# Grid search set up of Random Forests
gs_rf <– list(mtry =seq(1,8),
              ntree = seq(1,200,by =10)) %>%
  cross_df() # Convert to data frame grid
gs_rf


####–––––––––––––––––––––– XGBoost ––––––––––––––––––––––––####

# an example of prediction
train_x_num = data.matrix(select(train_num,–percentile))
train_y_num = train_num$percentile

```

```
test_x_num = data.matrix(select(test_num,−percentile))
test_y_num = test_num$percentile

xgb_train_num = xgb.DMatrix(data = train_x_num, label = train_y_num)
xgb_test_num = xgb.DMatrix(data = test_x_num, label = test_y_num)

# train model
xgb_num = xgboost::xgboost(data = xgb_train_num, max.depth = 5, nrounds = 100, eta = 0.9,
                           nthread = 2,early_stopping_rounds = 6)
print(xgb_num)
pred_xgb = predict(xgb_num, xgb_test_num)
pred_xgb
#view variable importance plot
mat <− xgb.importance (feature_names = colnames(train_x_num),model = xgb_num)
png('xgb_feature_importance.png',width = 1267, height = 829, units = "px",type = 'windows')
xgb.plot.importance (importance_matrix = mat[1:12],xlab = 'Gain', main ='Feature Importance' )
dev.off()
# CV for model comparison
# grid search three parameters with xgb.cv
gs <− list(eta =seq(0,0.3, by = 0.05 ),
           gamma = seq(0,80, by = 10),
           subsample = seq(0,0.5, by = 0.1)) %>%
  cross_df() # Convert to data frame grid
gs

grid_search_xgb = function(input_data,gs_df){
  best_rmse = numeric()
  start_time <− Sys.time()
  best_n_rounds = numeric()
  for (b in 1:nrow(gs_df)) {
    params <− list(booster = "gbtree", objective = "reg:squarederror",
                   eta=gs[b,]$eta, gamma=gs[b,]$gamma,
                   max_depth=4, subsample=gs[b,]$subsample,
                   colsample_bytree=1)
```

```
    xgbcv = xgb.cv( params = params,
                    data = input_data,
                    nrounds = 150,
                    nfold = 10,
                    showsd = T, stratified = T,
                    print_every_n = 10,
                    early_stop_round = 4,
                    maximize = F, metrics = "rmse")
    # best_n_rounds[b] = which.min(xgbcv$evaluation_log$test_rmse_mean)
    best_rmse[b] = min(xgbcv$evaluation_log$test_rmse_mean)
  }
  end_time <- Sys.time()
  return(tibble('best_parameter' = gs_df[which.min(best_rmse),],
                'best_rmse' =min(best_rmse),
                # 'best_iteration' =best_n_rounds[which.min(best_rmse)] ,
                'time_consumption' = end_time - start_time,
  ))
}



## 10-fold CV RMSE with grid searching
xgb_p = numeric()
rmse_xgb_i = c()
mae_xgb_i = c()
start_time_xgb <- Sys.time()
for (i in 1:k) {
  set.seed(1234)
  train = df_pct[folds != i,]
  test = df_pct[folds == i,]
  # prepare data
  train_x_num = data.matrix(select(train,-percentile))
  train_y_num = train$percentile
  test_x_num = data.matrix(select(test,-percentile))
```

```
  test_y_num = test$percentile

  xgb_train_num = xgb.DMatrix(data = train_x_num, label = train_y_num)
  xgb_test_num = xgb.DMatrix(data = test_x_num, label = test_y_num)
  # model
  # parameter grid searching
  gs_info = grid_search_xgb(input_data = xgb_train_num, gs_df = gs)
  # find it the xgb with best parameters
  xgb_m = xgboost::xgboost(data = xgb_train_num, max.depth = 4, nrounds = 150,
                           eta = gs_info$best_parameter$eta,gamma = gs_info$best_parameter$gamma,
                           nthread = 4,early_stopping_rounds = 3,
                           subsample=gs_info$best_parameter$subsample)
  xgb_p[folds == i] = predict( xgb_m, xgb_test_num)
  rmse_xgb_i[i] = caret::RMSE(test$percentile, xgb_p[folds == i])
  mae_xgb_i[i] = caret::MAE(test$percentile, xgb_p[folds == i])
}
end_time_xgb <- Sys.time()
# runningf time
grid_time_comsumption_xgb_10cv = end_time_xgb - start_time_xgb
ten_fold_RMSE_xgb = caret::RMSE(df_pct$percentile, xgb_p)
ten_fold_MAE_xgb = caret::MAE(df_pct$percentile, xgb_p)
ten_fold_RMSE_xgb


