ST443 - GROUP PROJECT

Group 9*

85770

81024

81090

83310

84316

*All the student contributed equally to the project (20%).

Table of Contents

PART 1: REAL WORLD DATA 4

Section 1. Introduction 4

1.1. Databases – train.csv and test.csv 4

1.2. Kaggle competition objective 4

1.3. Links 4

Section 2. Data Cleaning and Transformations 4

2.1. Missing data and imputation 4

2.2. Transformation of the response variable SalePrice 4

Section 3. Regression 5

3.1. Model Selection 5

3.2. Model Performance Comparison: 5

3.3. Best Regression Model 5

3.4. Other approaches 5

Section 4. Classification 6

4.1. Dependent variable 6

4.2. Data splitting 6

4.3. Model Selection and Results 6

4.4. Best Classification Model 6

Section 5. Conclusion 6

Part 2: Estimation of graphical model using lasso related approaches 7

Section 1: Mathematical Introduction 7

1.1. Graphical Models 7

1.2. Node-wise LASSO approach 7

1.3. Graphical lasso approach 7

Section 2: Results and conclusions 8

2.1. Simulation settings 8

2.2. Miss-classification error 8

2.3. ROC curves: comparison between different models 9

2.4. ROC curves: comparison between different p and n 9

2.5. ROC curves: numerical results 10

2.6 Conclusions 11

Bibliography 12

PART 1: REAL WORLD DATA

Section 1. Introduction

1.1. Databases – train.csv and test.csv

The data is available on Kaggle as part of the competition “House Prices: Advanced Regression Techniques”. The data is in two files train.csv and test.csv. The training dataset consist 1460 observation with 79 explanatory variables with the response “SalePrice”. The testing data contains 1459 observations with corresponding 79 variables and does not contain the dependent variable “SalePrice”. Out of the 79 variables 46 are categorical and 33 are continuous.

1.2. Kaggle competition objective

The objective of the competition is to predict the “SalePrice” in the testing data as accurately as possible. To measure the accuracy of the model one needs submit the predictions to Kaggle’s website and then the submissions are evaluated on Root-Mean-Squared-Error (RMSE) between the logarithm of the predicted value and the logarithm of the observed sales price.

1.3. Links

Kaggle’s competition website: https://www.kaggle.com/c/...

Our GitHub repository: https://github.com/heymic/ST443-Project-group9

Section 2. Data Cleaning and Transformations

2.1. Missing data and imputation

The datasets contain some missing values (NAs). After a closer look at the structure of the data, one finds that some of the missing values could represent either a separate level of the categorical variables or missing values. For instance, NAs in the "*PoolQuality*" variable could either mean that is no actual pool in the house or that the quality of the pool was not recorded.

To solve this step, we cautiously looked at both the data description document attached to the files and other corresponding variable (f.e for *PoolQuality -> PoolArea).* After deciding whether the data point is indeed missing, or it represents a level of the variable, we have decided to group the variables together in to order to assign the most accurate value using the mean, mode or median.

For instance, we have grouped the mean of the *PoolArea* together with *PoolQuality*. Through data inspection we notice that some observations have a positive *PoolArea* while having NAs in the *PoolQuality*, which clearly indicates that these NAs in *PoolQuality* are missing values. In this case we have assign the corresponding value to *PoolQuality* calculated by grouping, by the mean of *PoolArea*.

This approach has been applied to several other variables within the data set, see the attached code for the data cleaning we have done to the original dataset.

2.2. Transformation of the response variable *SalePrice*

Linear models are known to perform better when the response variable is normally distributed. Plotting the distribution of *SalePrice* we noticed that it is skewed to the right. Correcting for the skewness, we applied a logarithmic transformation on the response variable to obtain a “bell-shaped” distribution.

Section 3. Regression

3.1. Model Selection

We have trained many models on different techniques of transformations and imputations of the data and compared them by the 10-fold Cross-Validation errors. We have achieved the best outcome by following the procedures described in Section 2 above.[1]

3.2. Model Performance Comparison:

The two tables below summarise the RMSE of our models on predicting the sales price of the testing data set. As mentioned in Section 1.1, the test set does not have the *SalesPrice* variable, the below RMSE is given to us after submitting our predictions to Kaggle online.

No	Model	RMSE, Kaggle Score
1	Gradient Boosting Model (gbm) tuned	0.12765
2	Lasso with 10fold CV λmin	0.13110
3	Xgboost tuned	0.13380
4	Random Forest with m = p/3 = 26	0.14685
5	Random Forest with m = $\sqrt{p} = 9$	0.14723
6	Bagging Tree	0.15255
7	Linear Regression with all parameters	0.16705
8	Lasso with 10fold CV λ1se	0.16991
9	Regression Tree	0.22079

Combined Models, average of predictions	RMSE, Kaggle Score
No.1 + No.2	0.12438
No.1 + No.2 + No.3	0.12492

3.3. Best Regression Model

The best model obtained (highlighted in green) is the average of the predictions of the gradient of the Gradient Boosting Model (No.1) with the Lasso λmin (No.2) model.

Based on separate models the Gradient Boosting Model and tuned on R package "gbm" performs the best. Interestingly, the Lasso model with lambda chosen as minimum in the Cross-Validation procedure outperforms the random forest methods. These could lead to the hypothesis that house prices are linearly dependent to factors.

3.4. Other approaches

The dataset consists many categorical factors that are ordinal, meaning that one can order them from highest to lowest value. We have tried to train the model by grouping together these ordinal factors based on the Sales Price corresponding mean and treat them as numeric. However, this approach resulted in higher Cross-Validation error rate and higher test error.

[1] Graphs from the regression and classification models are found in the R codes, attached in the appendix at the end of this report.

Section 4. Classification

4.1. Dependent variable

The aim of the classification approaches applied on the dataset is to predict whether or not the house will be sold above the median price.

A binary variable was created based on the log transformed *SalePrice* variable in the dataset. Mean was chosen as the statistics to divide the sale price, as the skewness of *SalePrice* has already been corrected by the logarithmic transformation. Hence, any observation with log(*SalePrice*) greater than the mean would be classified as “High” and “Low” otherwise.

4.2. Data splitting

Testing for error rate would be different from section 3 above, as Kaggle does not accept classification predictions for the testing dataset and the testing data set does not have *SalePrice* for us to manually create our own confusion matrix to identify the error rate.

Hence, the original training data will be randomly divided, with 50% used as testing data and the remaining 50% to be used as training data.

4.3. Model Selection and Results

Using similar methodologies as Section 3, we have trained many models on different techniques and the table below summarises our findings.

No	Model	Error Rate Misclassification
1	Random Forest with m = p/3 = 26	0.0808219
2	Gradient Boosting Model (gbm)	0.0821918
3	Ridge with 10-fold CV λ1se	0.0890411
4	Support Vector Classifier	0.0890411
5	Ridge with 10-fold CV λmin	0.0917808
6	Lasso with 10-fold CV λmin	0.0931507
7	Bagging Tree	0.0986301
8	Lasso with 10-fold CV λ1se	0.1000000
9	Classification Tree (Pruned)	0.1164384
10	Classification Tree	0.1219178

4.4. Best Classification Model

The best classification model here is the random forest model, with the lowest error rate of 8.08%. It is followed closely by the gradient boosting model. The models used for classification generally have lower classification error rate when compared to the regression models in Section 3.

Section 5. Conclusion

Despite not having a basis of comparison for our classification models as Kaggle does not accept submissions for classification results. However, with our best regression model, we have scored in the top 30% of the competition. To improve this result, one could try to group, impute and transform the data in different ways seeking the minimal testing score. Looking at the website scoreboard the best model seems to be averages out of many different models. Improving the model in this way is trial and error, because the some of the models will lead to lower CV error rate and some to larger.

Part 2: Estimation of graphical model using lasso related approaches

Section 1: Mathematical Introduction

1.1. Graphical Models

A graphical model is a probabilistic model which illustrates the conditional dependence structure among p random variables, X =$(\mathrm{X}_1, \dots, \mathrm{X}_\mathrm{P})^\mathrm{T}$. For a network with p nodes, one node is for each variable, along with edges connecting a subset of the nodes. These edges represent the structure of conditional dependence of the p variables.

Specifically, for 1 ≤ j, l ≤ p, c_{jl} = Cov ($\mathrm{X}_\mathrm{j}, \mathrm{X}_\mathrm{l} | \mathrm{X}_\mathrm{k}$, 1≤k≤p, k ≠ j, l) represents the covariance of X_j and X_l conditional on the remaining variables. Then nodes j and l are connected by an edge if and only if $\mathrm{c}_{\mathrm{jl}} \neq 0$. We will mainly focus on two Gaussian graphical models through the lasso regularization in this part, node-wide lasso approach and graphical lasso approach.

Let G = (V, E) denotes an undirected graph with vertex set V = {1, … , p} and edge set

$$\mathrm{E} = \{(\mathrm{j,l}) : \mathrm{c}_{jl} \neq 0,\ 1 \leq \mathrm{j},\ \mathrm{l} \leq \mathrm{p},\ \mathrm{j} \neq \mathrm{l}\}$$

1.2. Node-wise LASSO approach

Node-wise lasso approach is one of the methods used to estimate the conditional dependence structure of Gaussian graphical models, more specifically, for each node j ∈ V, do regression X_j on the remaining variables X_l, l ∈ V, $\mathrm{l} \neq$ j, in the form of

$$\mathrm{X}_\mathrm{j} = \sum_{1 \leq \mathrm{l} \leq \mathrm{p}, \mathrm{l} \neq \mathrm{j}} \beta_{\mathrm{jl}} \mathrm{X}_\mathrm{l} + \varepsilon_{\mathrm{jl}}$$

To obtain a sparse solution for $\beta_{\mathrm{jl}}'s$, the lasso approach can be implemented to select which component in $\{\beta_{\mathrm{jl}}, \mathrm{l} \in \mathrm{V}, \mathrm{l} \neq \mathrm{j}\}$ is non-zero. For a certain tuning parameter λ, let the lasso estimator for β_{jl} to be $\hat{\beta}_{\mathrm{jl}}$: If $\hat{\beta}_{\mathrm{jl}} \neq 0$, then nodes j and l are estimated to be connected. Following rules, which are called node-wise lasso 1 and node-wise lasso 2, respectively, can be used to estimate E in (1):

$$\hat{\mathrm{E}}_1 = \{(\mathrm{j,l}) : \beta_{\mathrm{jl}} \neq 0 \text{ and } \beta_{\mathrm{lj}} \neq 0,\ 1 \leq \mathrm{j},\ \mathrm{l} \leq \mathrm{p},\ \mathrm{j} \neq \mathrm{l}\},$$

$$\hat{\mathrm{E}}_2 = \{(\mathrm{j,l}) : \beta_{\mathrm{jl}} \neq 0 \text{ or } \beta_{\mathrm{lj}} \neq 0,\ 1 \leq \mathrm{j},\ \mathrm{l} \leq \mathrm{p},\ \mathrm{j} \neq \mathrm{l}\}.$$

1.3. Graphical lasso approach

Graphical lasso approach is used for depicting the conditional dependence structure of Gaussian graphical models through the use of L1 (lasso) regularization.

Consider observations $(\mathrm{X}_1, \dots, \mathrm{X}_\mathrm{n})$ from multivariate Gaussian distribution with covariance matrix Σ, one can show that c_{jl} = 0 if and only if θ_{jl} = 0, where θ_{jl} is the (j, l)-th entry of the inverse covariance matrix, θ = Σ^{-1}, also called precision matrix.

The basic assumptions for the model is that the observations are from a multivariate Gaussian distribution with covariance matrix Σ. If the θ_{jl} = 0, then variables i and j are conditionally independent, given all the other variables. Thus, imposing L1 penalty for the estimation of Σ^{-1} will increase its sparsity.

The graphical lasso approach is able to estimate the inverse covariance matrix θ under a multivariate normal model by maximizing the ℓ1-penalized log-likelihood $(\log \det\theta - \mathrm{trace}(S\theta) + \lambda \sum_{j\neq l} \theta_{jl})$.

