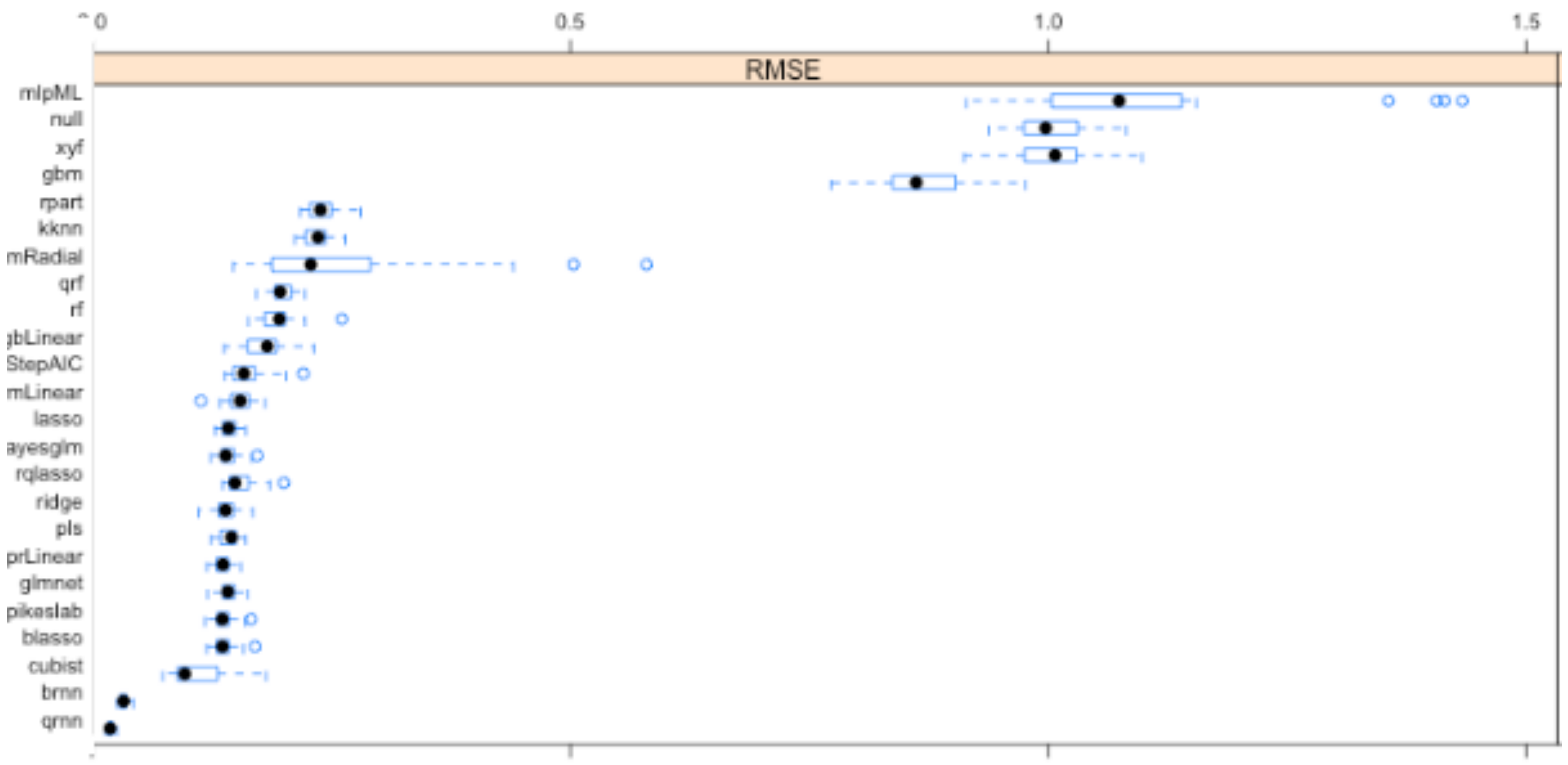
Report of Assignment 3

Zhen Huang

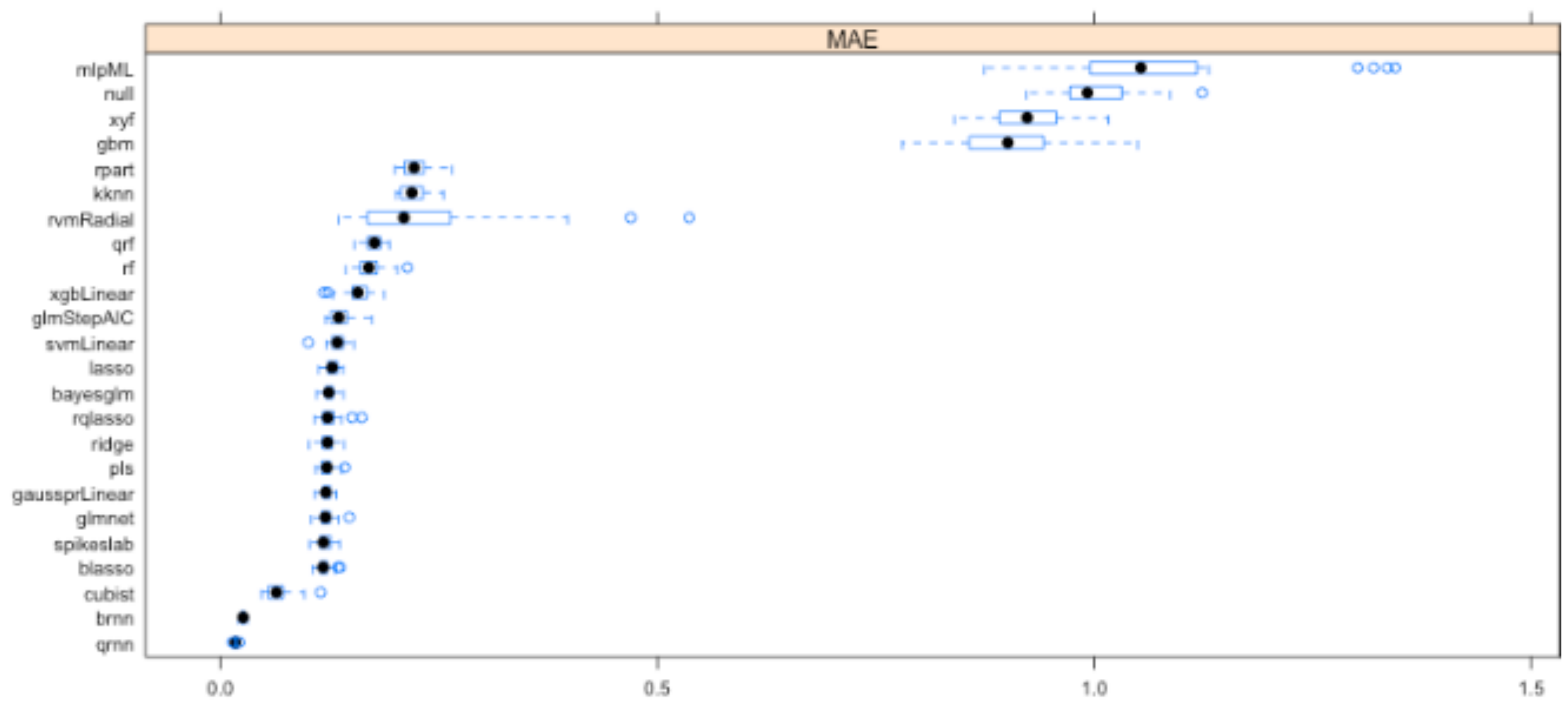
4/29/2020

1. Candidate Models

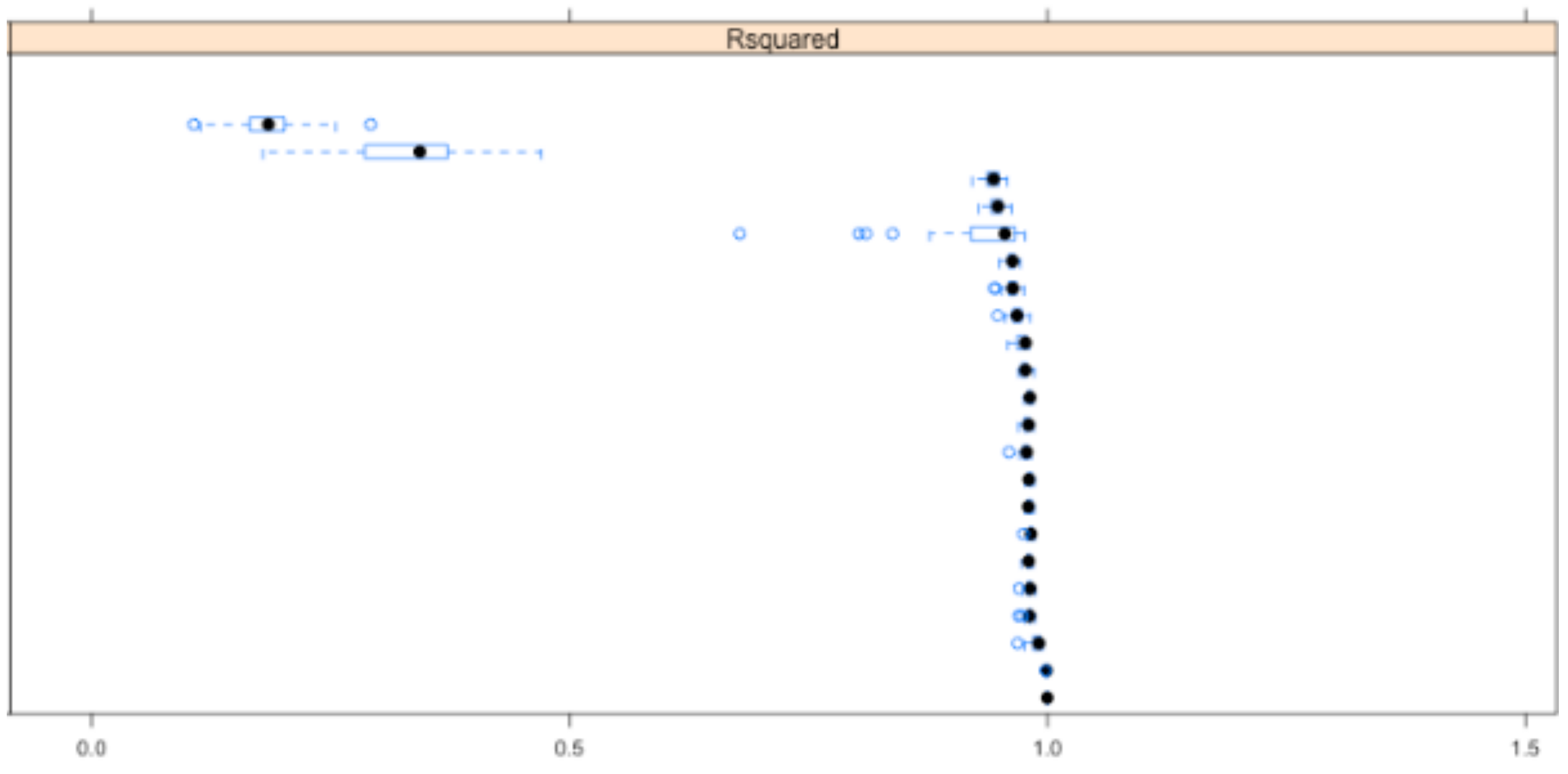
Twenty-three methods were trained with the data set and formed to be the candidate models selection shown below. Two methods are not included for avoiding to overload large rds files into memory. One of them is a `pcaNNet` model which performed worse than the Null model, the other is a `kernelpls` model which had resampled RMSE = 90.73.



RMSE of candidate models



MAE of candidate models



Rsquared of candidate models

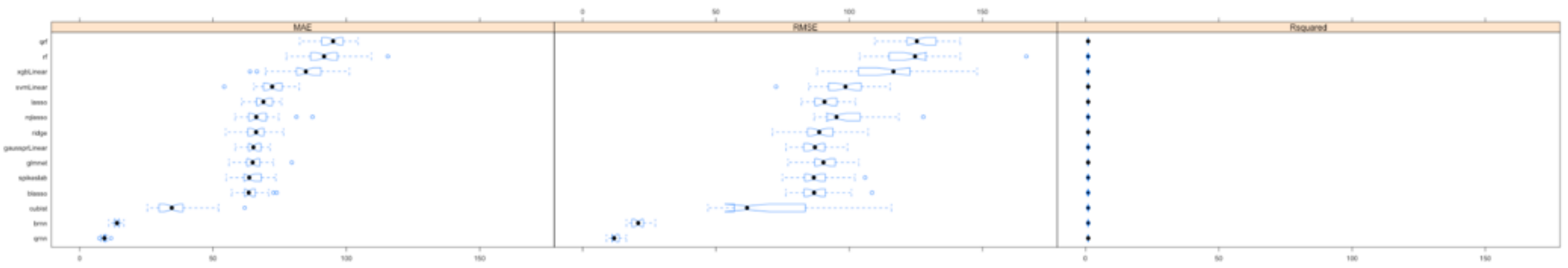
1.1 Table of candidate models

Choices	Model	Pre-processing	RMSE	Hyperparameters	Training Time
*Best candidate model	qrnn	“bagimpute”,“center”,“scale”,“dummy”	12.39	n.hidden=3, penalty=0 and bag=FALSE	1412.66 sec
	brnn	“bagimpute”,“center”,“scale”,“dummy”	21.31	neurons=4	349.26 sec
	cubist	“bagimpute”,“dummy”	68.57	committees=81, neighbors=5	340.89 sec
	gaussprLinear	“bagimpute”,“center”,“scale”,“pca”,“dummy”	85.35	No hyperparameter	35.23 sec