####---------- make the RMSE comparison chart ----------####
library(ggplot2)
RMSE_compare = cbind(ten_fold_RMSE_lr,
                     ten_fold_RMSE_svr_linear,
                     ten_fold_RMSE_svr_rbf,
                     ten_fold_rt,
                     ten_fold_RMSE_rf,
                     ten_fold_RMSE_xgb)
# round to 5 digits
is.num <- sapply(RMSE_compare, is.numeric)
```

```
RMSE_compare[is.num] <- lapply(RMSE_compare[is.num], round, 5)
RMSE_compare = tibble(Model = c('Linear Regression','SVR (Linear Kernel)','SVR (RBF Kernel)',
                                'Base Tree','Pre-pruning Tree','Post-pruning Tree',
                                'Random Forests','XGBoost'),
                      'RMSE' = as.numeric(RMSE_compare[1,]))

RMSE_compare = RMSE_compare %>% arrange(RMSE)

ggplot(RMSE_compare) +
  geom_bar(aes(x =reorder(Model, RMSE), y = RMSE),stat="identity",position = 'dodge') +
  # make the number show up above the bar
  geom_text(aes(x =reorder(Model, RMSE), y = RMSE,label=RMSE),
            position=position_dodge(width=0.9),
            vjust=-0.25) +
  labs(title = "10-fold CV RMSE of Different Models",
       subtitle = "Ranking in ascending order",
       x = "Model",
       y = "RMSE of Predicted Percentile")+
  # change title position
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_y_continuous(breaks=seq(0,1,0.005))
ggsave("10-fold_CV_RMSE_of_Different_Models.png", width = 30, height = 20, units = "cm")



####------------------- make the MAE comparison chart --------------------####
MAE_compare = cbind(ten_fold_MAE_lr,
                    ten_fold_MAE_svr_linear,
                    ten_fold_MAE_svr_rbf,
                    ten_fold_rt_mae,
                    ten_fold_MAE_rf,
                    ten_fold_MAE_xgb)
# round to 5 digits
is.num <- sapply(MAE_compare, is.numeric)
```

```
MAE_compare[is.num] <- lapply(MAE_compare[is.num], round, 5)
MAE_compare = tibble(Model = c('Linear Regression','SVR (Linear Kernel)','SVR (RBF Kernel)',
                               'Base Tree','Pre-pruning Tree','Post-pruning Tree',
                               'Random Forests','XGBoost'),
                     'MAE' = as.numeric(MAE_compare[1,]))

MAE_compare = MAE_compare %>% arrange(MAE)

ggplot(MAE_compare) +
  geom_bar(aes(x =reorder(Model, MAE), y = MAE),stat="identity",position = 'dodge') +
  # make the number show up above the bar
  geom_text(aes(x =reorder(Model, MAE), y = MAE,label=MAE),
            position=position_dodge(width=0.9),
            vjust=-0.25) +
  # guides(fill=FALSE) + # use this if changing the bar color
  labs(title = "10-fold CV MAE of Different Models",
       subtitle = "Ranking in ascending order",
       x = "Model",
       y = "MAE of Predicted Percentile")+
  # change title position
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_y_continuous(breaks=seq(0,1,0.005))
ggsave("10-fold_CV_MAE_of_Different_Models.png", width = 30, height = 20, units = "cm")


####---------------- make the RMSE & MAE comparison chart ----------------####
# The color-blind friendly palette begins with grey:
cbPalette <- c("#999999", "#E69F00", "#56B4E9",
               "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC79A7")

metrics_compare = left_join(RMSE_compare,MAE_compare, by = "Model")
metrics_compare = metrics_compare %>% tidyr::gather(c("RMSE","MAE"),
                                                    key = 'metrics',
                                                    value = 'value')
```

