Boston University

Stock Prediction Using Machine Learning

December 16, 2019

Author	*Student ID*
Niyu Jia	U97595078

Contents

1. Abstract

Our project aims at forecasting the monthly stock returns of a company and predicting whether a stock will grow or fall through the characteristics of the company and the market.

To begin with, we need to deal with our data: sweep out the outliers, standardize the variables and select 28 variables we need via Variance Inflation Factor (VIF) algorithm and Random Forest Variable selection method.

Then we use elastic net regression, boosting regression and auto-ML for our prediction of the monthly stock return. As for predicting the moving trend of the stock price, We use Bayesian Additive Regression Tree (BART) which is a relatively new method for our classification.

2. Data Cleaning

2.1 General Processing

First, we did some general data cleaning process including dealing with outliers, standardization and lagging problems.

When dealing with outliers, we found 5% and 95% quantile for each variable and replaced those extreme data outside the range by these two quantiles.

As for the standardization, we subtracted the mean of the predictor's data from the predictor values and divided by the standard deviation. Also, we identified numeric predictor columns with a single value (i.e. having zero variance) and excludes them from further calculations.

Since our goal is to use start-of-the-month data to construct a forecast of the return value and direction at the end of the month, we cannot use the same month's end data to predict. Therefore, we shifted those data ended with "_end" by one period backward and filled those missing values with 0.

2.2 VIF Variable Selection

VIF is the quotient of the variance in a model with multiple terms by the variance of a model with one term alone. It quantifies the severity of multicollinearity in an ordinary least square regression analysis. In this part, we use the Variance Inflation Factor (VIF) criteria to select variables to avoid multicollinearity problems. To start with, we did a linear regression between "RETMONTH_end" and all other variables and dropped all those variables that were linearly dependent. Then, we used the remaining variables to fit VIF variable selection method to avoid multicollinearity further.

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 X_2 + \beta_3 X_3 + \dots \beta_n X_n + \epsilon$$
$$\text{Var}\left(\hat{\beta}_j\right) = \frac{s^2}{(n-1)Var(X_j)}$$

Intutively, VIF measures how much variance of parameter β_j is explained by the

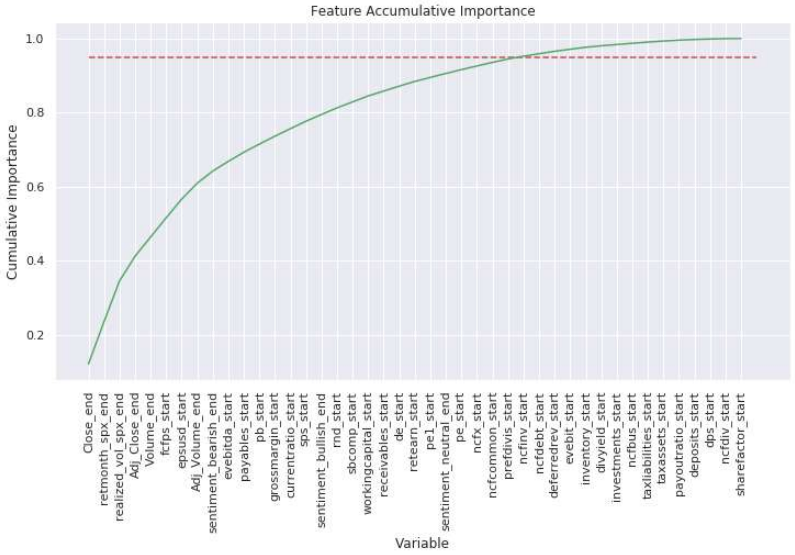
rest of the data. In our process, we calculated variance-inflation factors for linear model between "RETMONTH_end" and the remaining variables and dropped all those variables whose VIF is larger than 20. As a result, we selected 44 variables after this process.