Trained after finding brnn	blasso	“knnimpute”,“center”,“scale”,“dummy”	87.54	sparsity=0.29	44.89 sec
Trained after finding brnn	bayesglm	“knnimpute”,“center”,“scale”,“dummy”	91.37	No hyperparameter	42.64 sec
Trained after finding qrnn	spikeslab	“knnimpute”,“center”,“scale”,“dummy”	87.86	vars=16	59.31 sec
Trained after finding qrnn	rqlasso	“knnimpute”,“center”,“scale”,“dummy”	93.17	lambda=0.01	307.61 sec
Supplied	GLMnet	“bagimpute”,“Yeojohnson”,“dummy”,“center”,“scale”	90.58	alpha=0.99, lambda=2.97	341.07s
Supplied	PLS	“bagimpute”,“YeoJohnson”,“center”,“scale”,“dummy”	90.73	ncomp 13	118.49 sec
	kernelpls	“knnimpute”,“center”,“scale”	95.94	ncomp=12	136.75 sec
Supplied	Rpart	“center”,“scale”	152.68		Normalise could improve predict accuracy
	rf	“knnimpute”	124.80	mtry 14	55.22 sec
	lasso	“bagimpute”,“Yeojohnson”,“center”,“scale”,“dummy”	90.36	fraction=0.79	63.66 sec
	ridge	“bagimpute”,“Yeojohnson”,“center”,“scale”,“dummy”	89.45	lambda=0.01056009	150.89 sec
	xgbLinear	“knnimpute”	114.77	119.05 sec	
	glmboost	“knnimpute”,“dummy”	94.44	mstop=532 and prune=yes	78.09 sec
	glmStepAIC	“knnimpute”,“center”,“scale”,“dummy”	103.34	No hyperparameter	11.57 sec