```
475
476 metrics_compare %>%
477   # arrange(metrics,value)%>%
478   mutate(Model = factor(Model, levels = unique(Model))) %>%
479   ggplot() +
480   geom_bar(aes(x =Model, y = value,fill = metrics),stat="identity",
481            position = 'dodge') +
482   # make the number show up above the bar
483   geom_text(aes(x =Model, y = value,label=round(value,5),group = metrics),
484             position=position_dodge(width=1), vjust=-0.25) +
485   # guides(fill=FALSE) + # use this if changing the bar color
486   scale_fill_manual(values = c(cbPalette[1], cbPalette[2]))+
487   labs(title = "10-fold CV Evluation Metrics of Different Models",
488        subtitle = "Ranking in ascending order by RMSE",
489        x = "Model",
490        y = "Percentile")+
491   # change title position
492   theme(plot.title = element_text(hjust = 0.5)) +
493   scale_y_continuous(breaks=seq(0,1,0.025))
494
495 ggsave("10-fold_CV_Evluation_Metrics_of_Different_Models.png", width = 30, height = 20, units = "cm")
496 saveRDS(metrics_compare,"metrics_compare.rds")
497
498
499 ####---------- make prediction comparison chart ----------####
500
501
502 # find the drug name
503 all_drug_names = m_df$base_composition_name[ 1:60]
504 test_drug_names = all_drug_names[folds==1]
505 # find the test label
506 test_label = test_num$percentile
507 # pred_xgb = rep(0,length(test_drug_names))
508 prediction_compare = cbind(test_label,
```

```
                                pred_lr,pred_rf,
                                pred_rt_base,
                                pred_rt_postp,
                                pred_rt_prep,
                                pred_svr_linear,
                                pred_svr_RBF,pred_xgb)
prediction_compare = as_tibble(prediction_compare)
# round to 5 digits
is.num <- sapply(prediction_compare, is.numeric)
prediction_compare[is.num] <- lapply(prediction_compare[is.num], round, 5)
# put in the column of drug name
prediction_compare = prediction_compare %>% mutate(Drug_name = test_drug_names)
# delete drug name
column_gather = colnames(prediction_compare)[- length(colnames(prediction_compare))]
column_gather
prediction_compare = prediction_compare %>% tidyr::gather(column_gather, key = 'Model',value = 'value')

# 1. baseline models prediction
base_line_pred = prediction_compare %>% filter(Model %in% c('test_label',
                                                            'pred_lr','pred_svr_linear',
                                                            'pred_svr_RBF'))
base_line_name = (base_line_pred %>%
                    select(Model) %>%
                    distinct())$Model
base_line_name
# make the plot
ggplot(base_line_pred) +
  geom_line(aes(x = Drug_name, y = value,
                group = Model, color = Model,size = Model))+
  labs(#title = "Prediction Comparison of Baseline Models",
    x = "Drug Name",
    y = "Cut Percentile",
    color = "Model Name and Test Label") +
  # change title position
```

```
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_color_manual(labels = c( "Linear Regression", "SVR (linear kernel)",
                                 "SVR (RBF kernel)","Test Label"),
                     values = cbPalette[c(2:length(base_line_name),1)])+
  scale_y_continuous(breaks=seq(0,1,0.005))+
  scale_size_manual(values = c(rep(0.5,3),1),guide = 'none')
ggsave("pred_base_percentile.png", width = 30, height = 20, units = "cm")

saveRDS(base_line_pred,"base_line_pred.rds")
# 2. Tree-based models prediction
tree_pred = prediction_compare %>% filter(Model %in% c('test_label','pred_rf',
                                                       'pred_rt_base','pred_rt_prep',
                                                       'pred_rt_postp','pred_xgb'))
tree_name = (tree_pred %>% select(Model) %>% distinct())$Model
tree_name


ggplot(tree_pred) +
  geom_line(aes(x = Drug_name, y = value,group = Model,color = Model,size = Model))+
  labs(#title = "Prediction Comparison of Tree-based Models",
    x = "Drug Name",
    y = "Cut Percentile",
    color = "Model Name and Test Label") +
  # change title position
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_color_manual(labels = c( "Random Forests", "Base Regression Tree",
                                 "Post-pruning Regression Tree", "Pre-pruning Regression Tree",
                                 "XGboost","Test Label"),
                     values = cbPalette[c(2:length(tree_name),1)])+
  scale_y_continuous(breaks=seq(0,1,0.005))+
  scale_size_manual(values = c(rep(0.5,5),1),guide = 'none')

ggsave("Prediction_Comparison_of_Tree-based_Models.png", width = 30, height = 20, units = "cm")
```