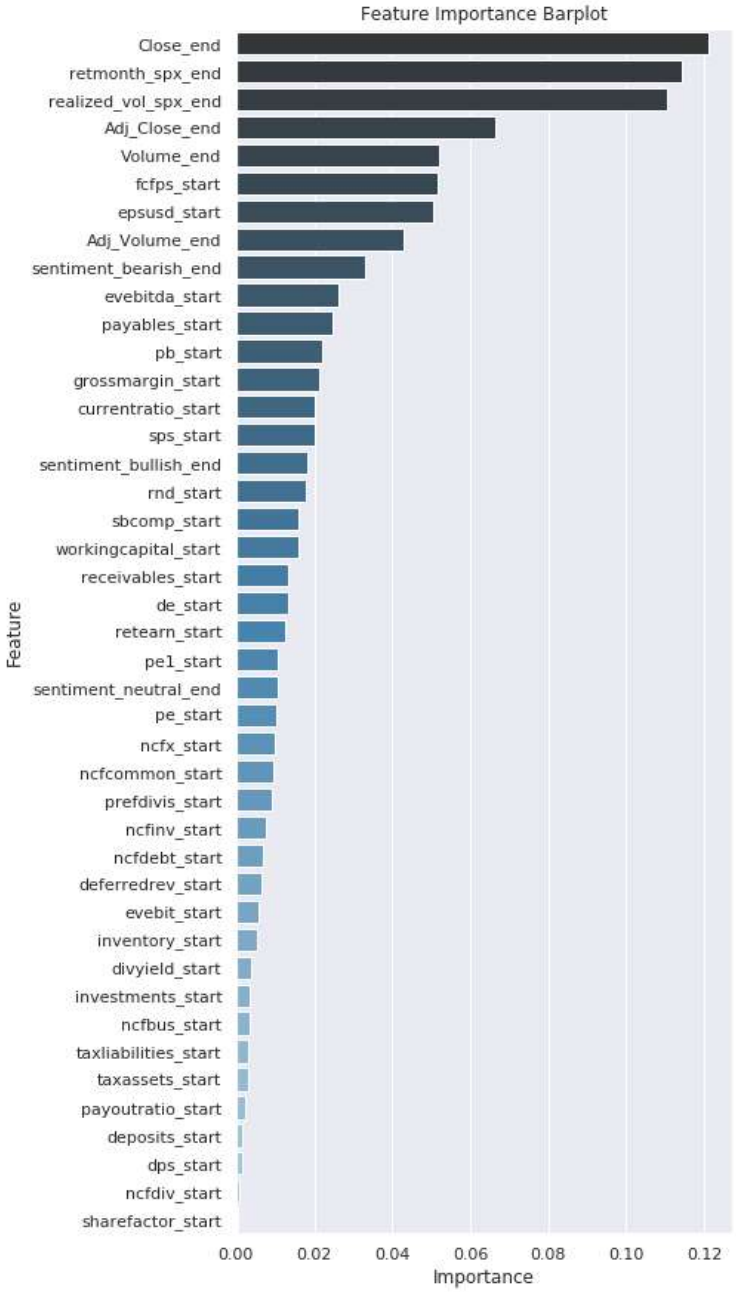
2.3 Random Forest Variable Selection

Next, we use the remaining variables to fit feature selection to go a step further which can effectively reduce overfitting and computational cost. Random forests are good for feature selection because it is easy to compute how much each variable is contributing to the decision. When training a tree, it is possible to compute how much each feature decreases the impurity. The more a feature decreases the impurity, the more important the feature is. In random forests, the impurity decrease from each feature can be averaged across trees to determine the final importance of the variable. To state is more concrete, features that are selected at the top of the trees are in general more important than features that are selected at the end nodes of the trees.

In our process, we first split data set into training set and testing set. Then we did model fitting using training set, computed every feature's importance percentage and ranked them in a decreasing order. Then we selected the first 28 variables which had cumulative importance percentage greater than 95%. The refinement variables are: Close_end, retmonth_spx_end, realized_vol_spx_end, Adj_Close_end, Volume_end, fcfps_start, epsusd_start, Adj_Volume_end, sentiment_bearish_end, evebitda_start, payables_start, pb_start, grossmargin_start, sps_start, currentratio_start, rnd_start, sentiment_bullish_end, workingcapital_start, sbcomp_start, de_start, receivables_start, retearn_start, pe_start, sentiment_neutral_end, pe1_start, ncfx_start, ncfcommon_start, prefdivis_start.