svmLinear	“knnimpute”,“center”,“scale”,“dummy”	98.49	C=0.07086818	254.67 sec
rvmRadial	“knnimpute”,“center”,“scale”,“dummy”	169.75	sigma=0.005040349	122.23 sec
kknn	“knnimpute”,“pca”,“dummy”	150.02	kmax=51, distance=1.006154 and kernel=inv	34.66 sec
gbm	“knnimpute”,“center”,“scale”,“dummy”	shrinkage=0.33, interaction.depth=1, n.minobsinnode=20, n.trees=321	550.26	114.77 sec
xyf	“knnimpute”,“dummy”	702.31	xdim=3, ydim=14, user.weights=0.875 and topo=rectangular	30.25 sec
mlpKerasDropout	“knnimpute”,“dummy”	680.52	75.25 sec	Not tolerant missing values
mlpML	“knnimpute”,“dummy”	702.31	layer1=2, layer2=5 and layer3=0	49.02 sec
pcaNNet	“knnimpute”,“dummy”	702.31	size=4 and decay=3.743782e-05	49.02 sec

2. The Best model and the other top models

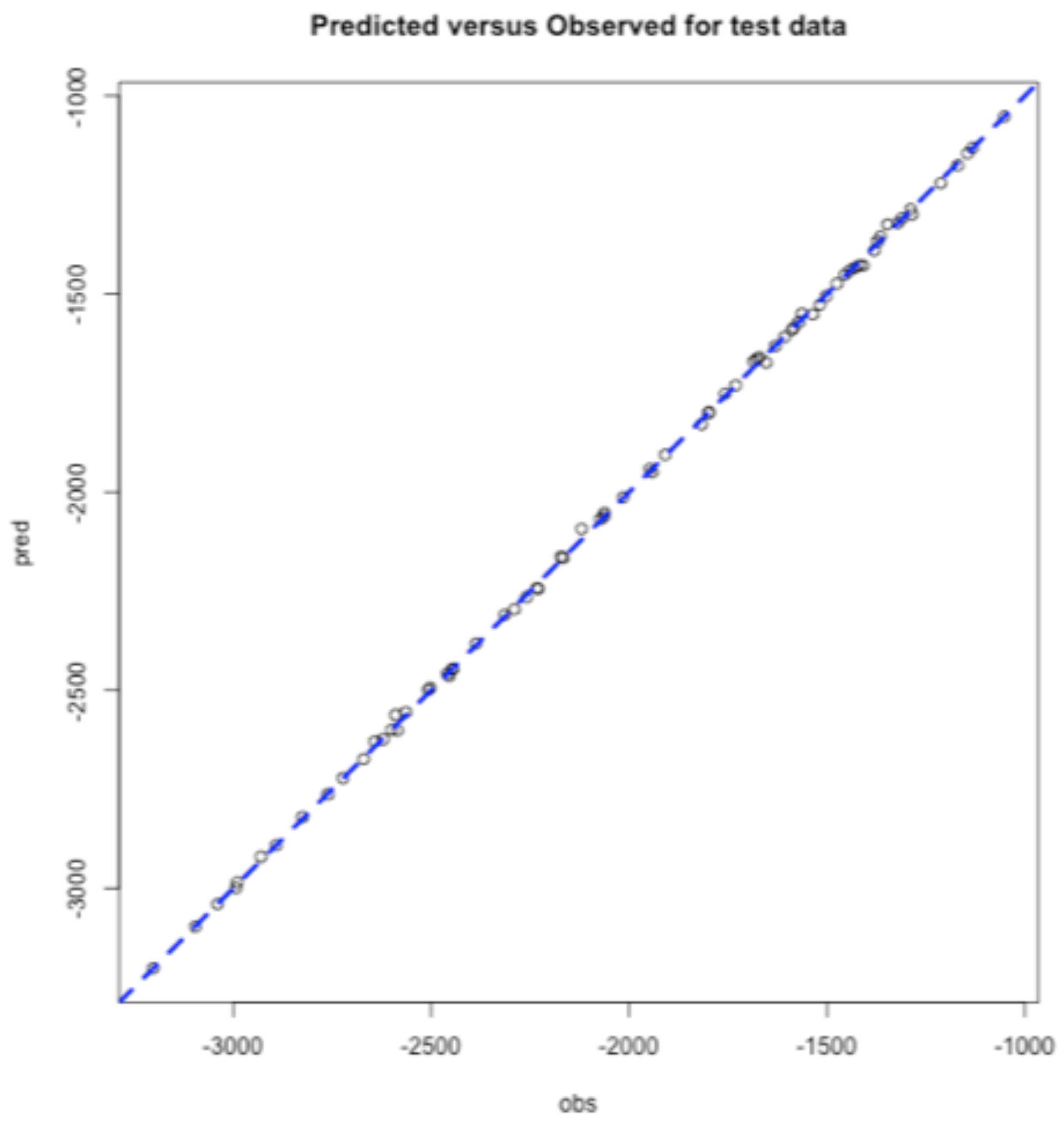
Assume “best” means “minimum RMSE”, the `qrnn` model performed better than any other candidate models, which had RMSE=12.39 from resampled performance validation. The screenshot below shows that `qrnn` had significantly good performance in terms of metrics without any overlap with the other models in candidate model group.



Model selection

Below are the statistics and visualisation for the best performing model based on test data.

```
Unseen data results for chosen model: qrnn
     RMSE  Rsquared       MAE
9.1621685 0.9997505 6.5061448
```



Screenshot of qrnn predicted test data

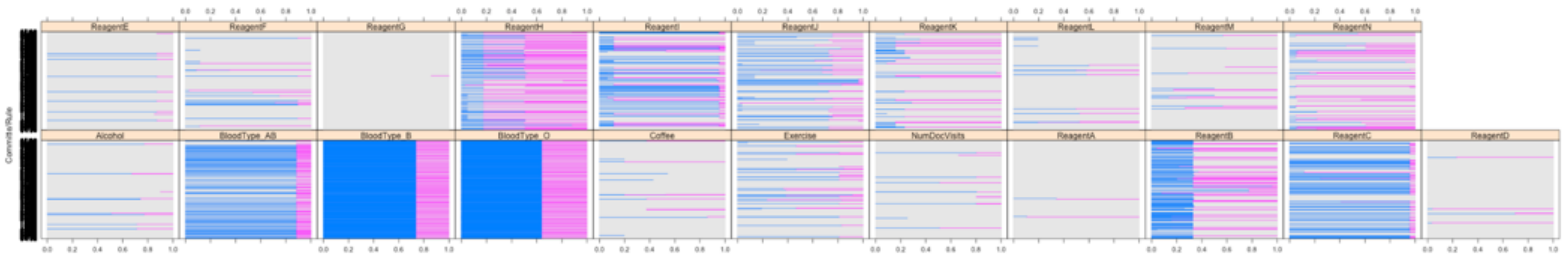
- **an explanation on how the best model works - do some research on your best performing method and hypothesize why it suits the data so well.**

Answer: Quantile regression neural network, `qrnn` , which were met luckly in this assignment, has some feature to significatly may improve its performance in this data set.