The edge set can be represented by E={(j,l) : $\theta_{jl} \neq 0$, $1 \leq$ j, l$\leq$ p, j≠l }. With a certain choice of tuning parameter, the estimated edge set can be shown as $\widehat{E}_3$={(j,l) : $\hat{\theta}_{jl} \neq 0$, $1 \leq$ j, l$\leq$ p, j≠l }.

Section 2: Results and conclusions

2.1. Simulation settings

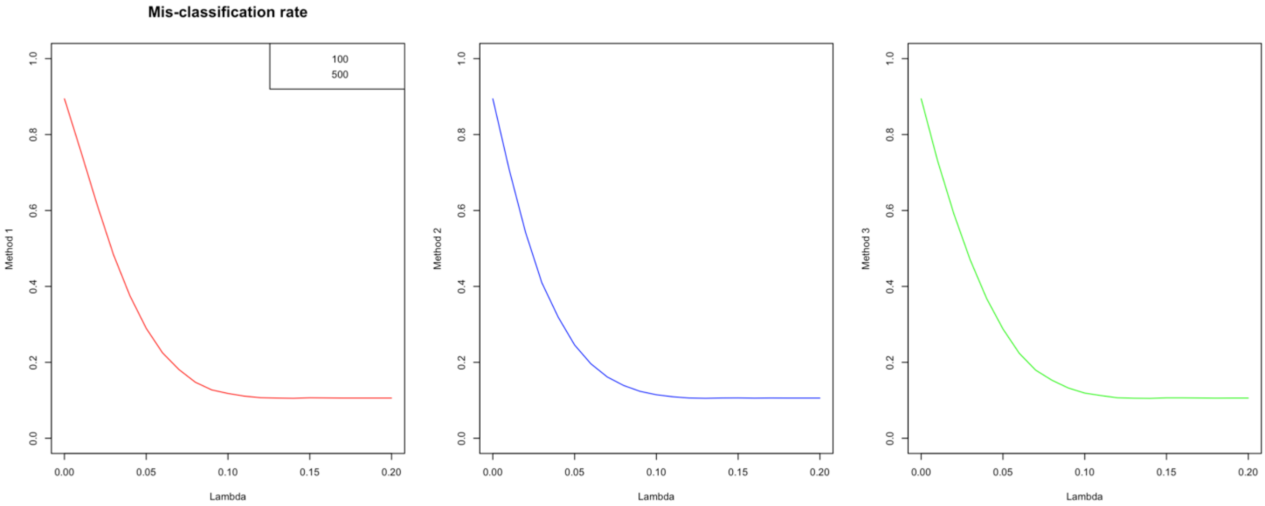
To compare the three different estimation methods, we have estimated the edges with different complexity and we have plotted the miss-classification rate and the ROC curves. The following combinations of p and n have been used:

1. p=50 and n=500;
2. p=100 and n=200;
3. p=100 and n=500;
4. p=100 and n=1000;
5. p=150 and n=500.

These combinations allowed us to compare different results of the same methods when p is fixed (100 edges) and n varies (200,500,1000) and when p varies (50, 100, 150) and n is fixed (500). We have considered a penalization factor lambda which varies from 0.0 to 0.2.

2.2. Miss-classification error

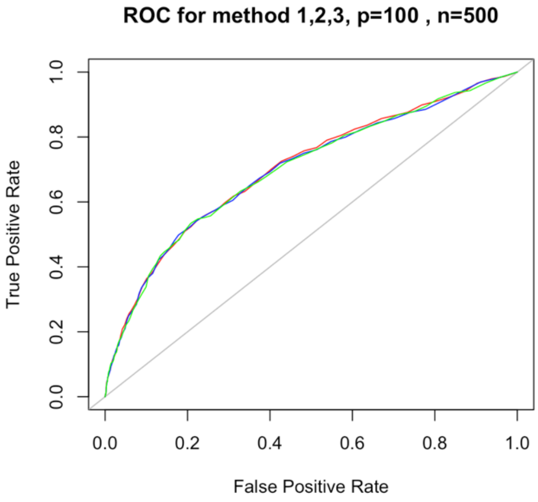
The first observation that we made is about the shape of the graph of the miss-classification error vs lambda. In the following graph lambda varies from 0 (no penalization) to 0.2 (maximum penalization considered), p is equal to 100 and n is equal to 500.



We can see that the miss-classification error start from 0.9 when lambda is equal to 0 and tends towards 0.1 when lambda is big. This can be explained easily recalling the setting of the problem. To generate the set of the vertices, we have use a Bernoulli random variable: a couple of edges is connected with probability 0.1 and it is not connected with probability 0.9. When lambda is equal to zero the solution of the regression it is not sparse and each parameter is different from zero. Thus, all vertices are estimated to be existing and we make an error for all the vertices which are not existing in the real set (roughly 90% of the total possible combinations of edges). Conversely, if lambda is big, the penalization term is too strong and all vertices are estimated to not exist. In this case we make an error each time that the vertices exist (roughly 10% of the times). It is important to underline that the miss-classification rate is not a good index too choose the model. In fact, in our example, it suggests picking a model which predicts that all vertices are not existing.

2.3. ROC curves: comparison between different models

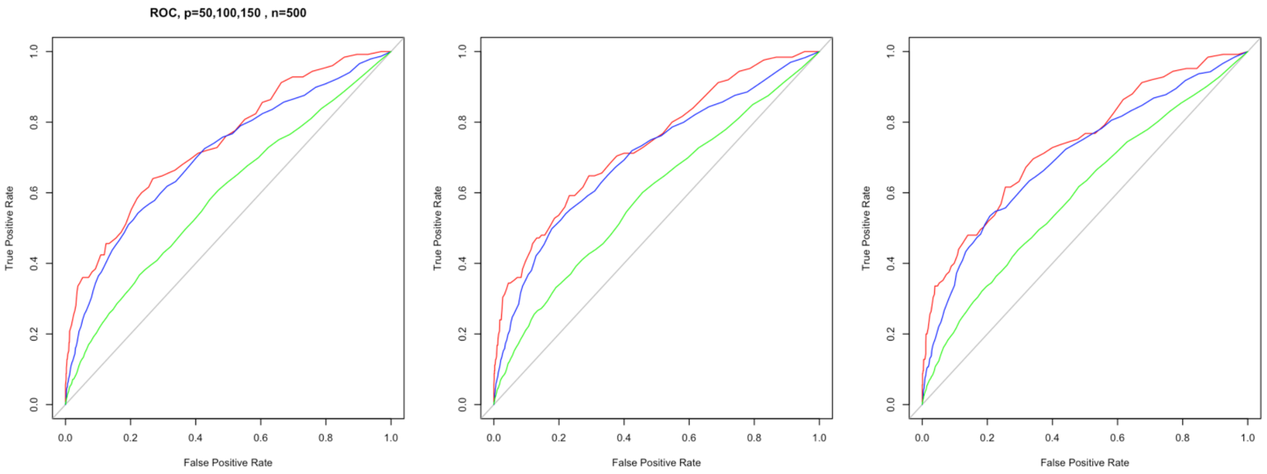
To compare the different estimation methods, we plotted the ROC curves for each method for different level of p and n. The following graph shows the ROC curve for each method for p=100 and n=500.



We can see that the outcome of the different calibration methods is very similar. The area under the ROC curve (AUROC) for method 1, 2 and 3 is respectively 0.704, 0.701 and 0.700. Similar results are obtained with different level of p and n. Hence, the difference between the three estimation methods is negligible.

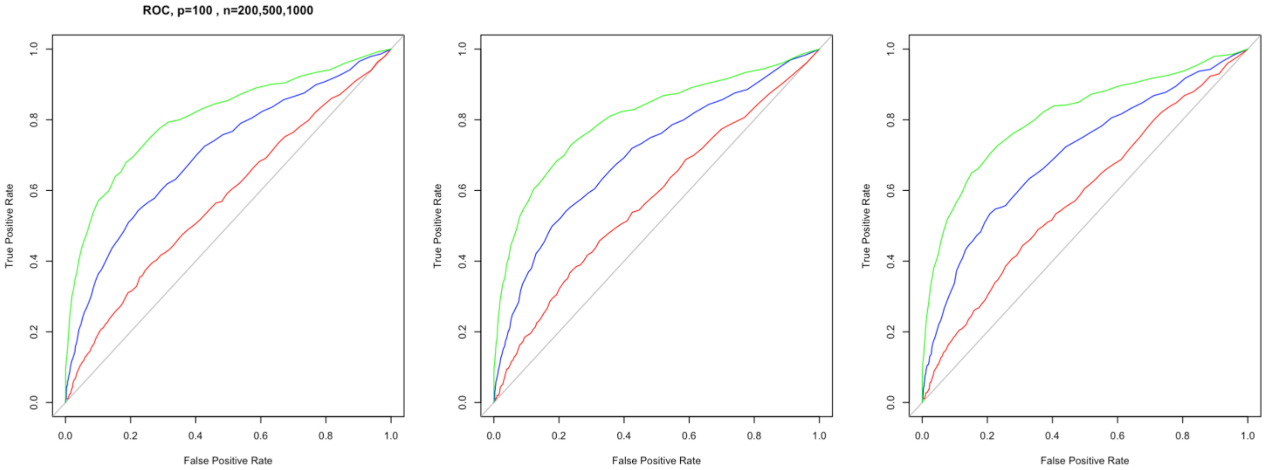
2.4. ROC curves: comparison between different p and n

In this paragraph, we analyze the how the quality of the estimation changes at different levels of p and n. Firstly we will analyze what happen when p changes. The following graphs show the ROC curves for methods 1,2,3, n=500 and p=50 (red line), p=100 (blue line) and p=150 (green line).



We can see that when p increases the AUROC decreases. This result is intuitive. Indeed, when the information available (n) is fixed, if the complexity of the problem (p) increases the quality of the estimation decreases.

When p is fixed, and n varies we expect that the AUROC will increase as well. Ceteris paribus, if the information available increase, the quality of the estimation should increase. The following graphs show the ROC curves for methods 1,2,3, p=100 and n=200 (red line), p=500 (blue line) and p=1000 (green line).



As expected, the ROC curves with n=1000 dominates the ROC curves for n=500 and n=200.

2.5. ROC curves: numerical results

The value of the AUROC in the five cases considered are reported in the following table.

				AUROC	
p	n	n/p	$\hat{E}_1$	$\hat{E}_2$	$\hat{E}_3$
50	500	10	0.7400509	0.7372255	0.7371309
100	200	2	0.5745979	0.5754654	0.5823614
100	500	5	0.7040512	0.7007001	0.6999867
100	1000	10	0.8066268	0.8045442	0.8079276
150	500	3.33	0.5905767	0.5911834	0.5953109

From Table 1, increasing p for same sample size n, AUROC decreases for all 3 methods when p increases from 50 to 100 and from 100 to 150. For same value of p with increasing sample size n, it is shown that AUROC is likely to increase based on our results for n = 200, 500 and 1000. Among the 6 combinations, the setting with $\mathrm{p} = 100$, $\mathrm{n} = 1000$ shows the highest value of AUROC (i.e. approx. 0.80), while the lowest is from with $\mathrm{p} = 100$, $\mathrm{n} = 200$.

Finally, we run the code 50 times with p=100 and n=500 to evaluate the mean and the variance of the AUROC estimation. The results are reported in table 2.

AUROC	Mean	Standard deviation	Variance
Method 1	0.6512545	0.02899642	0.0008407924
Method 2	0.6500842	0.02859026	0.0008174031
Method 3	0.6533424	0.02875512	0.0008268572

2.6 Conclusions

In this exercise, we compared the three estimations methods and they gave approximately the same results in term of AUROC and miss-classification rate. There is no evidence that one method is significantly better than the others. However, running the algorithm 50 times the graphic lasso regression perform slightly better than the node-wise lasso regression.

We evaluated the performance of the three methods with various p and n. We concluded that if p increases the AUROC decreases, while if n increases the AUROC increases as well.