Here, the difference between regression and classification is that in classification, we added a new variable "indicator" which indicates whether "RETMONTH_end" is greater than 0. In other words, if "RETMONTH_end" is greater than 0, indicator is 1; otherwise, indicator is 0.


Feature Accumulative Importance
Cumulative Importance
1.0
0.8
0.6
0.4
0.2
Close_end
retmonth_spx_end
realized_vol_spx_end
Adj_Close_end
Volume_end
fcfps_start
epsusd_start
Adj_Volume_end
sentiment_bearish_end
evebitda_start
payables_start
pb_start
grossmargin_start
currentratio_start
sps_start
sentiment_bullish_end
rnd_start
sbcomp_start
workingcapital_start
receivables_start
de_start
retearn_start
pe1_start
sentiment_neutral_end
pe_start
ncfx_start
ncfcommon_start
prefdivis_start
ncfinv_start
ncfdebt_start
deferredrev_start
evebit_start
inventory_start
divyield_start
investments_start
ncfbus_start
taxliabilities_start
taxassets_start
payoutratio_start
deposits_start
dps_start
ncfdiv_start
sharefactor_start
Variable


Feature Importance Barplot
Close_end
retmonth_spx_end
realized_vol_spx_end
Adj_Close_end
Volume_end
fcfps_start
epsusd_start
Adj_Volume_end
sentiment_bearish_end
evebitda_start
payables_start
pb_start
grossmargin_start
currentratio_start
sps_start
sentiment_bullish_end
rnd_start
sbcomp_start
workingcapital_start
receivables_start
de_start
retearn_start
pe1_start
sentiment_neutral_end
pe_start
ncfx_start
ncfcommon_start
prefdivis_start
ncfinv_start
ncfdebt_start
deferredrev_start
evebit_start
inventory_start
divyield_start
investments_start
ncfbus_start
taxliabilities_start
taxassets_start
payoutratio_start
deposits_start
dps_start
ncfdiv_start
sharefactor_start
Feature
0.00
0.02
0.04
0.06
0.08
0.10
0.12
Importance