1. Using quantile instead of mean in estimating model parameters, may overcome the disadvantage of ordinal least square or other least square method. For example, more robust in facing of data outliers when fitting a model.
2. Three hidden layers of hyperparameter, which is not usual as described in `qrnn` package document, formed a complex non-linear model to predict mixed discrete-continuous variables in the data set.
3. Implementing bootstrap aggregation makes the ensembled model, may improve the accuracy of prediction.
4. Weight penalty regularization can be added to help avoid overfitting, but in our hyperparameter, penalty=0.

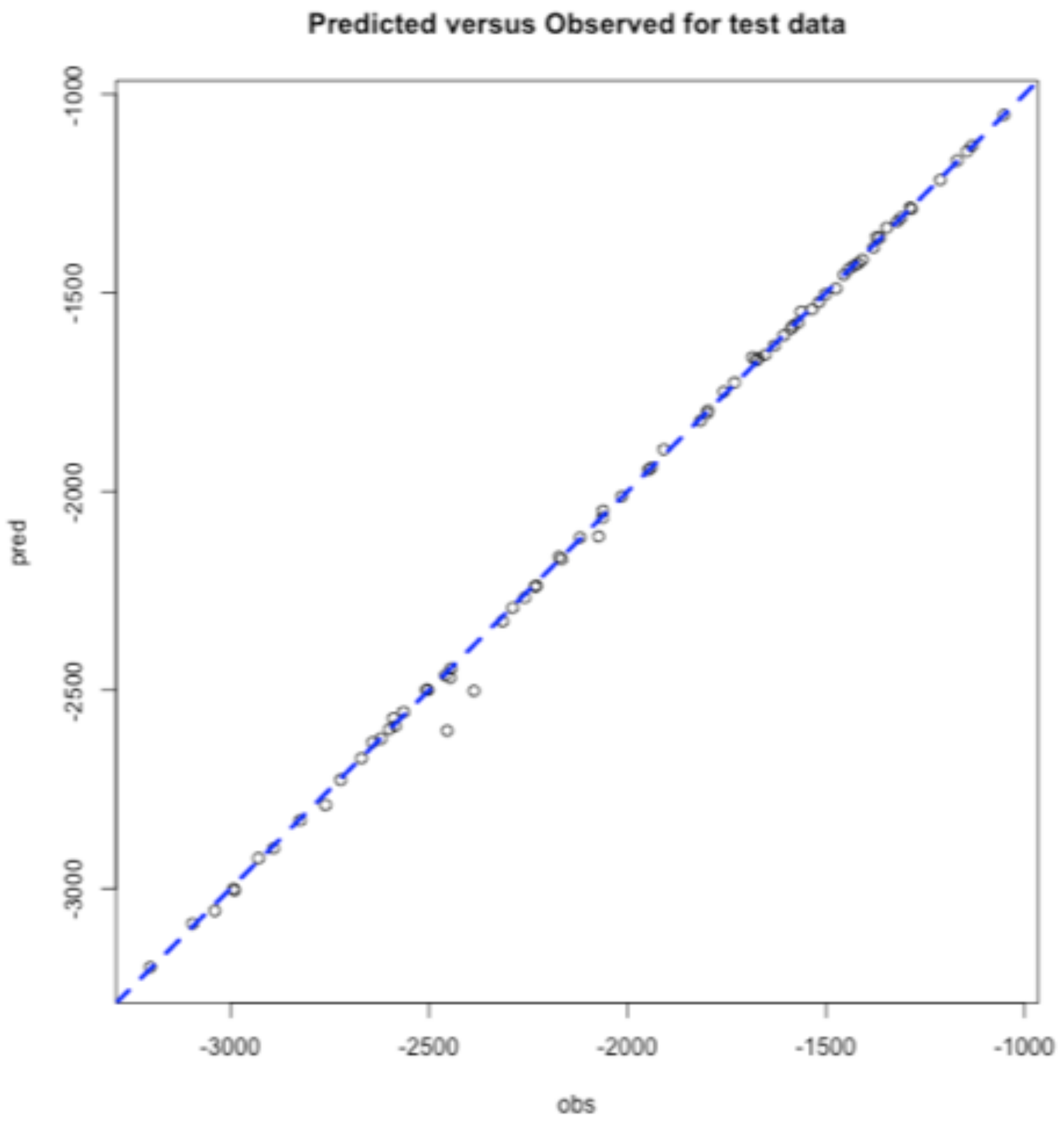
- **how would the optimum model change if transparency were very important?**

Answer: `cubist` would be the optimum medel if transparency are taken into account. As the sceenshot shows below, a tree-base structure of model may show the best split of each node. `rf` (random forest) also performed well in the test date set, the method also demestrated the features which are more important (Bloodtype). There is another finding that `rf` methods could have more accurate estimated RMSE by implement `oob` resampling method.



Variables splits in cubist model

```
Unseen data results for chosen model: cubist
      RMSE   Rsquared        MAE
23.6221164  0.9984519 10.1285528
```



Screenshot of brnn predicted test data

- **whether you recommend utilising an ensemble of some of the top models and why?**