Bibliography

Chen, T. He, T. Benesty, M. Khotilovich, V. and Tang, Y. (2017). *xgboost: Extreme Gradient Boosting. R package version 0.6-4*. https://CRAN.R-project.org/package=xgboost

Friedman J., Hastie T., Tibshirani R. (2010). *Regularization Paths for Generalized Linear Models via Coordinate Descent*. Journal of Statistical Software, 33(1), 1-22. URL http://www.jstatsoft.org/v33/i01/.

Friedman, J. Hastie, T. Tibshirani, R. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. New York: Springer-Verlang, ch. 17.

Friedman J, Hastie T. and Tibshirani R. (2014). *glasso: Graphical lasso- estimation of Gaussian graphical models*. R package version 1.8. https://CRAN.R-project.org/package=glasso

Liaw A. and Wiener M. (2002). *Classification and Regression by randomForest.* R News 2(3), 18--22.

Kuhn M. Contributions from Jed Wing, Steve Weston, Andre Williams, Chris Keefer, Allan Engelhardt, Tony Cooper, Zachary Mayer, Brenton Kenkel, the R Core Team, Michael Benesty, Reynald Lescarbeau, Andrew Ziem, Luca Scrucca, Yuan Tang, Can Candan and Tyler Hunt. (2017). *caret: Classification and Regression Training*. R package version 6.0-77. https://CRAN.R-project.org/package=caret

Ridgeway G. with contributions from others (2017). *gbm: Generalized Boosted Regression Models. R package version 2.1.3*. https://CRAN.R-project.org/package=gbm

Ripley, B. (2016). *tree: Classification and Regression Trees*. R package version 1.0-37.

Venables, W. N. & Ripley, B. D. (2002) *Modern Applied Statistics with S. Fourth Edition*. Springer, New York. ISBN 0-387-95457-0

Wickham, H. Francois, F. Henry, L. Müller, K. (2017). *dplyr: A Grammar of Data Manipulation*. R package version 0.7.4.

Zeileis, A. and Grothendieck G. (2005). *zoo: S3 Infrastructure for Regular and Irregular Time Series. Journal of Statistical Software*, 14(6), 1-27. doi:10.18637/jss.v014.i06

Greg Ridgeway with contributions from others (2017). gbm: Generalized Boosted Regression Models. R package version 2.1.3. https://CRAN.R-project.org/package=gbm

LSE, ST443, Project, Part 1

Group 9

Michealmas Term 2017

Data Cleaning and Transformation

```
library(dplyr)
```

```
Mode <- function(x) {
  ux <- unique(x)
  ux[which.max(tabulate(match(x, ux)))]
}
```

```
train_raw = read.csv("train.csv", row.names = "Id", stringsAsFactors=FALSE)
testing_raw = read.csv("test.csv", row.names = "Id", stringsAsFactors=FALSE)
```

```
#combining train and test data for quicker data prep
testing_raw$SalePrice <- NA
train_raw$isTrain <- 1
testing_raw$isTrain <- 0
df <- rbind(train_raw,testing_raw)
```

Missing Values and imputation.

```
colSums(sapply(df, is.na))
```

```
##    MSSubClass      MSZoning   LotFrontage       LotArea        Street
##             0             4           486             0             0
##         Alley      LotShape   LandContour     Utilities     LotConfig
##          2721             0             0             2             0
##     LandSlope  Neighborhood    Condition1    Condition2      BldgType
##             0             0             0             0             0
##    HouseStyle   OverallQual   OverallCond     YearBuilt  YearRemodAdd
##             0             0             0             0             0
##     RoofStyle      RoofMatl   Exterior1st   Exterior2nd    MasVnrType
##             0             0             1             1            24
##    MasVnrArea     ExterQual     ExterCond    Foundation      BsmtQual
##            23             0             0             0            81
##      BsmtCond  BsmtExposure  BsmtFinType1    BsmtFinSF1  BsmtFinType2
##            82            82            79             1            80
##    BsmtFinSF2     BsmtUnfSF   TotalBsmtSF       Heating     HeatingQC
##             1             1             1             0             0
##    CentralAir    Electrical     X1stFlrSF     X2ndFlrSF  LowQualFinSF
##             0             1             0             0             0
##     GrLivArea  BsmtFullBath  BsmtHalfBath      FullBath      HalfBath
##             0             2             2             0             0
##  BedroomAbvGr  KitchenAbvGr   KitchenQual  TotRmsAbvGrd    Functional
##             0             0             1             0             2
##    Fireplaces   FireplaceQu    GarageType   GarageYrBlt  GarageFinish
##             0          1420           157           159           159
##    GarageCars    GarageArea    GarageQual    GarageCond    PavedDrive
##             1             1           159           159             0
##    WoodDeckSF   OpenPorchSF EnclosedPorch    X3SsnPorch   ScreenPorch
##             0             0             0             0             0
##      PoolArea        PoolQC         Fence   MiscFeature       MiscVal
##             0          2909          2348          2814             0
##        MoSold        YrSold      SaleType SaleCondition     SalePrice
##             0             0             1             0          1459
##       isTrain
##             0
```

```
df[,c('PoolQC','PoolArea')] %>%
  group_by(PoolQC) %>%
  summarise(mean = mean(PoolArea), counts = n())
```

```
## # A tibble: 4 x 3
##   PoolQC        mean counts
##    <chr>       <dbl>  <int>
## 1     Ex 359.7500000      4
## 2     Fa 583.5000000      2
## 3     Gd 648.5000000      4
## 4   <NA>   0.4719835   2909
```

```
df[(df$PoolArea > 0) & is.na(df$PoolQC),c('PoolQC','PoolArea')]
```

```
##      PoolQC PoolArea
## 2421   <NA>      368
## 2504   <NA>      444
## 2600   <NA>      561
```

Imputing the missing values of pools, if no pool then assign ‘None’

```
df[2421,'PoolQC'] = 'Ex'
df[2504,'PoolQC'] = 'Ex'
df[2600,'PoolQC'] = 'Fa'
df$PoolQC[is.na(df$PoolQC)] = 'None'
```

```
garage.cols <- c('GarageArea', 'GarageCars', 'GarageQual', 'GarageFinish', 'GarageCond', 'GarageType')
#df[is.na(df$GarageCond),garage.cols]
```

Imputing the missing values of Garages. If the no garage then assigning 0 or None

```
#length(which(df$GarageYrBlt == df$YearBuilt))
df[(df$GarageArea > 0) & is.na(df$GarageYrBlt), c(garage.cols, 'GarageYrBlt')]
```

```
##      GarageArea GarageCars GarageQual GarageFinish GarageCond GarageType
## 2127        360          1       <NA>         <NA>       <NA>     Detchd
## NA           NA         NA       <NA>         <NA>       <NA>       <NA>
##      GarageYrBlt
## 2127          NA
## NA            NA
```

```
df$GarageYrBlt[2127] <- df$YearBuilt[2127]
df[2127, 'GarageQual'] <- Mode(df$GarageQual)
df[2127, 'GarageFinish'] <- Mode(df$GarageFinish)
df[2127, 'GarageCond'] <- Mode(df$GarageCond)
df$GarageYrBlt[which(is.na(df$GarageYrBlt))] <- 0
```

to numeric - 0, to categorical = 'None'

```
for(i in garage.cols){
if (sapply(df[i], is.numeric) == TRUE){
    df[,i][which(is.na(df[,i]))] <- 0
  }
  else{
    df[,i][which(is.na(df[,i]))] <- "None"
  }
}
```

```
df$KitchenQual[which(is.na(df$KitchenQual))] <- Mode(df$KitchenQual)
```

```
df[is.na(df$MSZoning),c('MSZoning','MSSubClass')]
```

```
##      MSZoning MSSubClass
## 1916     <NA>         30
## 2217     <NA>         20
## 2251     <NA>         70
## 2905     <NA>         20
```

```
table(df$MSZoning, df$MSSubClass)
```

```
##
##               20   30   40   45   50   60   70   75   80   85   90  120  150
##   C (all)    3    8    0    0    7    0    4    0    0    0    0    0    0
##   FV        34    0    0    0    0   43    0    0    0    0    0   19    0
##   RH         4    2    0    1    2    0    3    0    0    0    4    6    0
##   RL      1016   61    4    6  159  529   57    9  115   47   92  117    1
##   RM        20   67    2   11  119    3   63   14    3    1   13   40    0
##
##              160  180  190
##   C (all)    0    0    3
##   FV        43    0    0
##   RH         0    0    4
##   RL        21    0   31
##   RM        64   17   23
```

```
df$MSZoning[c(2217, 2905)] = 'RL'
df$MSZoning[c(1916, 2251)] = 'RM'
```

There are 486 Nas in LotFrontage, setting the NAs to median.

```
df$LotFrontage[which(is.na(df$LotFrontage))] <- median(df$LotFrontage,na.rm = T)
```

There are 2721 NAs in Alley, set them equal to 'None'

```
df$Alley[which(is.na(df$Alley))] <- "None"
```

One of the data is missing the rest set to 0 or 'None'

```
#df[(df$MasVnrArea > 0) & (is.na(df$MasVnrType)),c('MasVnrArea','MasVnrType')]
df[2611, 'MasVnrType'] = 'BrkFace'
df$MasVnrType[is.na(df$MasVnrType)] = 'None'
df$MasVnrArea[is.na(df$MasVnrArea)] = 0
```

For small number of NAs we apply Mode to the categorical, and median to the continous

```
for(i in colnames(df[,sapply(df, is.character)])){
  if (sum(is.na(df[,i])) < 5){
    df[,i][which(is.na(df[,i]))] <- Mode(df[,i])
  }
}

for(i in colnames(df[,sapply(df, is.integer)])){
  if (sum(is.na(df[,i])) < 5){
    df[,i][which(is.na(df[,i]))] <- median(df[,i], na.rm = T)
  }
}
```

For large number of NAs we apply string "None" to the categorical as a seperate Level, and 0 to the continous

```
for(i in colnames(df[,sapply(df, is.character)])){
    df[,i][which(is.na(df[,i]))] <- "None"
}
```

We have filled in all the missing values. The remaining ones are the SalesPrice in the predicting Dataset that is fine!

```
#colSums(sapply(df, is.na))
sum(is.na(df)) == 1459
```

```
## [1] TRUE
```

Creating categorical variables and checking whether and some problem appear. if f.e testing has more levels than the training data!

```
train_df <- df[df$isTrain==1,]
test_df <- df[df$isTrain==0,]

train_df$isTrain <- NULL
test_df$isTrain <- NULL
test_df$SalePrice <- NULL
```

```
train_df$MSSubClass <- as.factor(train_df$MSSubClass)
test_df$MSSubClass <- as.factor(test_df$MSSubClass)

train_df$OverallQual <- as.factor(train_df$OverallQual)
test_df$OverallQual <- as.factor(test_df$OverallQual)

train_df$OverallCond <- as.factor(train_df$OverallCond)
test_df$OverallCond <- as.factor(test_df$OverallCond)
```

```
for(i in colnames(train_df[,sapply(train_df, is.character)])){
    train_df[,i] <- as.factor(train_df[,i])
}
for(i in colnames(test_df[,sapply(test_df, is.character)])){
    test_df[,i] <- as.factor(test_df[,i])
}
```

```
#Check is some there are more levels in some of the categorical factors in the testing compared to the
for(i in colnames(train_df[,sapply(train_df, is.factor)])){
  if (length(levels(train_df[,i])) < length(levels(test_df[,i]))) {
    print(i)
    print(levels(train_df[,i]))
    print(levels(test_df[,i]))
  }
}
```