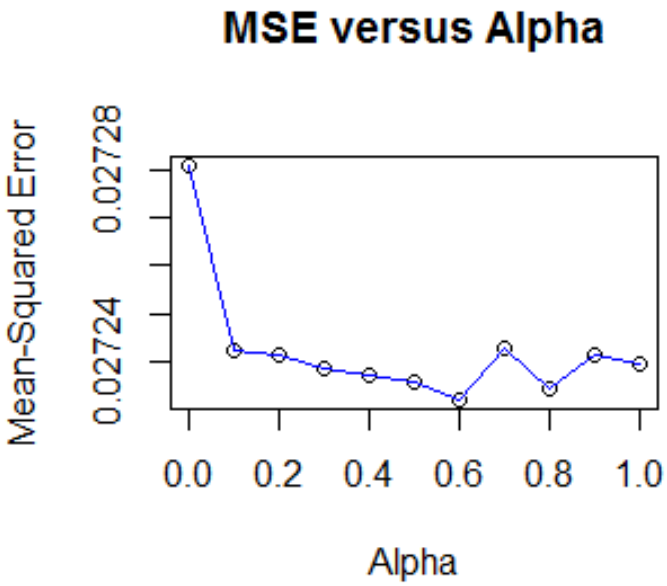
3. Regression Modeling

3.1 Regression Using Elastic Net

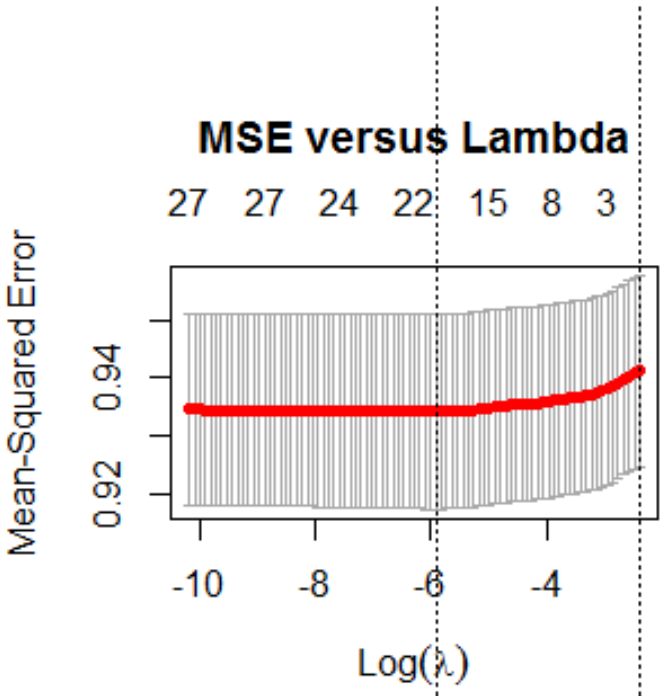
The elastic net is a regularized regression method that linearly combines the lasso and ridge regression. We need to find the coefficient $\hat{\alpha}$ by solving:

$$\min \sum_{i=1}^{n} \left(y_i - \alpha^\top x_i\right)^2 + \lambda \left(\nu \sum_{j=1}^{d} |\alpha_j| + (1-\nu) \sum_{j=1}^{d} \alpha_j^2\right)$$

To figure out the best α, we plot MSE against different α:



From the figure above, we set α as 0.6 when MSE achieves its minimum.



We also plot MSE against λ, to minimize MSE, we set λ to be 0.002703. The main parameters of our model are given by:

alpha	MSE	R Squared	Lambda
0.6	0.027232	−0.03014	0.002703

The main coefficients of the main predictors of our model are shown below:

variable	coefficient
intercept	0.023218
realized_vol_spx_end	−0.02852
sentiment_neutral_end	−0.03478
sentiment_bearish_end	0.016273
Close_end	0.009633
sbcomp_start	0.001467
epsusd_start	−0.00273
pe1_start	0.011321
currentratio_start	−0.0019
sps_start	−0.00144
prefdivis_start	−0.0149
payables_start	0.001285
pe_start	−0.00104
fcfps_start	0.002621
Adj_Close_end	−0.01153
evebitda_start	−0.00027
Adj_Volume_end	0.86682
Volume_end	−0.8284
retmonth_spx_end	0.04357
receivables_start	0.004339
workingcapital_start	0.003301

3.2 Gradient Boosting

Different from random forest, gradient boosting Machines produce a prediction model in the form of an ensemble of weak prediction models. Each weak model will adapt itself based on previous model to improve the performance of the whole model. Gradient boosting Machines usually can deal with null data, give us higher accuracy in our prediction and are applicable for various of loss functions, which makes it suitable for our project.

However, since gradient boosting machines keep adapting to minimize the error, it may bring us over-fitting problems. Besides, gradient boosting machines are computationally expensive, As for these drawbacks, we will deal with them cautiously in our tuning process.

There are 2 parameters on which we concentrate in gradient boosting machines: the number of the trees and the learning rate. Generally speaking, gradient boosting machines can contain lots of tree, yet too many trees may lead to over-fitting, which means we should figure out the proper number of trees to minimize our loss function. The learning rate determines how fast we can calculate the gradient descent in our model, but in practice, we have to use lower learning rate and consume more time to avoid over fitting problems.

When we build our boosted tree model, we first fit a small tree, then fit additional small trees in each iteration for the unexplained residuals. After B times iteration, our model comes down to:

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$$

We use the package "h2o" in R to do the modelling. It is a powerful machine learning platform that offers implementations of many supervised and unsupervised machine learning algorithms as well as gradient boosting machine. Please note that in order to use this package in R, users must install Java on the computer. The Java SE downloading website is as follows:
($https://www.oracle.com/technetwork/java/javase/downloads/index.html$).