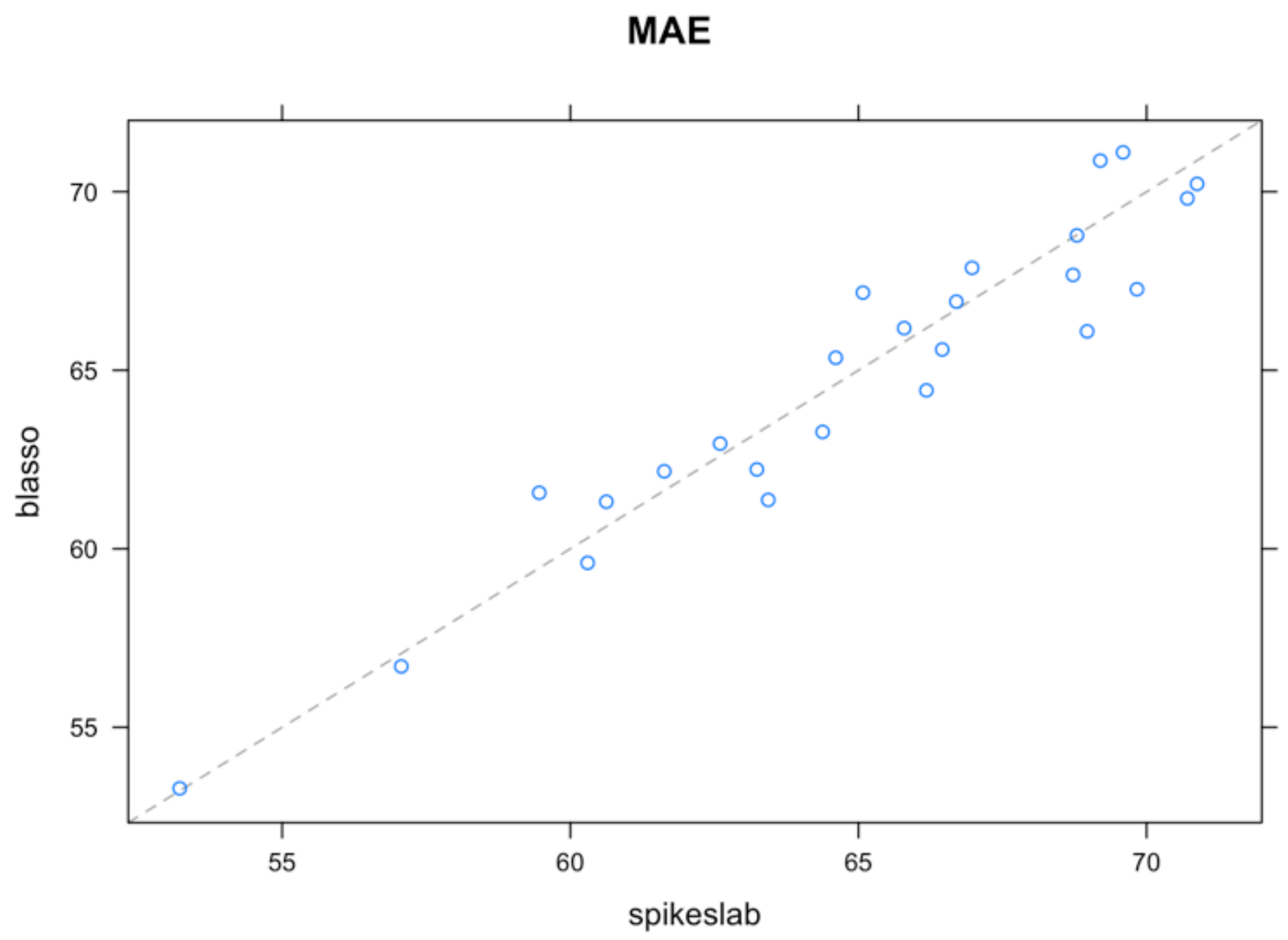
Answer: I would not recommend utilising an ensemble of top models in the group for three reasons.

1. Ensembling lower performance model with a high model would not improve the final performance. Here `qrnn` and the rest of the top models had a gap in terms of performance metrics. So, we may not get better result than just using single `qrnn`
2. `qrnn` or `brnn` are both neural network mothods, which means they are already very complex models. If they are ensembled, we are running the risk of overfitting the model.
3. The rest of the top models had related predict results (or similar method to each other), such as blasso and ridge, glmnet and gaussprLinear. Similar methods or methods which have correlated results may not improve the final predict accuracy when they are ensembled.

Below are some test code to ensemble `blasso` and `spikeslab`.

As we see in the plot and summary, two methods had a strong relationship (0.893).

```
xyplot(resamples(model_list))
```



```
modelCor(resamples(model_list))
```

```
##              blasso spikeslab
## blasso    1.0000000 0.9539384
## spikeslab 0.9539384 1.0000000
```

The ensembled model prediction accuracy was RMSE = 100.028, which was worse than either of the single model (96.76 of blasso and 94.978 of spikeslab).

```
greedy_ensemble <- caretEnsemble(
  model_list,
  metric="RMSE",
  trControl=getTrControl())
summary(greedy_ensemble)
```

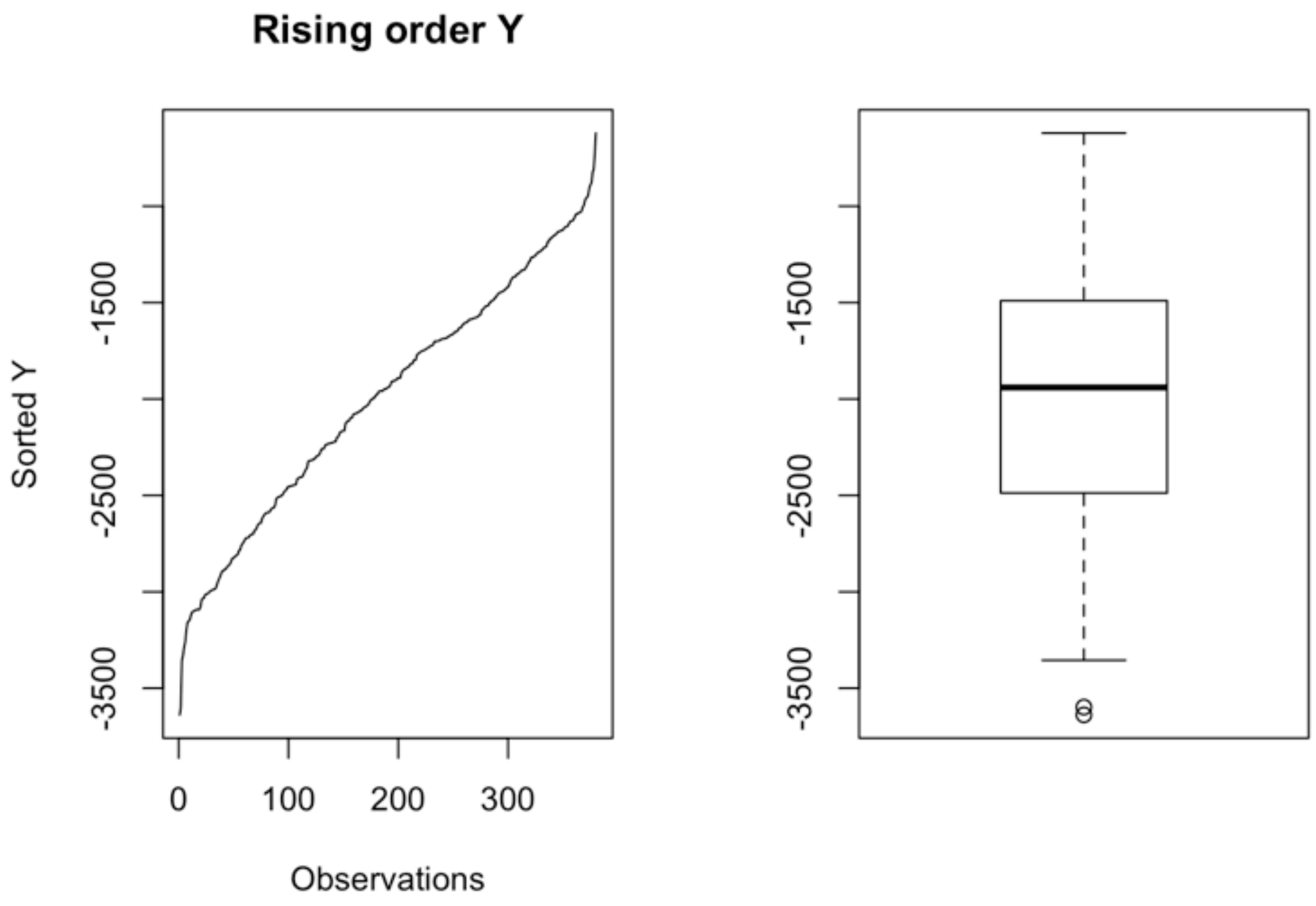
```
## The following models were ensembled: blasso, spikeslab
## They were weighted:
## -0.2998 0.3749 0.625
## The resulting RMSE is: 93.8372
## The fit for each individual model on the RMSE is:
##     method     RMSE   RMSESD
##     blasso 90.07215 8.541894
##  spikeslab 89.88570 7.879607
```