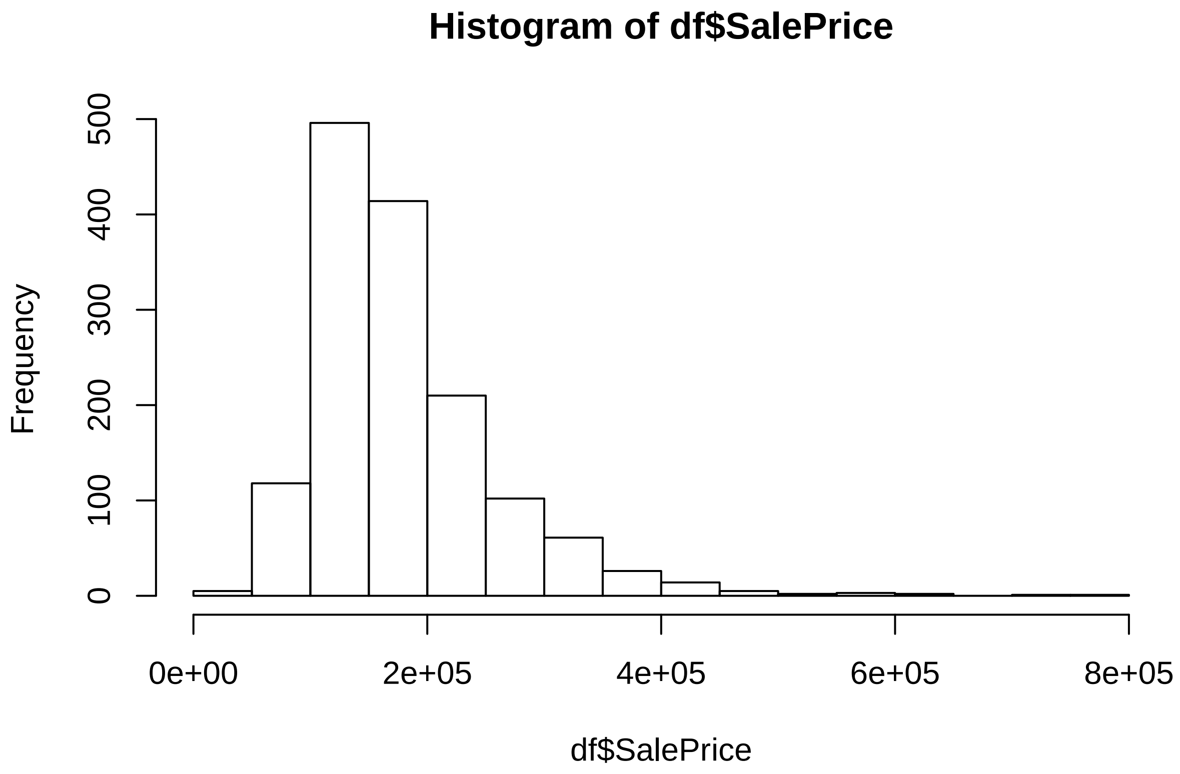
```
## [1] "MSSubClass"
##  [1] "20"  "30"  "40"  "45"  "50"  "60"  "70"  "75"  "80"  "85"  "90"
## [12] "120" "160" "180" "190"
##  [1] "20"  "30"  "40"  "45"  "50"  "60"  "70"  "75"  "80"  "85"  "90"
## [12] "120" "150" "160" "180" "190"
```

level '150' appears once in the testing data and no such level is in the training data. Remove this level.

```
#df[df$MSSubClass == 150,]
df[df$MSSubClass == 150,"MSSubClass"] <- 120
```

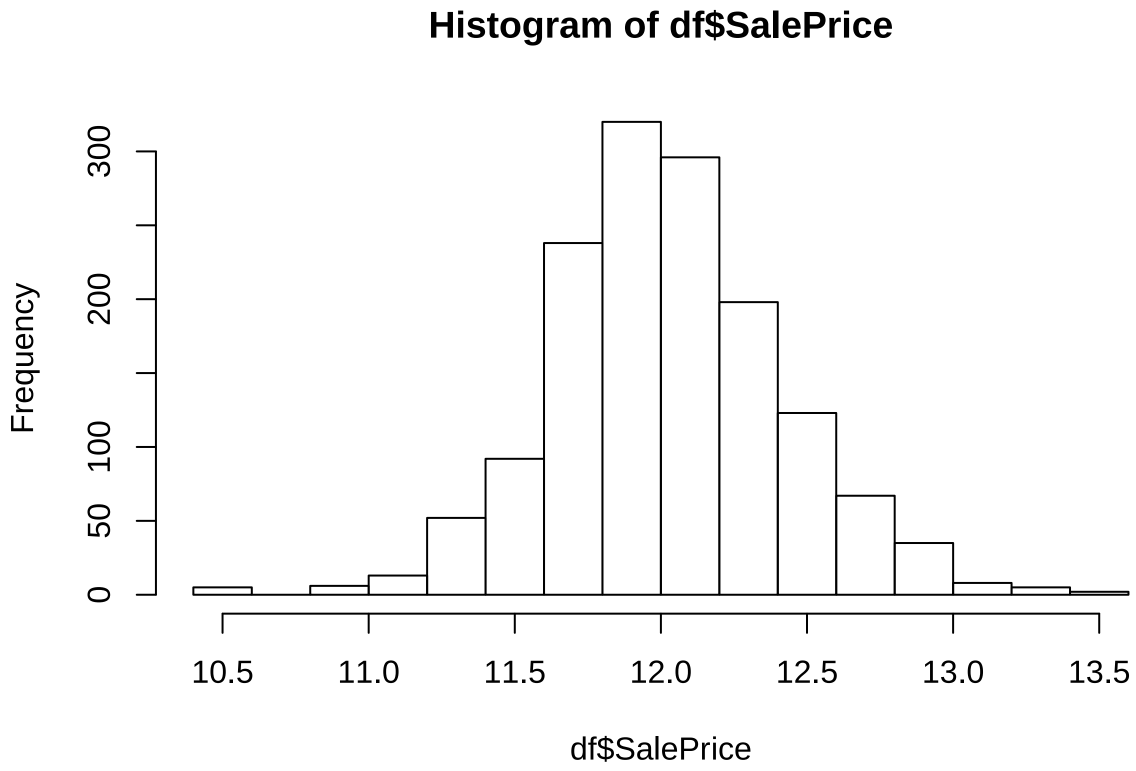
Transformations

```
hist(df$SalePrice)
```



```
df$SalePrice <- log(df$SalePrice)
```

```
hist(df$SalePrice)
```



Create factors in the combined dataframe and split the data into testing and training.

```
for(i in colnames(df[,sapply(df, is.character)])){
    df[,i] <- as.factor(df[,i])
}

df$MSSubClass <- as.factor(df$MSSubClass)
df$OverallQual <- as.factor(df$OverallQual)
df$OverallCond <- as.factor(df$OverallCond)

### THINGS TO CONSIDER:
#df$GarageYrBlt <- as.factor(df$GarageYrBlt) # treat as factor as some of them are '0'
#add years as dummies - POSSIBILITY - but a problem appears, the algorithms cannot treat categorical va
#df$YearBuilt <- as.factor(df$YearBuilt)
#df$YearRemodAdd <- as.factor(df$YearRemodAdd)
#df$YrSold <- as.factor(df$YrSold)
```

```
train_df <- df[df$isTrain==1,]
test_df <- df[df$isTrain==0,]

train_df$isTrain <- NULL
test_df$isTrain <- NULL
test_df$SalePrice <- NULL
```

```r
str(df)
```

```
## 'data.frame':    2919 obs. of  81 variables:
##  $ MSSubClass   : Factor w/ 15 levels "20","30","40",..: 6 1 6 7 6 5 1 6 5 15 ...
##  $ MSZoning     : Factor w/ 5 levels "C (all)","FV",..: 4 4 4 4 4 4 4 4 5 4 ...
##  $ LotFrontage  : int  65 80 68 60 84 85 75 68 51 50 ...
##  $ LotArea      : int  8450 9600 11250 9550 14260 14115 10084 10382 6120 7420 ...
##  $ Street       : Factor w/ 2 levels "Grvl","Pave": 2 2 2 2 2 2 2 2 2 2 ...
##  $ Alley        : Factor w/ 3 levels "Grvl","None",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ LotShape     : Factor w/ 4 levels "IR1","IR2","IR3",..: 4 4 1 1 1 1 4 1 4 4 ...
##  $ LandContour  : Factor w/ 4 levels "Bnk","HLS","Low",..: 4 4 4 4 4 4 4 4 4 4 ...
##  $ Utilities    : Factor w/ 2 levels "AllPub","NoSeWa": 1 1 1 1 1 1 1 1 1 1 ...
##  $ LotConfig    : Factor w/ 5 levels "Corner","CulDSac",..: 5 3 5 1 3 5 5 1 5 1 ...
##  $ LandSlope    : Factor w/ 3 levels "Gtl","Mod","Sev": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Neighborhood : Factor w/ 25 levels "Blmngtn","Blueste",..: 6 25 6 7 14 12 21 17 18 4 ...
##  $ Condition1   : Factor w/ 9 levels "Artery","Feedr",..: 3 2 3 3 3 3 3 5 1 1 ...
##  $ Condition2   : Factor w/ 8 levels "Artery","Feedr",..: 3 3 3 3 3 3 3 3 3 1 ...
##  $ BldgType     : Factor w/ 5 levels "1Fam","2fmCon",..: 1 1 1 1 1 1 1 1 1 2 ...
##  $ HouseStyle   : Factor w/ 8 levels "1.5Fin","1.5Unf",..: 6 3 6 6 6 1 3 6 1 2 ...
##  $ OverallQual  : Factor w/ 10 levels "1","2","3","4",..: 7 6 7 7 8 5 8 7 7 5 ...
##  $ OverallCond  : Factor w/ 9 levels "1","2","3","4",..: 5 8 5 5 5 5 5 6 5 6 ...
##  $ YearBuilt    : int  2003 1976 2001 1915 2000 1993 2004 1973 1931 1939 ...
##  $ YearRemodAdd : int  2003 1976 2002 1970 2000 1995 2005 1973 1950 1950 ...
##  $ RoofStyle    : Factor w/ 6 levels "Flat","Gable",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ RoofMatl     : Factor w/ 8 levels "ClyTile","CompShg",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ Exterior1st  : Factor w/ 15 levels "AsbShng","AsphShn",..: 13 9 13 14 13 13 13 7 4 9 ...
##  $ Exterior2nd  : Factor w/ 16 levels "AsbShng","AsphShn",..: 14 9 14 16 14 14 14 7 16 9 ...
##  $ MasVnrType   : Factor w/ 4 levels "BrkCmn","BrkFace",..: 2 3 2 3 2 3 4 4 3 3 ...
##  $ MasVnrArea   : num  196 0 162 0 350 0 186 240 0 0 ...
##  $ ExterQual    : Factor w/ 4 levels "Ex","Fa","Gd",..: 3 4 3 4 3 4 3 4 4 4 ...
##  $ ExterCond    : Factor w/ 5 levels "Ex","Fa","Gd",..: 5 5 5 5 5 5 5 5 5 5 ...
##  $ Foundation   : Factor w/ 6 levels "BrkTil","CBlock",..: 3 2 3 1 3 6 3 2 1 1 ...
##  $ BsmtQual     : Factor w/ 5 levels "Ex","Fa","Gd",..: 3 3 3 5 3 3 1 3 5 5 ...
##  $ BsmtCond     : Factor w/ 5 levels "Fa","Gd","None",..: 5 5 5 2 5 5 5 5 5 5 ...
##  $ BsmtExposure : Factor w/ 5 levels "Av","Gd","Mn",..: 4 2 3 4 1 4 1 3 4 4 ...
##  $ BsmtFinType1 : Factor w/ 7 levels "ALQ","BLQ","GLQ",..: 3 1 3 1 3 3 3 1 7 3 ...
##  $ BsmtFinSF1   : num  706 978 486 216 655 ...
##  $ BsmtFinType2 : Factor w/ 7 levels "ALQ","BLQ","GLQ",..: 7 7 7 7 7 7 7 2 7 7 ...
##  $ BsmtFinSF2   : num  0 0 0 0 0 0 0 32 0 0 ...
##  $ BsmtUnfSF    : num  150 284 434 540 490 64 317 216 952 140 ...
##  $ TotalBsmtSF  : num  856 1262 920 756 1145 ...
##  $ Heating      : Factor w/ 6 levels "Floor","GasA",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ HeatingQC    : Factor w/ 5 levels "Ex","Fa","Gd",..: 1 1 1 3 1 1 1 1 3 1 ...
##  $ CentralAir   : Factor w/ 2 levels "N","Y": 2 2 2 2 2 2 2 2 2 2 ...
##  $ Electrical   : Factor w/ 5 levels "FuseA","FuseF",..: 5 5 5 5 5 5 5 5 2 5 ...
##  $ X1stFlrSF    : int  856 1262 920 961 1145 796 1694 1107 1022 1077 ...
##  $ X2ndFlrSF    : int  854 0 866 756 1053 566 0 983 752 0 ...
##  $ LowQualFinSF : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ GrLivArea    : int  1710 1262 1786 1717 2198 1362 1694 2090 1774 1077 ...
##  $ BsmtFullBath : int  1 0 1 1 1 1 1 1 0 1 ...
##  $ BsmtHalfBath : int  0 1 0 0 0 0 0 0 0 0 ...
##  $ FullBath     : int  2 2 2 1 2 1 2 2 2 1 ...
##  $ HalfBath     : int  1 0 1 0 1 1 0 1 0 0 ...
##  $ BedroomAbvGr : int  3 3 3 3 4 1 3 3 2 2 ...
##  $ KitchenAbvGr : int  1 1 1 1 1 1 1 1 2 2 ...
##  $ KitchenQual  : Factor w/ 4 levels "Ex","Fa","Gd",..: 3 4 3 3 3 4 3 4 4 4 ...
##  $ TotRmsAbvGrd : int  8 6 6 7 9 5 7 7 8 5 ...
##  $ Functional   : Factor w/ 7 levels "Maj1","Maj2",..: 7 7 7 7 7 7 7 7 3 7 ...
##  $ Fireplaces   : int  0 1 1 1 1 0 1 2 2 2 ...
##  $ FireplaceQu  : Factor w/ 6 levels "Ex","Fa","Gd",..: 4 6 6 3 6 4 3 6 6 6 ...
##  $ GarageType   : Factor w/ 7 levels "2Types","Attchd",..: 2 2 2 6 2 2 2 2 6 2 ...
##  $ GarageYrBlt  : num  2003 1976 2001 1998 2000 ...
##  $ GarageFinish : Factor w/ 4 levels "Fin","None","RFn",..: 3 3 3 4 3 4 3 3 4 3 ...
##  $ GarageCars   : num  2 2 2 3 3 2 2 2 2 1 ...
##  $ GarageArea   : num  548 460 608 642 836 480 636 484 468 205 ...
##  $ GarageQual   : Factor w/ 6 levels "Ex","Fa","Gd",..: 6 6 6 6 6 6 6 6 2 3 ...
##  $ GarageCond   : Factor w/ 6 levels "Ex","Fa","Gd",..: 6 6 6 6 6 6 6 6 6 6 ...
##  $ PavedDrive   : Factor w/ 3 levels "N","P","Y": 3 3 3 3 3 3 3 3 3 3 ...
##  $ WoodDeckSF   : int  0 298 0 0 192 40 255 235 90 0 ...
##  $ OpenPorchSF  : int  61 0 42 35 84 30 57 204 0 4 ...
##  $ EnclosedPorch: int  0 0 0 272 0 0 0 228 205 0 ...
##  $ X3SsnPorch   : int  0 0 0 0 0 320 0 0 0 0 ...
##  $ ScreenPorch  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ PoolArea     : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ PoolQC       : Factor w/ 4 levels "Ex","Fa","Gd",..: 4 4 4 4 4 4 4 4 4 4 ...
##  $ Fence        : Factor w/ 5 levels "GdPrv","GdWo",..: 5 5 5 5 5 3 5 5 5 5 ...
##  $ MiscFeature  : Factor w/ 5 levels "Gar2","None",..: 2 2 2 2 2 4 2 4 2 2 ...
##  $ MiscVal      : int  0 0 0 0 0 700 0 350 0 0 ...
##  $ MoSold       : int  2 5 9 2 12 10 8 11 4 1 ...
##  $ YrSold       : int  2008 2007 2008 2006 2008 2009 2007 2009 2008 2008 ...
##  $ SaleType     : Factor w/ 9 levels "COD","Con","ConLD",..: 9 9 9 9 9 9 9 9 9 9 ...
##  $ SaleCondition: Factor w/ 6 levels "Abnorml","AdjLand",..: 5 5 5 1 5 5 5 5 1 5 ...
##  $ SalePrice    : num  12.2 12.1 12.3 11.8 12.4 ...
##  $ isTrain      : num  1 1 1 1 1 1 1 1 1 1 ...
```