3.2.1 Regression Using GBM

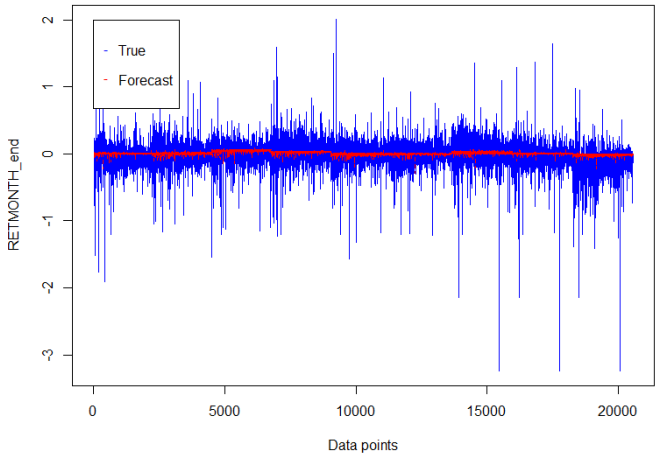
In our process, we first divided data into train set and test set. To state it more concrete, all data before 2018.01.01 were included in train set and the others are viewed as test set.

In order to fit GBM model into our train set, we need to determine some hyperparameters first. Here, the two hyperparameters we would like for tuning are "learning rate" and "number of trees", whose range we set were [0.01, 0.1] with step size 0.01 and [10, 300] with step size 10. Then, we used grid search method and "RandomDiscrete" strategy to select the best combination of two parameters with minimum MSE out of 50 models. The tuning result is to use 210 trees with learning rate 0.02.

```
H2O Grid Details
================

Grid ID: gbm_grid2
Used hyper parameters:
  -  learn_rate
  -  ntrees
Number of models: 50
Number of failed models: 0

Hyper-Parameter Search Summary: ordered by increasing MSE
  learn_rate ntrees          model_ids                mse
1       0.02    210 gbm_grid2_model_42 1.1985023929971437
2       0.02    230 gbm_grid2_model_18 1.1985217887518336
3       0.02    240  gbm_grid2_model_4 1.1988119944360316
4       0.02    200 gbm_grid2_model_11 1.1990907148138519
5       0.02    160 gbm_grid2_model_24 1.2000446994600007

---
   learn_rate ntrees          model_ids                mse
45       0.09    220 gbm_grid2_model_13  1.211313554508512
46        0.1    170 gbm_grid2_model_27 1.2113495570633157
47       0.08    290 gbm_grid2_model_32 1.2120533765052668
48       0.07    210 gbm_grid2_model_34 1.2125650466166211
49       0.09    240  gbm_grid2_model_8 1.2143547482936015
50        0.1    260 gbm_grid2_model_20 1.2146775646497976
```

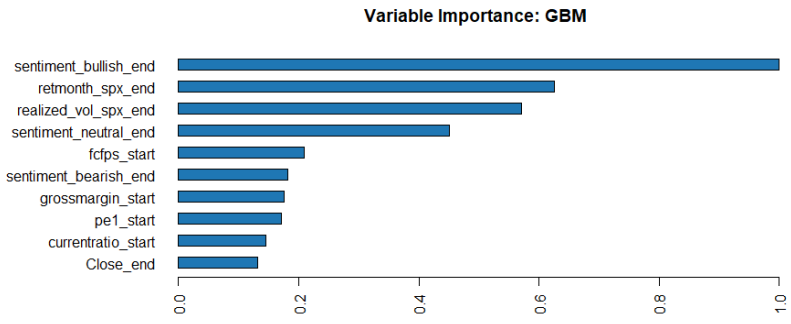
Afterwards, we constructed the GBM model with these two parameters and 28 features, which we got after VIF and RF variable selection methods, using "h2o.gbm" function in R package ("h2o") and fit this model into the train set. By evaluating this model in the test set, we got the Out-of-sample R-Squared at around 1.2%.

```
> summary(model_gbm)
Model Details:
==============

H2ORegressionModel: gbm
Model Key:  gbm
Model Summary:
  number_of_trees number_of_internal_trees model_size_in_bytes min_depth max_depth mean_depth min_leaves max_leaves
1             210                     210               86321         5         5    5.00000         15         32
  mean_leaves
1    27.99048

H2ORegressionMetrics: gbm
** Reported on training data. **

MSE:  0.8273288
RMSE:  0.9095762
MAE:  0.5909882
RMSLE:  NaN
Mean Residual Deviance :  0.8273288
R^2 :  0.1209996


H2ORegressionMetrics: gbm
** Reported on validation data. **

MSE:  1.198502
RMSE:  1.094761
MAE:  0.6795595
RMSLE:  NaN
Mean Residual Deviance :  1.198502
R^2 :  0.01205486
```