3. Explore the data set and pre-process steps with supplied methods

3.1 Data types

The target data set has 380 observations and 20 variables. ‘Y’ is the output variable with numeric (real) and continues negative values. So it suggested this is a regression problem.

```
my_data <-  read.csv(file = "Ass3Data.csv", row.names = "ID")
par(mfrow=c(1,2))
plot(sort(my_data$Y), type = 'l', main="Rising order Y", xlab="Observations", ylab="Sorted Y")
boxplot(my_data$Y, outline = TRUE, range = 1)
```



There are 19 variables predictors. The variable “BloodType” is a factor with four levels. The rest of the predictors are numeric.

3.2 Data features

The boxplots shows the possible outliers in those variables, which we need to be careful in the following analysis. In addition, when the IQR was increased to 2.4 then all the variables showed no outliers. The same boxplots also suggested the distribution of each predictors and there is no extremely left or right skew of the these variable. The missing data chart showed no obvious pattern in the order of observations. The correlation chart showed some interesting groups in the variables and potential relations between outcome and predictors. * Group1: Strong positive relationship between Reagent (D, F, H, J, L, N). Weak positive relationship between group1 and Y. * Group2: Strong postive relationship between Reagent (A, C, G, I). Moderate negative relationship between group 2 and Y. * Group3: Strong relationship between Reagent E and M. * Group4: Strong relationship between Reagent B and K. Using these groups to examine the missing pattern again and there is no strong pattern in the groups themselves.

3.3 Data roles

The observation identifier,“ID”, was loaded as row.names of dataframes, which is 100% unique. “BloodType” could be a potential observation stratifier. The counts of each bloodtype are showed below.

```
table(my_data$BloodType)
```

```
## 
##   A  AB   B   O 
##  94  49 101 136
```

Uncorrelated (or very weak) predictors are “Alchole”, “Coffee”, “Excercise” and “NumDocVisits”, and the other numeric predictors can also have implicit data roles, such as case weights, we may explore in the following steps.

3.4 Pre-process steps

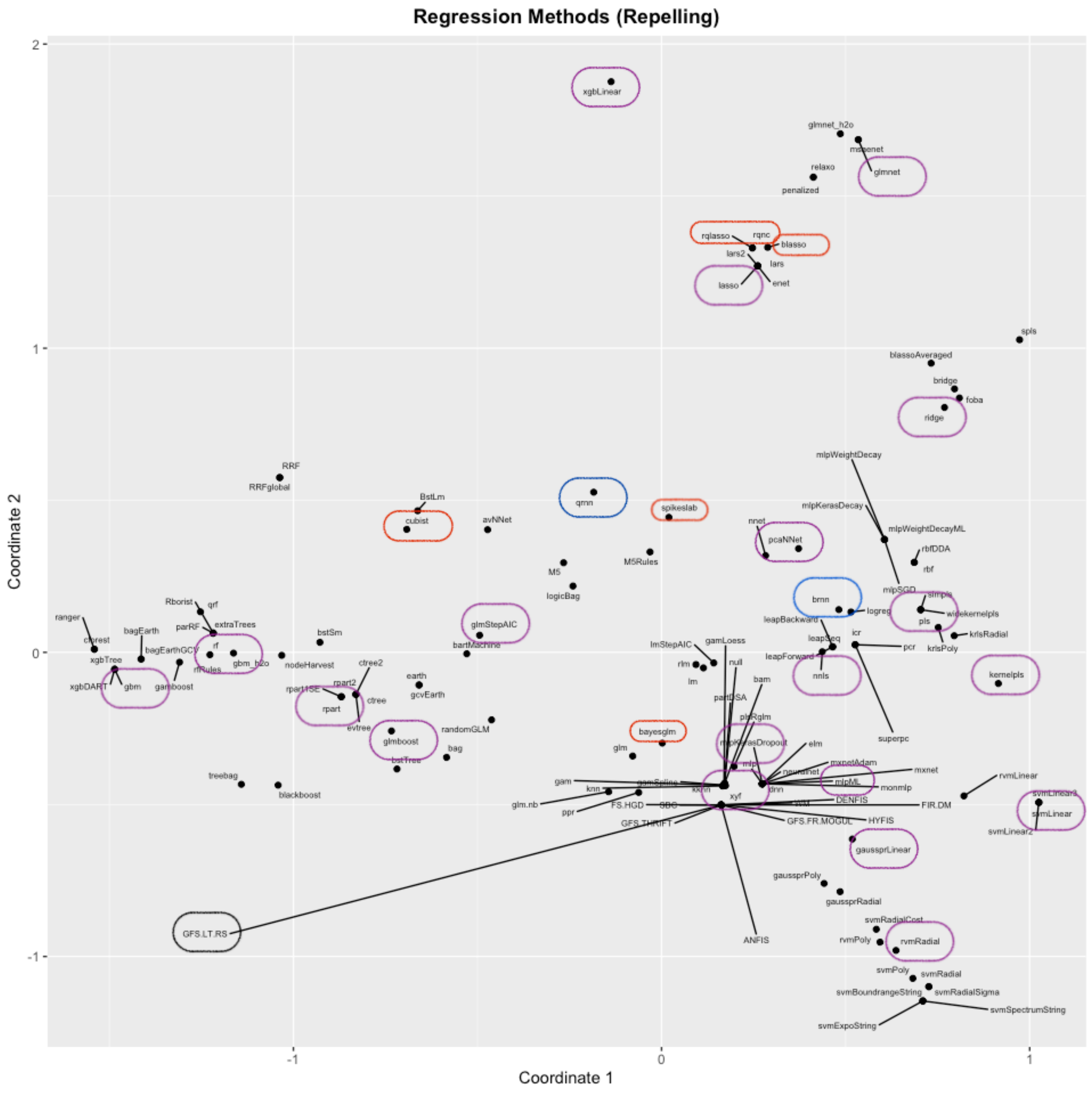
Started from keeping all the default setting in the supplied method. The ’Null Model" showed RMSE 640.30, which was the lowest criterion of methods performance. Using 80/20 train/test data set splitting, then following a general suggestion of pre-processing Ordering of Steps (https://tidymodels.github.io/recipes/articles/Ordering.html). The supplied steps in the example code were added into the suggested `recipe` steps for better understanding. 1. Impute knnimpute, bagimpute, medianimpute (naomit instead of imputing) 2. Individual transformations for skewness and other issues YeoJohnson, poly 3. Discretize Not using in supplied steps 4. Create dummy variables dummy 5. Create interactions Not using in supplied steps 6. Normalization steps center, scale 7. Multivariate transformation pca, pls, ica (also kpca, isomap)

Two steps `step_poly` (also `step_ns`) are not used in example, but may be used to replaced a variable with new one for fitting a non-linear model. Filters, such as `step_nzv` and `step_corr` are not showed in above list, but would be tried later. All the `step_` functions are listed out on Reference (https://tidymodels.github.io/recipes/reference/index.html)

Choice	Model	Pre-processing	RMSE	Hyperparameters	Training Time	Comments
Supplied	GLMnet	“bagimpute”,“Yeojohnson”,“dummy”,“center”,“scale”	90.58	alpha = 0.99,	341.07s	edge of the

				lambda = 2.97		grid
Supplied	PLS	“bagimpute”,“YeoJohnson”,“center”,“scale”,“dummy”	90.73	ncomp 13	118.49 sec	About ten times long to use bagimpute instead of knnimpute, slightly improve the model performance
	kernelpls	“knnimpute”,“center”,“scale”	95.94	136.75 sec		
Supplied	Rpart	“center”,“scale”	152.68		Normalise could improve predict accuracy	

4. Finding candidate models



4.1 Supplied methods

GLMnet (grid edge)

After implementing the choosen pre-processing steps, GLMnet got the lower metrics. While two hyper parameters alpha 0.99 and lambda 2.97 were on the edge of the `random` search.