Regression Code

```
library(tree) # Normal Tree
library(glmnet) # Lasso/Ridge
library(randomForest) # random Forest
library(xgboost) # boosting trees
library(caret) # for tuning xgboost
library(gbm) # for the gradient boosting model
```

Generating matrix from the data frames, no intercept

```
X_train<- model.matrix(SalePrice~.-1, data = train_df)
y_train <- train_df$SalePrice
X_test <- model.matrix(~.-1, data=test_df)
```

Linear Regression

Regression with all the parameters - just as a benchmark

```
lm_fit_all = lm(SalePrice ~., data = train_df)
#summary(lm_fit_all)
```

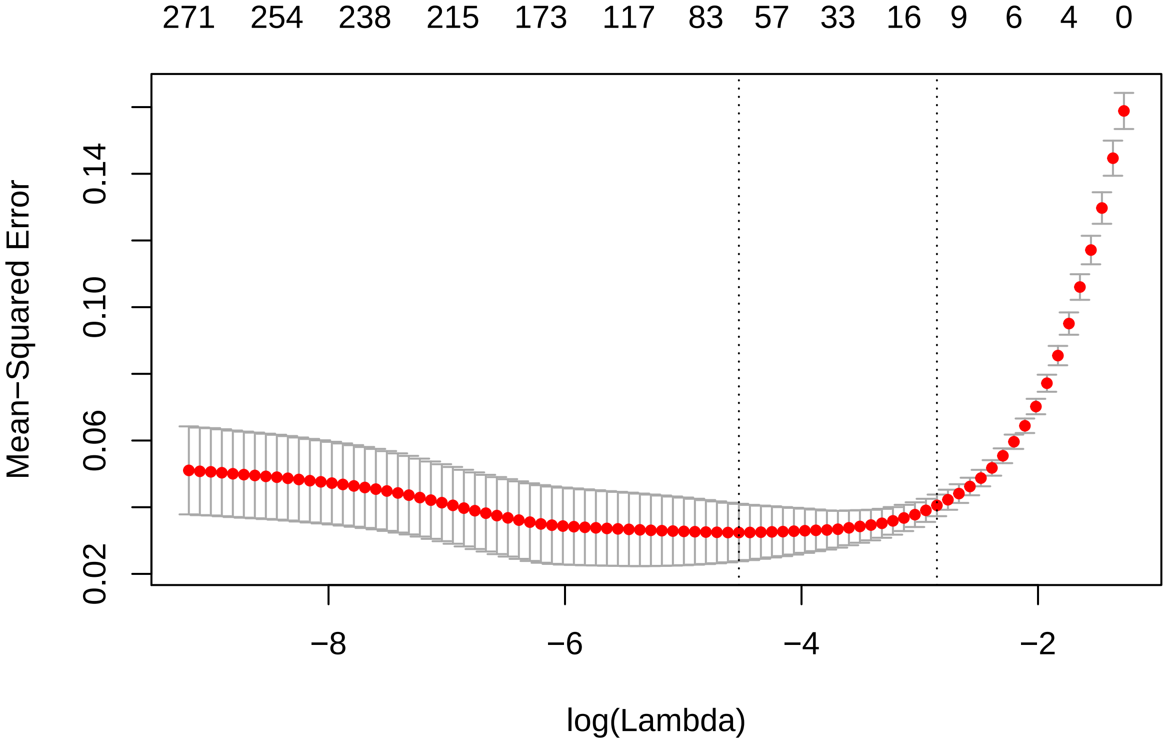
```
prediction_LR_ALL_log <- predict(lm_fit_all, test_df, type="response")
prediction_LR_ALL <- exp(prediction_LR_ALL_log)
```

```
prediction_LR_ALL <- cbind(Id = rownames(test_df), SalesPrice = prediction_LR_ALL)
```

```
write.table(prediction_LR_ALL, file="prediction_LR_ALL.csv",col.names = c("Id","SalePrice"), sep =',',
```

Lasso

```
cv.lasso <-cv.glmnet(X_train, y_train, nfolds = 10, alpha = 1)
plot(cv.lasso)
```



```
penalty_min <- cv.lasso$lambda.min #optimal lambda
penalty_1se <- cv.lasso$lambda.1se # 1 Standard Error Apart
fit.lasso_min <-glmnet(X_train, y_train, alpha = 1, lambda = penalty_min) #estimate the model with min
fit.lasso_1se <-glmnet(X_train, y_train, alpha = 1, lambda = penalty_1se) #estimate the model with 1se
```

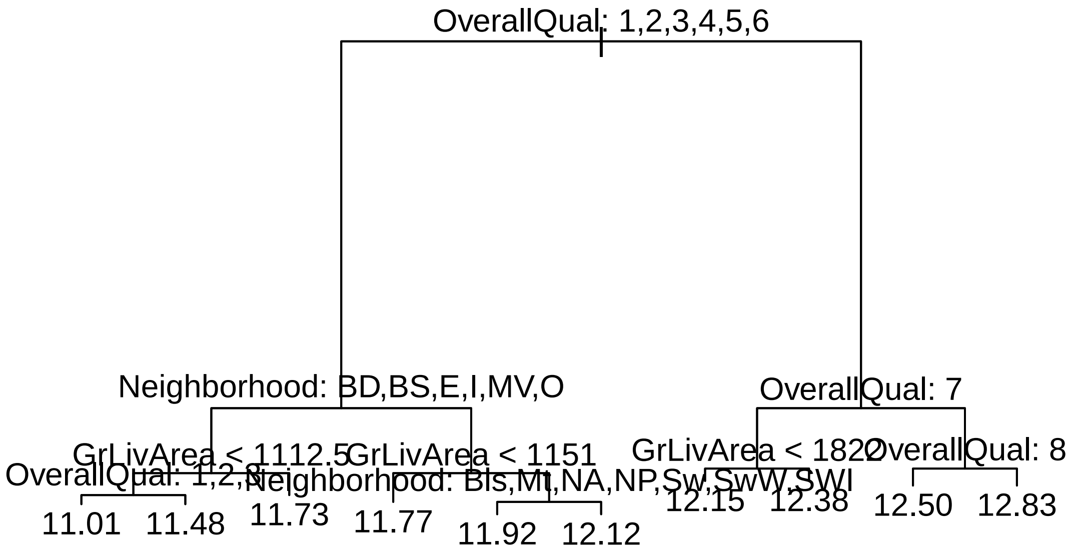
```
prediction_LASSO_min_log <- predict(fit.lasso_min, X_test)
prediction_LASSO_1se_log <- predict(fit.lasso_1se, X_test)

prediction_LASSO_min <- exp(prediction_LASSO_min_log)
prediction_LASSO_1se <- exp(prediction_LASSO_1se_log)
```

```
prediction_LASSO_min <-cbind(Id = rownames(test_df), SalesPrice = prediction_LASSO_min)
prediction_LASSO_1se <- cbind(Id = rownames(test_df), SalesPrice = prediction_LASSO_1se)
```

```
write.table(prediction_LASSO_min, file="prediction_LASSO_min.csv",col.names = c("Id", "SalePrice"), sep
write.table(prediction_LASSO_1se, file="prediction_LASSO_1se.csv",col.names = c("Id", "SalePrice"), sep
```