The chart above is "True Monthly Return vs Forecast Monthly Return". From the chart we can see that our model can predict the main trend but it cannot show more details like sharp fluctuations and this maybe because our model cannot describe jumps, which is really common in the real financial world.

```
Variable Importances: (Extract with `h2o.varimp`)
=================================================

Variable Importances:
                variable relative_importance scaled_importance percentage
1 sentiment_bullish_end        51692.542969          1.000000   0.233330
2      retmonth_spx_end        32369.277344          0.626189   0.146108
3  realized_vol_spx_end        29484.072266          0.570374   0.133085
4 sentiment_neutral_end        23289.775391          0.450544   0.105125
5           fcfps_start        10816.777344          0.209252   0.048825

---
                variable relative_importance scaled_importance percentage
23    receivables_start          858.390808          0.016606   0.003875
24         epsusd_start          713.104736          0.013795   0.003219
25 workingcapital_start          595.203308          0.011514   0.002687
26           ncfx_start          543.290161          0.010510   0.002452
27      prefdivis_start          181.743546          0.003516   0.000820
28       payables_start          141.996719          0.002747   0.000641
```



This chart shows the top 10 variables' scaled importance or contribution to our GBM model. Top 3 variables are "sentiment_bullish_end", "retmonth_spx_end" and "realized_vol_spd_end" and they contribute about 50% importance to the whole model.

4. Classification with Bayesian Additive Regression Tree

We use a relatively new method for classification,which is Bayesian Additive Regression Tree (BART). We choose this model because we suffer from the time in tuning parameters especially for tree-based method. With bayesian-based method, we are able to be free from choosing parameters. Only we need to do is to check convergence of posterior which saves a lot of training time.Of course, at a cost, this method is memory-intensive.

4.1 Model Construction

4.1.1 BART Model

Let's start from general regression:

$$Y = f(x) + \epsilon \qquad \epsilon \sim N(0, \sigma^2) \tag{4.1}$$

To do this, we considering modeling an approximation $f(x) = E(Y|x)$ and $f(x) \approx h(x) = \sum_{j=1}^{m} g_j(x)$. Where g_j denotes a single tree, m is the total number of trees.

To fit the sum-of-trees model, BART uses a tailored version of Bayesian backfitting MCMC [Hastie and Tibshirani(2000)] that iteratively constructs and fits successive residuals. The special thing is that the BART weaken each by using a prior.

We begin by establishing notation for a single tree model:
T: a binary tree
$M = \{\mu_1, \mu_2, ...\mu_b\}$: a set of parameters associated with each of b terminal nodes of T
x : the predictors, the decision rules are binary split:

For single components of $x = (x_1, x_2...x_p)$, we split of the form $\{x \in A\}$ and $\{x \notin A\}$; For continues x_i, we split of the form $\{x > c\}$ and $\{x \leq c\}$

Therefore the sum of tree can be intepreted as:

$$Y = g(x; T, M) + \epsilon \qquad \epsilon \sim N(0, \sigma^2) \tag{4.2}$$

$$Y = \sum_{j=1}^{m} g(x; T_j, M_j) + \epsilon \qquad \epsilon \sim N(0, \sigma^2) \tag{4.3}$$