Result:

- RMSE 90.58
- Hyper parameter:
  - alpha = 0.99, lambda = 2.97

Pre-process

Choice	Model	Pre-processing	RMSE	Time	Comments
	GLMnet	“naomit”,“dummy”	250675.90		default setting, worse than Null model
	GLMnet	“knnimpute”,“dummy”	98.39		“medianimpute” had similar performances, “bagimpute” performaned a bit better but more time-consuming
	GLMnet	“knnimpute”,“dummy”,“center”,“scale”	93.73	57s	Standardise and Normalise numeric predictors expected to improve the performance
	GLMnet	“knnimpute”,“Yeojohnson”,“dummy”,“center”,“scale”,“pca”	365.70		GLMnet Implicits feature selection, so pca did not improve the model performance
	GLMnet	“knnimpute”,“Yeojohnson”,“dummy”,“center”,“scale”,“pls”	95.02		similar situation with “pca”
	GLMnet	“knnimpute”,“Yeojohnson”,“dummy”,“center”,“scale”,“ica”	503.54		similar situation with “pca”
*	GLMnet	“bagimpute”,“Yeojohnson”,“dummy”,“center”,“scale”	90.58	341.07s	Candidate GLMnet Model

PLS

Compare with PCR, PLS took outcome variable into account, which do improve the model performance in terms of RMSE.

Result:

- RMSE 90.73
- Hyper parameter:
  - ncomp 13

Pre-process

Choice	Model	Pre-processing	RMSE	Time	Comments
	PLS	“knnimpute”,“dummy”	142.36	8.26s	
	PLS	“knnimpute”,“YeoJohnson”,“center”,“scale”,"“dummy”"	96.42	8.26 sec	PLS also prefers numeric values to be normal distributed
*	PLS	“bagimpute”,“YeoJohnson”,“center”,“scale”,“dummy”	90.73	118.49 sec	About tem times

					long to use bagimpute instead of knnimpute, slightly improve the model performance

Rpart (grid edge, case weight?)

Result: * RMSE 90.73 * Hyper parameter: * mtry 13

Pre-process

Choice	Model	Pre-processing	RMSE	Time	Comments
	Rpart		160.49		initial setting
	Rpart	“center”,“scale”	152.68		Normalise could improve predict accuracy

4.2 Extended methods

lasso

With experience in assiagnment 2, the pre-processing steps should be similar like GLMnet. The result proved this judgement.

Result:

- RMSE 90.36
- Hyper parameter:
    - fraction 0.79

Pre-process

Choice	Model	Pre-processing	RMSE	Time	Comments
*	lasso	“bagimpute”,“Yeojohnson”,“center”,“scale”,“dummy”	90.36	63.66 sec	
	lasso	“bagimpute”,“center”,“scale”,“dummy”	90.92	72.63 sec	

ridge

Result:

- RMSE 89.45
- Hyper parameter:
    - lambda = 0.01056009

Pre-process

Choice	Model	Pre-processing	RMSE	Time	Comments
*	ridge	“bagimpute”,“Yeojohnson”,“center”,“scale”,“dummy”	89.45	150.89 sec	

kernelpls

Result:

- RMSE 90.73
- Hyper parameter:
    - ncomp 12

Pre-process

Choice	Model	Pre-processing	RMSE	Time	Comments
	kernelpls	“knnimpute”	184.03	178.69 sec	
*	kernelpls	“knnimpute”,“center”,“scale”	95.94	136.75 sec	
	kernelpls	“knnimpute”,“center”,“scale”,“dummy”	95.87	136.75 sec	dummy had no significatn effect

rf (Random Forest)

There is only one tuning hyper parameter `mtry` of `rf` in `caret` , the other `ntree` is set default 500. The method does not tolerant missing values, so “knnimpute” was set as default. Using out of bag instead of bootstrap to be the validation method can improve the model preformance. (Result was recorded below, but trainControl() was not modified in the final shiny app for a consistency of model selection) Tree-base methods may not predict accurate in regression problem, but they do offer insight of the importance of variables.

Result:

- RMSE 124.6
- Hyper parameter:
    - mtry 14

Pre-process

Choice	Model	Pre-processing	RMSE	Time	Comments
*	rf	"knnimpute"	124.80	55.22 sec	
	rf	"knnimpute","center","scale"	124.60	102.22 sec	Normalise did not improve random forest
	rf	"knnimpute","center","scale"	115.32	102.22 sec	"oob" resample method improve rf method

xgbLinear ()

The method are not tolerant to missing value and norminal variables, so "knnimpute" and "dummy" were the initial pre-process steps.

Result:

- RMSE 114.77
- Hyper parameter:
    - nrounds = 60, lambda = 0.251548, alpha = 0.0002734832, eta = 1.294612.

Pre-process

Choice	Model	Pre-processing	RMSE	Time	Comments
	xgbLinear	"knnimpute"	114.77	119.05 sec	

gbm

The method can handle missing value, which is not metioned in the tags. Whether normalise nor dummy variables did not improve the method performance.

Result:

- RMSE 550.26
- Hyper parameter:
    - shrinkage = 0.33, interaction.depth = 1, n.minobsinnode = 20, n.trees = 321

Pre-process

Choice	Model	Pre-processing	RMSE	Time	Comments
	gbm		556.26	163.29 sec	
	gbm	"knnimpute","center","scale"	551.30	144.25 sec	
	gbm	"knnimpute","center","scale","dummy"	550.26	114.77 sec	

glmboost mboost

Result:

- RMSE 94.44
- Hyper parameter:
    - mstop = 532 and prune = yes

Pre-process

Choice	Model	Pre-processing	RMSE	Time	Comments
	glmboost		556.26	163.29 sec	
*	glmboost	"knnimpute","dummy"	94.44	78.09 sec	
	glmboost	"knnimpute","center","scale","pls","dummy"	95.53	106.22 sec	

svmLinear kernlab (grid edge)

Result:

- RMSE 98.49
- Hyper parameter:
  - C = 0.07086818

Pre-process

Choice	Model	Pre-processing	RMSE	Time	Comments
	svmLinear	“knnimpute”,“dummy”	99.00	116.53 sec	Not tolerant missing values
*	svmLinear	“knnimpute”,“center”,“scale”,“dummy”	98.49	254.67 sec	
	svmLinear	“knnimpute”,“center”,“scale”,“pls”,“dummy”	101.79	246.42 sec	

glmStepAIC (MASS)