Regression Tree

```
tree.SalePrice <-tree(SalePrice~., data = train_df)
#summary(tree.SalePrice)
plot(tree.SalePrice)
text(tree.SalePrice, pretty=1)
```



```
cv.SalePrice <-cv.tree(tree.SalePrice, K = 10)
plot(cv.SalePrice$size, cv.SalePrice$dev, type="b")
## In this case, the most complex tree is selected by cross-validation
prune.SalePrice <-prune.tree(tree.SalePrice, best=10)
plot(prune.SalePrice)
text(prune.SalePrice, pretty=1)
cv.SalePrice$dev # no PRUNING DONE
```

```
prediction_TREE_log <- predict(prune.SalePrice ,test_df)
prediction_TREE <- exp(prediction_TREE_log)
```

```
prediction_TREE <- cbind(Id = rownames(X_test), SalesPrice = prediction_TREE)
```

```
write.table(prediction_TREE, file="prediction_TREE.csv",col.names = c("Id", "SalePrice"), sep =',', row
```

Bagging

```
set.seed(1)
bag.SalePrice <-randomForest(SalePrice~., data=train_df, mtry= 79, importance=TRUE)
bag.SalePrice
```

```
prediction_bag_log <- predict(bag.SalePrice, newdata = test_df)
prediction_bag <- exp(prediction_bag_log)
```

```
prediction_bag <- cbind(Id = rownames(X_test), SalesPrice = prediction_bag)
```

```
write.table(prediction_bag, file="prediction_bag.csv",col.names = c("Id", "SalePrice"), sep =',', row.na
```

Random Forest $m = \frac{p}{3}$

```
RF_p3.SalePrice <-randomForest(SalePrice~., data=train_df, mtry = 26, importance=TRUE)
```

```
prediction_RF_p3_log <- predict(RF_p3.SalePrice, newdata = test_df)
prediction_RF_p3 <- exp(prediction_RF_p3_log)
```

```
prediction_RF_p3 <- cbind(Id = rownames(X_test), SalesPrice = prediction_RF_p3)
```

```
write.table(prediction_RF_p3, file="prediction_RF_p3.csv",col.names = c("Id", "SalePrice"), sep =',', r
```

Random Forest $m = \sqrt(p)$

```
## We could change the number of trees grown by randomForest() using ntree argument
RF.SalePrice <-randomForest(SalePrice~., data=train_df, importance=TRUE)
```

```
prediction_RF_log <- predict(RF.SalePrice, newdata = test_df)
prediction_RF <- exp(prediction_RF_log)
```

```
prediction_RF <- cbind(Id = rownames(X_test), SalesPrice = prediction_RF)
```

```
write.table(prediction_RF, file="prediction_RF.csv",col.names = c("Id", "SalePrice"), sep =',', row.nam
```

Tuning Random Forest - selecting 'm'

```
x_train_df <- train_df
x_train_df$SalePrice <- NULL
```

```
results <- rfcv(x_train_df, y_train, cv.fold=10, scale="log", step=0.5)
```

```
results$error.cv
```

```
## We could change the number of trees grown by randomForest() using ntree argument
RF.SalePrice_tuned <-randomForest(SalePrice~., data=train_df, importance=TRUE, ntree = 1000, mtry=38)
```

```
prediction_RF_tuned_log <- predict(RF.SalePrice_tuned, newdata = test_df)
prediction_RF_tuned <- exp(prediction_RF_tuned_log)
```

```
prediction_RF_tuned <- cbind(Id = rownames(X_test), SalesPrice = prediction_RF_tuned)
```

```
write.table(prediction_RF_tuned, file="prediction_RF_tuned.csv",col.names = c("Id", "SalePrice"), sep =
```

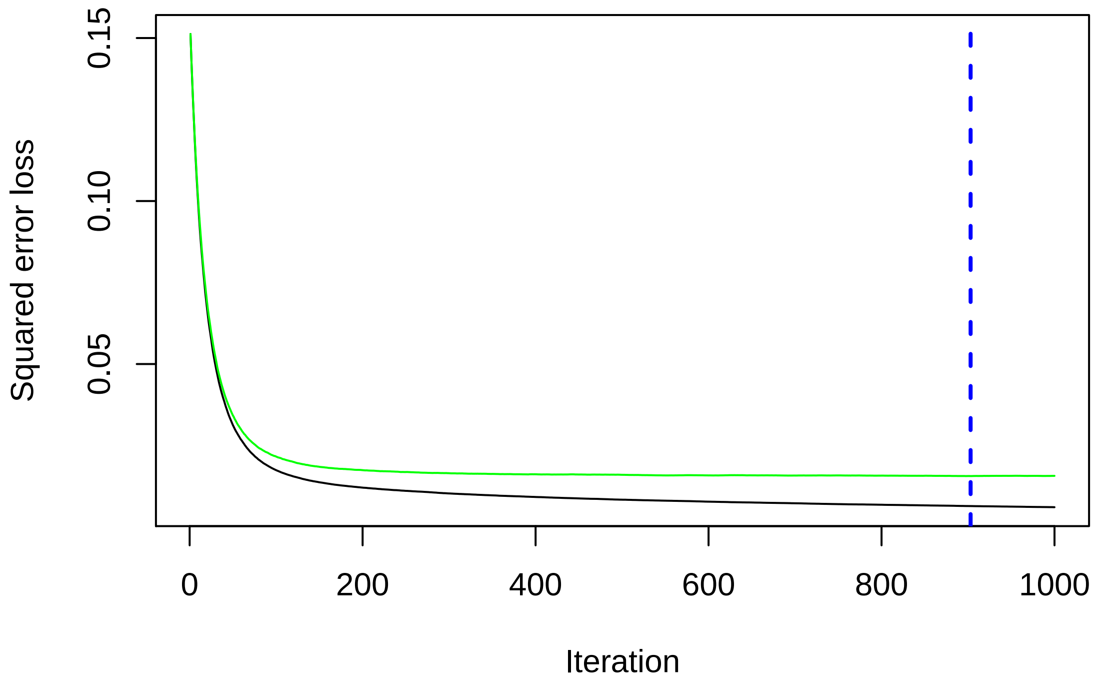
Gradient Boosting Model - library (gbm)

```
set.seed(1)
boost.train = gbm(SalePrice ~. , data=train_df,
                  distribution = "gaussian",
                  n.trees = 1000,
                  shrinkage = 0.05,
                  interaction.depth = 2,
                  bag.fraction = 0.66,
                  cv.folds = 10,
                  verbose = FALSE,
                  n.cores = 8)

min(boost.train$cv.error)
```

```
## [1] 0.01562838
```

```
#attributes(boost.train)
bestTreeForPrediction = gbm.perf(boost.train)
```

```
## Using cv method...
```



```
prediction_BOOST_log <- predict(boost.train, test_df)
prediction_BOOST <- exp(prediction_BOOST_log)
```

```
prediction_BOOST <- cbind(Id = rownames(X_test), SalesPrice = prediction_BOOST)
```

```
write.table(prediction_BOOST, file="prediction_BOOST.csv",col.names = c("Id", "SalePrice"), sep =',', r
```

XGBoost

```
#setup
library(parallel) #checking the number of cores
```

```r
detectCores() #=> 8
dtrain <- xgb.DMatrix(X_train, label = y_train)
```

Cross Validation in XGBoost - CV
Source: StackOverflow

```r
best_param2 = list()
best_rmse = Inf
best_rmse_index = 0
best_seednumber = 1234

for (iter in 1:20) {
    param <- list(objective = "reg:linear",
          max_depth = sample(2:6, 1),
          eta = runif(1, .01, .05),
          gamma = runif(1, 0.0, 0.013),
          subsample = runif(1, .7, .8),
          colsample_bytree = runif(1, .6, .7),
          min_child_weight = sample(1:3, 1)
          )
    cv.nround = 1000
    cv.nfold = 10
    seed.number = sample.int(10000, 1)[[1]]
    set.seed(seed.number)
    mdcv <- xgb.cv(data=dtrain, params = param, nthread=8,
                    nfold=cv.nfold, nrounds=cv.nround,
                    verbose = F, early.stop.rounds=8, maximize=FALSE)

    min_rmse = min(mdcv$evaluation_log[,test_rmse_mean])
    min_rmse_index = which.min(mdcv$evaluation_log[,test_rmse_mean])


    if (min_rmse < best_rmse) {
        best_rmse = min_rmse
        best_rmse_index = min_rmse_index
        best_seednumber = seed.number
        best_param2 = param
    }
    print(iter)
}
```

```r
print(best_param) # rmse = 0.12567
```

```r
nround = best_rmse_index
set.seed(best_seednumber)
model_XGB_tune2 <- xgb.train(data=dtrain, params=best_param2, nrounds=nround, nthread=8)
```

```r
prediction_XGB_tune2_log <- predict(model_XGB_tune2, X_test)
prediction_XGB_tune2 <- exp(prediction_XGB_tune2_log)
```

```r
prediction_XGB_tune2 <- cbind(Id = rownames(X_test), SalesPrice = prediction_XGB_tune2)
```

```r
write.table(prediction_XGB_tune2, file="prediction_XBG_tune2.csv",col.names = c("Id", "SalePrice"), sep
```

Classification Code

```
mean(train_df$SalePrice)
```

Coverting SalePrice into a binary variable

```
train_df$Classifier <- ifelse(train_df$SalePrice <= 12.024,"Low","High")
train_df$Classifier <- as.factor(train_df$Classifier)
train_df$Classifier <- factor(train_df$Classifier, levels = c("Low", "High"))
```

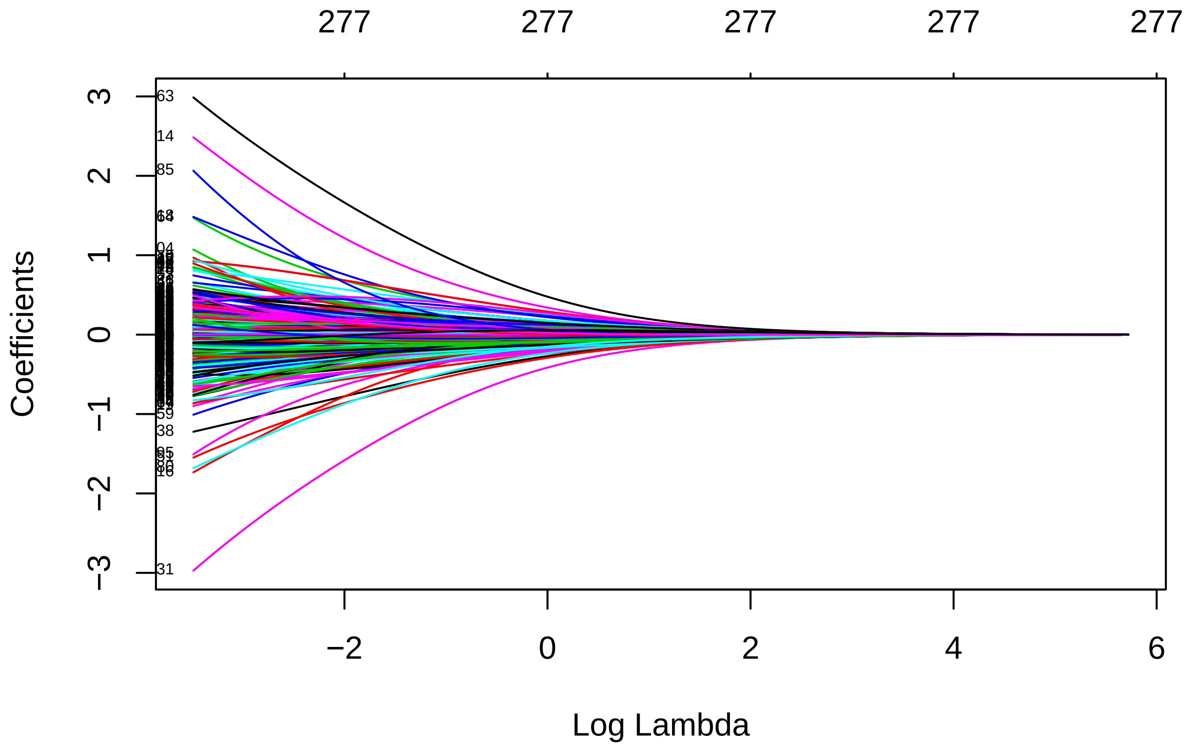
Splitting of data into training and test set

```
attach(train_df)
set.seed(1)
training<- sample(1:nrow(train_df), 1460*0.5)
test.train_df <- train_df[-training,]
train.train_df <- train_df[training,]
Classifier.test <- test.train_df$Classifier
```