4.1.2 Prior Settings

We set three critical priors in this model: priors on T_j, M_j and σ

$$\begin{aligned} p((T_1, M_1), P(T_2, M_2), ..., (T_m, M_m), \sigma) &= [\prod_j p(T_j, M_j)]p(\sigma) \\ &= [\prod p(M_j|T_j)p(T_j)]p(\sigma) \end{aligned} \tag{4.4}$$

$$\text{and } p(M_j|T_j) = \textstyle\prod_i p(\mu_{ij}|T_j)$$

where $\mu_{ij} \in M_j$

•T_j prior:
$p(T_j)$ contains three important parts:

(i) The probability that a node at depth $d(= 1, 2, 3...)$ is non-terminal, given by

$$\alpha(1 + d)^{-\beta}, \alpha \in (0, 1), \beta \in [0, \infty]$$

(ii) The distribution on the splitting variable assignments at each interior node
(iii) The distribution on the splitting rule assignment in each interior node,conditional on the splitting variable
For (ii) and (iii), we just use uniform prior.

•$\mu_{ij}|T_j$ prior:
We use conjugate normal distribution $N(\mu_\mu, \sigma_\mu^2)$ which offers computational benefits because μ_{ij} can be margined out. Note that $E(Y|X)$ is the sum of $m\mu_{ij}$'s under the sum of tree model and because the μ_{ij} are apriori i.i.d.

Therefore the prior of $E(Y|X)$ should be $N(m\mu_\mu, m\sigma_\mu^2)$. One trick we do is to choose μ_μ, σ_μ so that $N(m\mu_\mu, m\sigma_\mu^2)$ assings substantial probability to the interval (y_{min}, y_{max}). This can be conveniently done by choosing μ_μ, σ_μ so that $m\mu_\mu - k\sqrt{m}\sigma_\mu = y_{min}$ and $m\mu_\mu + k\sqrt{m}\sigma_\mu = y_{max}$ for some pre-selected value of k. In this application, we choose $k = 2$ to yield a 95 percent prior probability.

•σ prior:
We use the inverse chi-square distribution for $p(\sigma)$, that is $\sigma \sim \nu\lambda/{\mathcal{X}_\nu}^2$. For the choice of ν, λ, we do following steps:

(i) Regress original Y against X by OLS, calculate the residual standard deviation $\hat{\sigma}$

(ii) Choose appropriate ν, λ to shape the distribution, so that $P(\sigma < \hat{\sigma}) = q$, where q means the q_{th} quantile. We usually consider the value such as 0.75,0.9,0.95,0.99 to center the distribution below $\hat{\sigma}$

4.1.3 MCMC for Posterior

Giving the observed data y, the Bayesian setup induces a posterior distribution

$$p((T_1, M_1), ..., (T_m, M_m), \sigma|y)$$

The following backfitting MCMC can be used to sample from this posterior. Let $T_{(j)}$ denotes the set of all trees in the sum except T_j and similarily define $M_{(j)}$. The number of tree is m, the Gibbs sampler will entail m successive draw of (T_j, M_j), conditional on $(T_{(j)}, M_{(j)}, \sigma)$:

$$(T_j, M_j)|(T_{(j)}, M_{(j)}, \sigma, y)$$

$j = 1, 2, ..., m$, followed by a draw of σ from the full conditional :

$$\sigma|T_1, ..., T_m, M_1, ..., M_m, y$$

The draw of σ is simply a draw from an inverse gamma distribution. The more challenge thing is how to draw the m draws of (T_j, M_j).First, note that the conditional distribution $p(T_j, M_j|T_{(j)}, M_{(j)}, \sigma, y)$ depends on $(T_{(j)}, M_{(j)})$ only through

$$R_j = y - \sum_{k \neq j} g(x; T_k, M_k)$$

the n-vector of partial residuals based on a fit that excludes the j_th tree. Thus draw of $(T_j, M_j)|(T_{(j)}, M_{(j)}, \sigma, y)$ is equivalent to draw of $(T_j, M_j)|R_j, \sigma$. Since we have conjugate prior for M_j

$$p(T_j|R_j, \sigma) \propto p(T_j) \int p(R_j|M_j, T_j, \sigma) p(M_j|T_j, \sigma) dM_j$$

can be obtained in closed form up to a norming constant. Our draw of above can be acquired in two successive steps as:

$$(1)\ T_j|R_j, \sigma$$
$$(2)\ M_j|T_j, R_j, \sigma$$

The draw of T_j can be obtained by Metropolis-Hastings and the draw of M_j is just a set of independent draws of the terminal node μ_{ij}'s from a normal distribution. To estimate $f(x)$ or predict Y at a particular x, it's a natural choice for average of the smaple after burn-in.