Result:

- RMSE 103.44
- No hyper parameter:

Pre-process

Choice	Model	Pre-processing	RMSE	Time	Comments
	glmStepAIC	“knnimpute”,“dummy”	103.44	52.23 sec	Not tolerant missing values
*	glmStepAIC	“knnimpute”,“center”,“scale”,“dummy”	103.34	11.57 sec	

rvmRadial (kernlab)

Result:

- RMSE 169.75
- Hyper parameter:
  - sigma = 0.005040349

Pre-process

Choice	Model	Pre-processing	RMSE	Time	Comments
	rvmRadial	“knnimpute”,“dummy”	2091.18	52.23 sec	Not tolerant missing values
*	rvmRadial	“knnimpute”,“center”,“scale”,“dummy”	169.75	122.23 sec	Normalisation can siginificantly improve the method fitting

gaussprLinear (kernlab)

Result:

- RMSE 86.88
- No hyper parameter:

Pre-process

Choice	Model	Pre-processing	RMSE	Time	Comments
	gaussprLinear	“knnimpute”,“dummy”	96.48	9.88 sec	Not tolerant missing values
	gaussprLinear	“knnimpute”,“center”,“scale”,“dummy”	92.68	9.91 sec	
	gaussprLinear	“bagimpute”,“center”,“scale”,“dummy”	86.88	35.23 sec	
	gaussprLinear	“bagimpute”,“center”,“scale”,“pca”,“dummy”	85.35	35.23 sec	

qrnn (qrnn)

The quantile regression **Result:**

- RMSE 12.39
- Hyper parameter:

    - n.hidden = 3, penalty = 0 and bag = FALSE

Pre-process

Choice	Model	Pre-processing	RMSE	Time	Comments
	qrnn	“knnimpute”,“dummy”	30.08	1125.58 sec	Not tolerant missing values
*	qrnn	“bagimpute”,“center”,“scale”,“dummy”	12.39	1412.66 sec	

brnn (brnn)

Result:

- RMSE 21.31
- Hyper parameter:
    - neurons = 4

Pre-process

Choice	Model	Pre-processing	RMSE	Time	Comments
	brnn	“knnimpute”,“dummy”	40.48	94.52 sec	Not tolerant missing values
	brnn	“knnimpute”,“YeoJohnson”,“dummy”	33.82	34.66 sec	
	brnn	“knnimpute”,“center”,“scale”,“dummy”	33.47	34.66 sec	
	brnn	“knnimpute”,“center”,“scale”,“pca”,“dummy”	35.08	110.53 sec	Method has done regularization, so feature extraction did no improve here
*	brnn	“bagimpute”,“center”,“scale”,“dummy”	21.31	349.26 sec	

kknn (kknn)

Result:

- RMSE 150.02
- Hyper parameter:
    - kmax = 51, distance = 1.006154 and kernel = inv

Pre-process

Choice	Model	Pre-processing	RMSE	Time	Comments
	kknn	“knnimpute”,“dummy”	200.2	30.25 sec	Not tolerant missing values
*	kknn	“knnimpute”,“pca”,“dummy”	150.02	34.66 sec	

xyf (kohonen)

Result:

- RMSE 702.31
- Hyper parameter:
    - xdim = 3, ydim = 14, user.weights = 0.8750179 and topo = rectangular

Pre-process

Choice	Model	Pre-processing	RMSE	Time	Comments
	xyf	“knnimpute”,“dummy”	702.31	30.25 sec	Not tolerant missing values

mlpKerasDropout (keras)

The mothod can work in test scripts, but not work in shiny. So only result recorded below.

Result:

- RMSE 680.52
- Hyper parameter:
    -

Pre-process

Choice	Model	Pre-processing	RMSE	Time	Comments
	mlpKerasDropout	“knnimpute”,“dummy”	680.52	75.25 sec	Not tolerant missing values

mlpML (RSNNS)

Result:

- RMSE 702.31
- Hyper parameter:
    - layer1 = 2, layer2 = 5 and layer3 = 0

Pre-process

Choice	Model	Pre-processing	RMSE	Time	Comments
	mlpML	“knnimpute”,“dummy”	702.31	49.02 sec	Not tolerant missing values

pcaNNet (nnet)

Result:

- RMSE 702.31
- Hyper parameter:
    - size = 4 and decay = 3.743782e-05

Pre-process

Choice	Model	Pre-processing	RMSE	Time	Comments
	pcaNNet	“knnimpute”,“center”,“scale”,“dummy”	2098.83	24.46 se	Standardise deteriorated the model performance
	pcaNNet	“knnimpute”,“dummy”	702.31	49.02 sec	Not tolerant missing values

GFS.LT.RS (frbs)

Failed to fit the model, due to more than three hours run in Rstudio.

cubist (Cubist)

Result:

- RMSE 79.49
- Hyper parameter:
    - committees = 81 and neighbors = 5

Pre-process

Choice	Model	Pre-processing	RMSE	Time	Comments
	cubist	“knnimpute”,“dummy”	79.49	187.6 sec	
	cubist	“knnimpute”,“center”,“scale”,“dummy”	86.7	176 sec	Normalise did not improve, but enlarge the RMSE
*	cubist	“bagimpute”,“dummy”	68.57	340.89 sec	

spikeslab (spikeslab)

Result:

- RMSE 87.86
- Hyper parameter:
    - vars = 16

Pre-process

Choice	Model	Pre-processing	RMSE	Time	Comments

	spikeslab	“knnimpute”,“center”,“scale”,“dummy”	87.86	59.31 sec	

bayesglm (arm)

Result:

- RMSE 91.37
- No hyper parameter

Pre-process

Choice	Model	Pre-processing	RMSE	Time	Comments
	bayesglm	“knnimpute”,“center”,“scale”,“dummy”	91.37		42.64 sec

rqlasso (rqPen)

Result:

- RMSE 93.17
- Hyper parameter:
  - lambda = 0.01

Pre-process

Choice	Model	Pre-processing	RMSE	Time	Comments
	rqlasso	“knnimpute”,“center”,“scale”,“dummy”	93.17	307.61 sec	

blasso (monomvn)

Result:

- RMSE 87.54
- Hyper parameter:
  - sparsity = 0.29

Pre-process

Choice	Model	Pre-processing	RMSE	Time	Comments
	blasso	“knnimpute”,“center”,“scale”,“dummy”	87.54	44.89 se	

Additional readings:

The Remote Sensing Model for Estimating Urban Impervious Surface Percentage Based on the Cubist Model Tree (http://www.dqxxkx.cn/article/2016/1560-8999/1560-8999-18-10-1399.shtml)

A Brief Introduction to caretEnsemble (https://cran.r-project.org/web/packages/caretEnsemble/vignettes/caretEnsemble-intro.html)

How to Build an Ensemble Of Machine Learning Algorithms in R (https://machinelearningmastery.com/machine-learning-ensembles-with-r/)

First steps with Non-Linear Regression in R (https://datascienceplus.com/first-steps-with-non-linear-regression-in-r/)

Second steps with Non-Linear Regression in R (https://datascienceplus.com/second-step-with-non-linear-regression-adding-predictors/)

(https://blog.csdn.net/weixin_34138377/article/details/94026160?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromBaidu-3&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromBaidu-3)