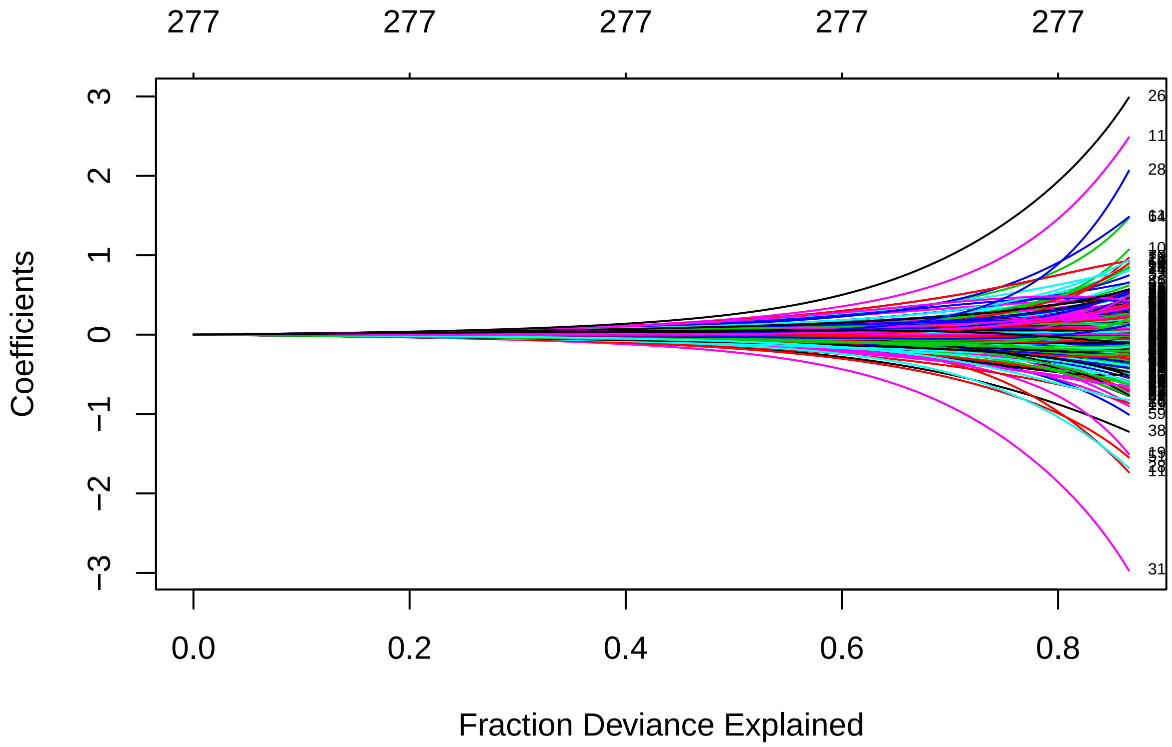
Ridge

```
x.train.classifier <- model.matrix(Classifier~.-SalePrice, data=train.train_df)
y.train.classifier <- train.train_df$Classifier
x.test.classifier <- model.matrix(Classifier~.-SalePrice, data=test.train_df)
```

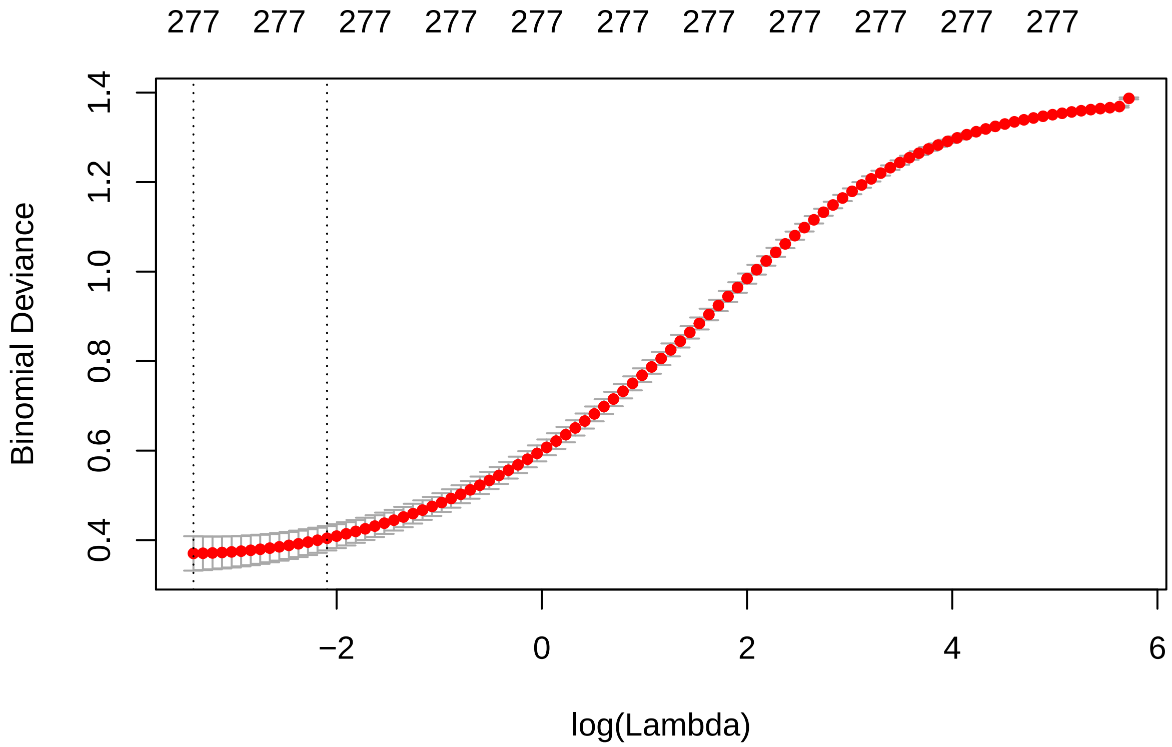
```
fit.ridge <- glmnet(x.train.classifier,y.train.classifier,alpha=0, family="binomial")
plot(fit.ridge, xvar='lambda', label=TRUE)
```



```r
plot(fit.ridge, xvar='dev', label=TRUE)
```



```r
cv.ridge <- cv.glmnet(x.train.classifier,y.train.classifier,alpha=0,family="binomial")
plot(cv.ridge)
```



Ridge-Minimum Lambda

```
ridge.min.lambda=cv.ridge$lambda.min
fit.ridge.min <- glmnet(x.train.classifier,y.train.classifier,alpha=0, family="binomial", lambda=ridge.
prediction.ridge.min.log <- predict(fit.ridge.min, x.test.classifier)
prediction.ridge.min.log.classifier <- (ifelse(prediction.ridge.min.log >0.5,1,0))
table(prediction.ridge.min.log.classifier, Classifier.test)
```

```
##                                     Classifier.test
## prediction.ridge.min.log.classifier Low High
##                                   0 362   40
##                                   1  27  301
```

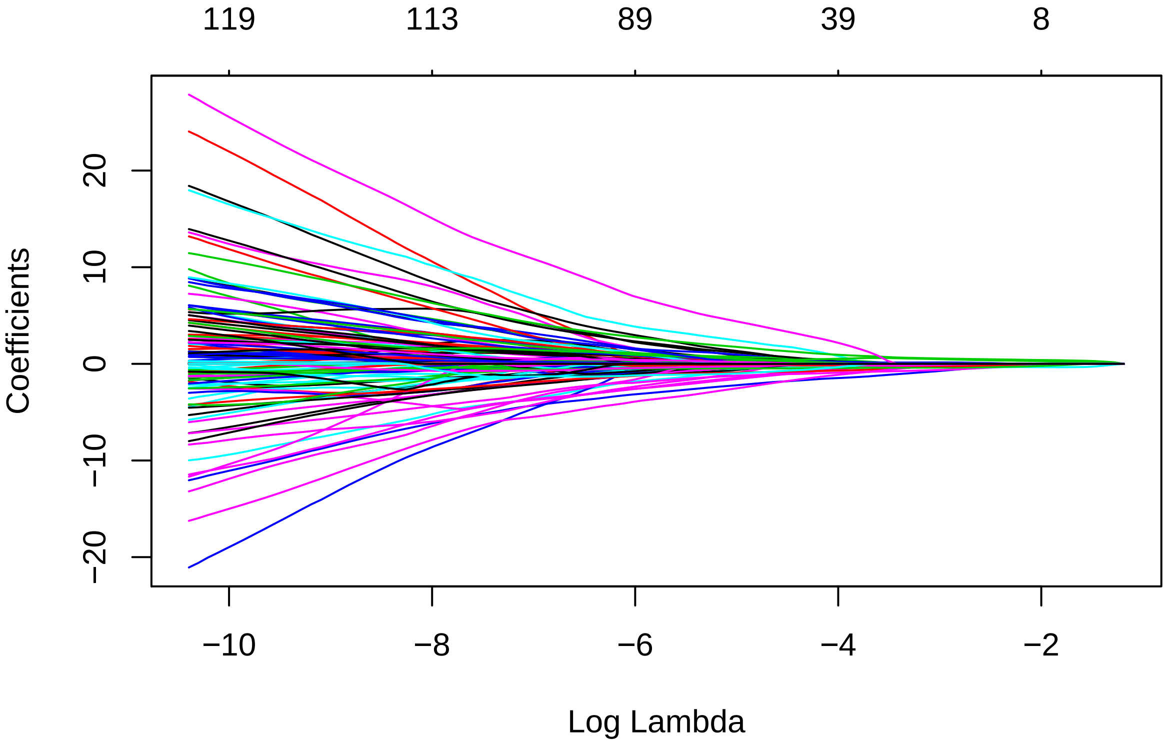
Ridge-1se

```
ridge.1se.lambda=cv.ridge$lambda.1se
fit.ridge.1se <- glmnet(x.train.classifier,y.train.classifier,alpha=0, family="binomial", lambda=ridge.
prediction.ridge.1se.log <- predict(fit.ridge.1se, x.test.classifier)
prediction.ridge.1se.log.classifier <- ifelse(prediction.ridge.1se.log >0.5,1,0)
table(prediction.ridge.1se.log.classifier, Classifier.test)
```