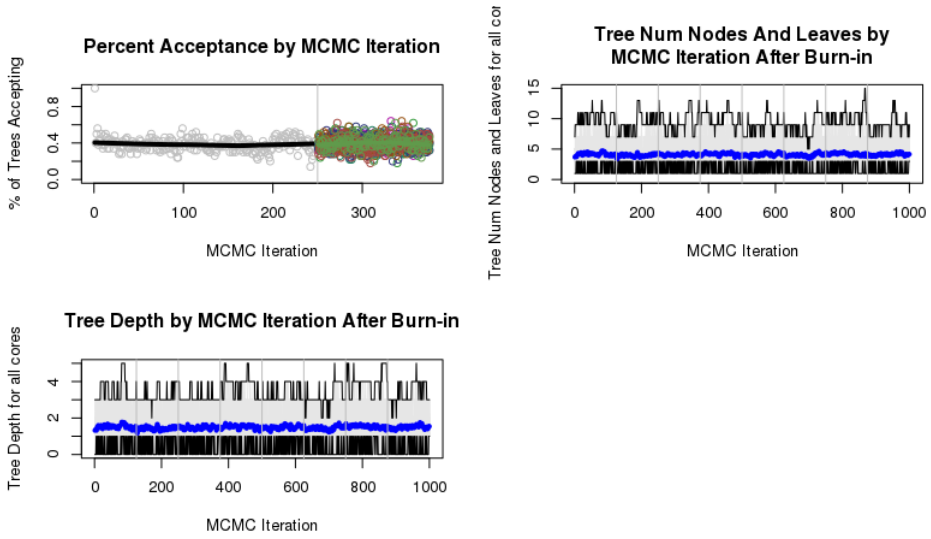
4.2 Results

Using the recommended parameter of $m = 50, \nu = 3, \alpha = 0.95, \beta = 2, k = 2, q = 0.9$. For each BART model, since the single tree would be weakened, we randomly choose 1000 observations for each BART, and build 128 BART model, which means we actually used 128,000 records.

Following shows the model basic statistics:

```
training data n = 1000 and p = 28
built in 2.5 secs on 8 cores, 50 trees, 250 burn-in and 1000 post. samples

confusion matrix:

           predicted NO predicted YES model errors
actual NO       337.000       158.000        0.319
actual YES      129.000       376.000        0.255
use errors        0.277         0.296        0.287
```

The only thing we need to do is to check the convergence to make sure the model is stable. The following diagnosis plot shows the convergence



We see that the algorithm converges. The tree depth converges to approximately 2. Then we use these 128 BART machines to predict out of sample test data. For each test x, we can acquire 128 predictions. Then the final results will be obtained by voting. For choosing out-of-sample data as the final period of each company,size of 2463, we reached an accuracy between 0.65-0.72.

5. Conclusion

In summary, We constructed both regression and classification models with machine learning in this project, to forecast monthly stock returns. We applied the technique we learned in data processing, variable and model selection, parameter optimization, and gave the forecasting results.

Firstly, to get the training and testing dataset, we take the following steps: correct the outliner, use VIF method to solve collinearity problem, use random forest to do variable selection, align the time of data, and do normalization. To avoid using future data, we use the data before 2018-01-01 as train set, and make the data after that point be test set to verify our model.

For the regression model, we choose from elastic net and boosting. By adjusting the tuning parameters, gradient boosting gives the best result, which achieves a 1.2 % R^2 on the out-of-sample data.

For the classification model, the tree based models firstly come to mind. We make attempt from random forest, bagging, boosting, finally decide on BART model, to get rid of the massive work in tuning parameters. As a result, we get 0.71 accuracy.