```
saveRDS(tree_pred,"tree_pred.rds")
####---------------------- make box plot (RMSE) ----------------------####
RMSE_compare_box = cbind(rmse_lr_i,
                         rmse_svr_linear_i,
                         rmse_svr_rbf_i,
                         rmse_base_i,
                         rmse_pre_pruned_i,
                         rmse_post_pruned_i,
                         rmse_rf_i,rmse_xgb_i)
RMSE_compare_box = as_tibble(RMSE_compare_box)
model_names = c('Linear Regression',
                'SVR (Linear Kernel)','SVR (RBF Kernel)',
                'Base Tree','Pre-pruning Tree','Post-pruning Tree',
                'Random Forests','XGBoost')
colnames(RMSE_compare_box) = model_names

RMSE_compare_box
saveRDS(RMSE_compare_box,"RMSE_compare_box.rds")


m <- apply(RMSE_compare_box, MARGIN = 2, FUN = range, na.rm = TRUE)
dff_range = m[2,] - m[1,]
# set the order of model in x-axis
o <- order(dff_range, decreasing = FALSE)
o
png('boxplot_RMSE.png',width = 1189, height = 679, units = "px",type = 'windows')
boxplot(RMSE_compare_box[, o],ylab = 'RMSE', ylim = c(0.0,0.11),
        main = 'Ranking by range difference of RMSE in ascending order ')
dev.off()

####---------------------- make box plot (MAE)----------------------####
# mae_xgb_i = rep(0,k)
MAE_compare_box = cbind(mae_lr_i,
                        mae_svr_linear_i,
```

```
                        mae_svr_rbf_i,
                        mae_base_i,
                        mae_pre_pruned_i,
                        mae_post_pruned_i,
                        mae_rf_i,mae_xgb_i)
# round to two digits
MAE_compare_box = as_tibble(MAE_compare_box)
model_names = c('Linear Regression',
                'SVR (Linear Kernel)','SVR (RBF Kernel)',
                'Base Tree','Pre-pruning Tree','Post-pruning Tree',
                'Random Forests','XGBoost')
colnames(MAE_compare_box) = model_names

MAE_compare_box
saveRDS(MAE_compare_box,"MAE_compare_box.rds")

m <- apply(MAE_compare_box, MARGIN = 2, FUN = range, na.rm = TRUE)
dff_range = m[2,] - m[1,]
# set the order of model in x-axis
o <- order(dff_range, decreasing = FALSE)
o
png('boxplot_mae.png',width = 1189, height = 679, units = "px",type = 'windows')
boxplot(MAE_compare_box[, o],ylab = 'MAE', ylim = c(0.0,0.11),
        main = 'Ranking by range difference of MAE in ascending order ')
dev.off()



####---------------------- make box plot (RMSE & MAE)----------------------####
# combine the two boxplots in one figure
metrics_compare_boxplot = rbind(RMSE_compare_box %>% mutate('metrics' = 'RMSE') %>%
                                  tidyr::gather(model_names, key = 'Model',value = 'value'),
                                MAE_compare_box %>% mutate('metrics' = 'MAE') %>%
                                  tidyr::gather(model_names, key = 'Model',value = 'value')
```

```
)


metrics_compare_boxplot %>%
  ggplot(aes(x =Model, y = value,fill = metrics)) +
  geom_boxplot() +
  stat_boxplot(geom='errorbar')+
  facet_grid(metrics~.)+
  scale_fill_manual(values = c(cbPalette[1], cbPalette[2]))+
  labs(title = "Boxplot Comparison of Evluation Metrics of Different Models",
       x = "Model",
       y = "Percentile")+
  # change title position
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_y_continuous(breaks=seq(0,0.1,0.005))

ggsave("boxplot_comparison_MAR_RMSE.png", width = 30, height = 20, units = "cm")
```