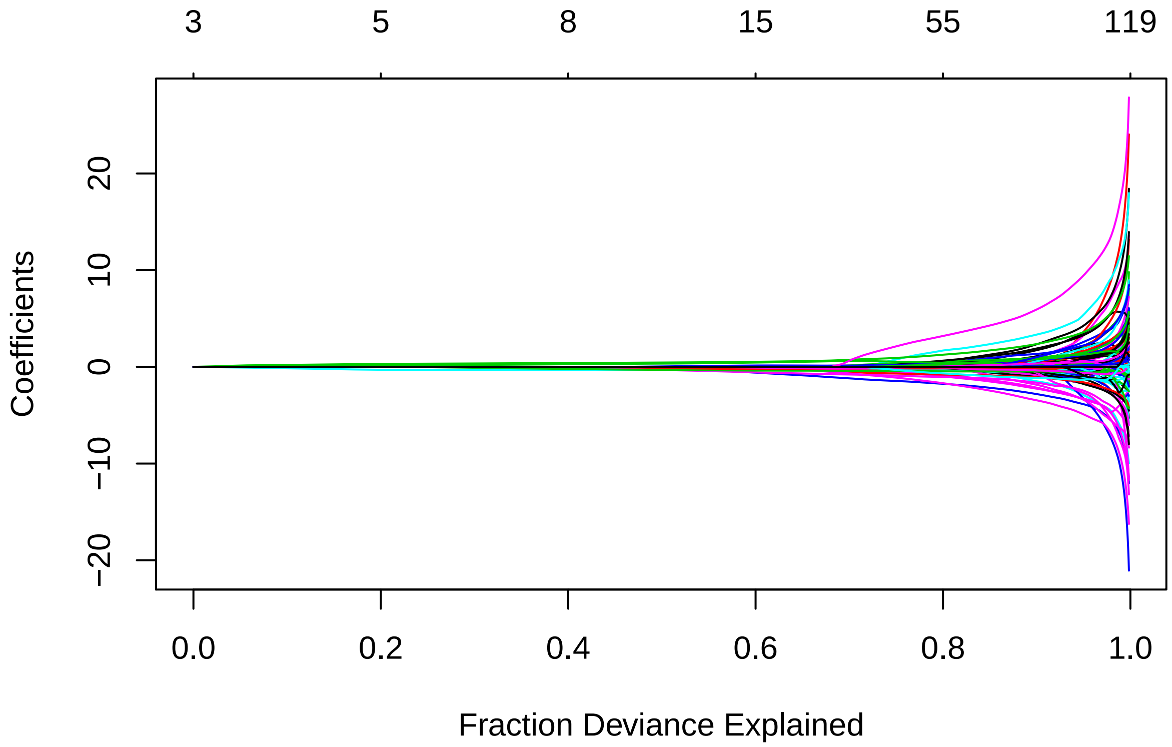
```
##                                     Classifier.test
## prediction.ridge.1se.log.classifier Low High
##                                   0 364   40
##                                   1  25  301
```

Lasso

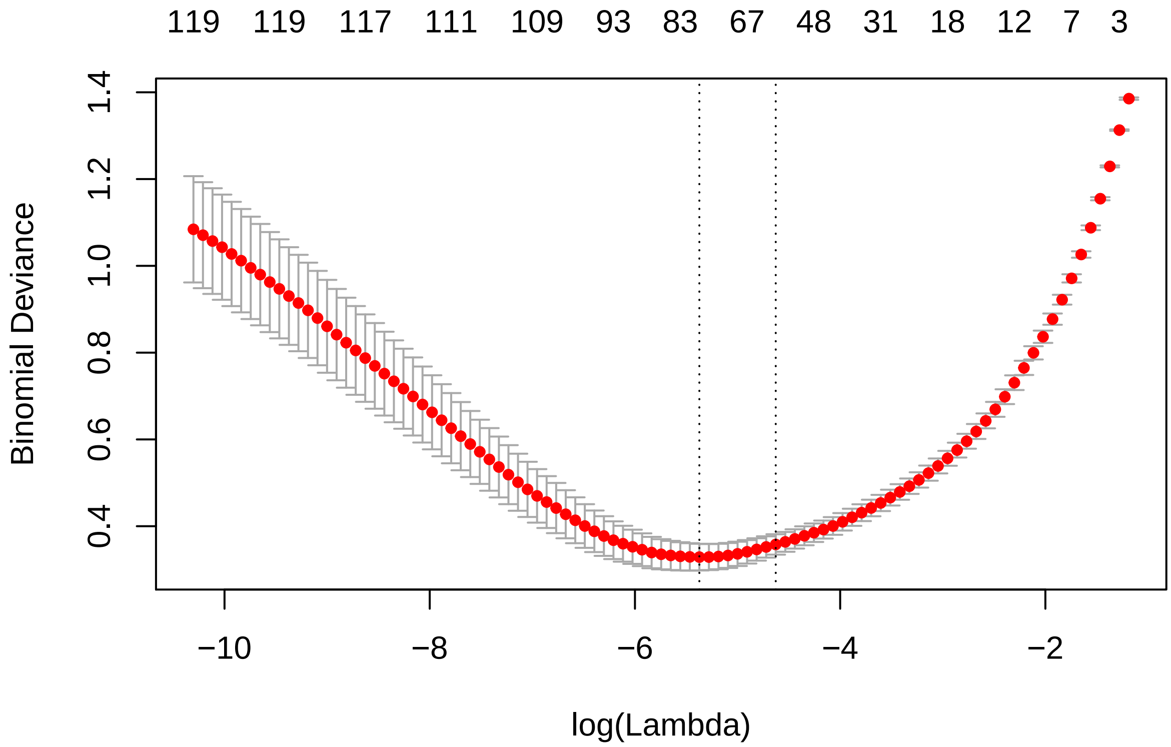
```
fit.lasso <- glmnet(x.train.classifier,y.train.classifier, family="binomial")
plot(fit.lasso, xvar='lambda', lanel=TRUE)
```



```
plot(fit.lasso, xvar='dev', lanel=TRUE)
```



```
cv.lasso <- cv.glmnet(x.train.classifier,y.train.classifier ,family="binomial")
plot(cv.lasso)
```



Lasso-Minimum Lambda

```
lasso.min.lambda=cv.lasso$lambda.min
fit.lasso.min <- glmnet(x.train.classifier,y.train.classifier, family="binomial", lambda=lasso.min.lamb
prediction.lasso.min.log <- predict(fit.lasso.min, x.test.classifier)
prediction.lasso.min.log.classifier <- (ifelse(prediction.lasso.min.log >0.5,1,0))
table(prediction.lasso.min.log.classifier, Classifier.test)
```

```
##                                     Classifier.test
## prediction.lasso.min.log.classifier Low High
##                                   0 354   33
##                                   1  35  308
```

Lasso-1se

```
lasso.1se.lambda=cv.lasso$lambda.1se
fit.lasso.1se <- glmnet(x.train.classifier,y.train.classifier,alpha=0, family="binomial", lambda=lasso.
prediction.lasso.1se.log <- predict(fit.lasso.1se, x.test.classifier)
prediction.lasso.1se.log.classifier <- ifelse(prediction.lasso.1se.log >0.5,1,0)
table(prediction.lasso.1se.log.classifier, Classifier.test)
```

```
##                                     Classifier.test
## prediction.lasso.1se.log.classifier Low High
##                                   0 357   41
##                                   1  32  300
```

Classification Tree

```
tree.train_df <- tree(Classifier~.-SalePrice, train.train_df)
```

```
tree.pred <- predict(tree.train_df, test.train_df, type="class")
length(tree.pred)
```

```
## [1] 730
```

```
length(Classifier.test)
```

```
## [1] 730
```

```
table(tree.pred, Classifier.test)
```

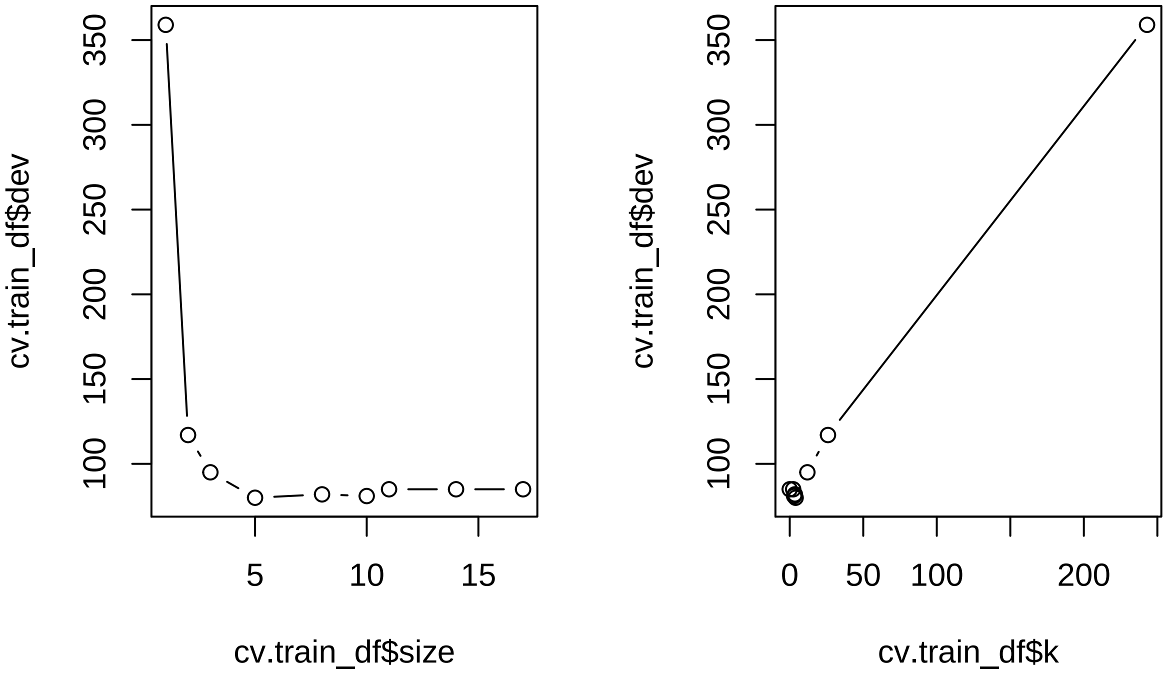
```
##          Classifier.test
## tree.pred Low High
##      Low  335   35
##      High  54  306
```

```
mean(tree.pred!=Classifier.test)
```

```
## [1] 0.1219178
```

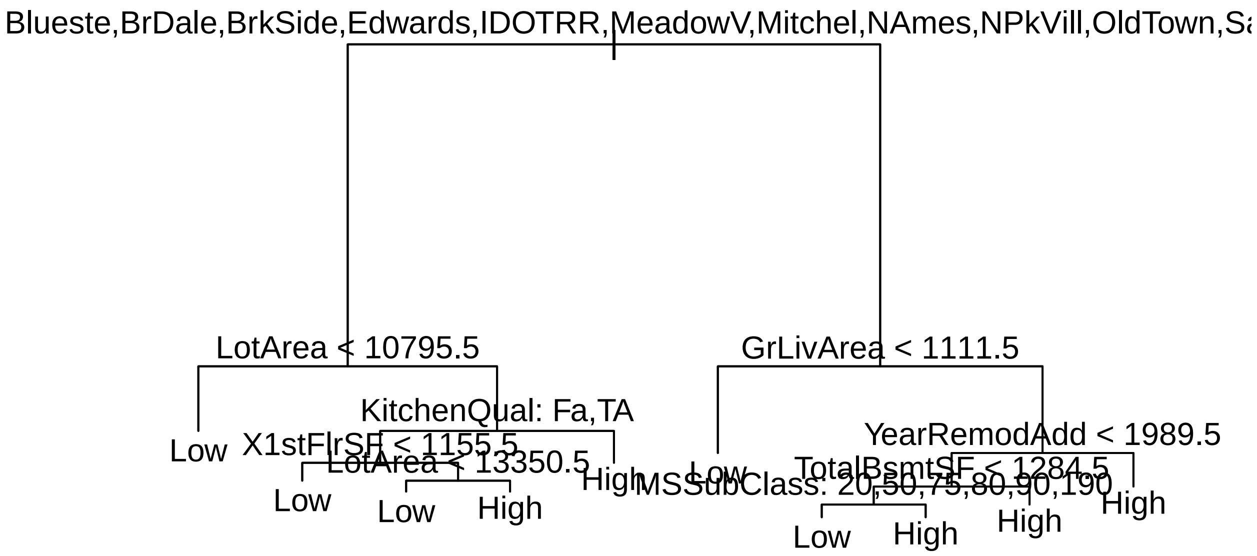
Prune the tree

```
cv.train_df <- cv.tree(tree.train_df, FUN= prune.misclass)
par(mfrow=c(1,2))
plot(cv.train_df$size, cv.train_df$dev, type="b")
plot(cv.train_df$k, cv.train_df$dev, type="b")
```



The optimal number of terminal node is 9

```
par(mfrow=c(1,1))
prune.train_df <-prune.misclass(tree.train_df, best=9)
plot(prune.train_df)
text(prune.train_df, pretty=0)
```



Compute the test error rate using the pruned tree

```
tree.pred <-predict(prune.train_df, test.train_df, type="class")
table(tree.pred,Classifier.test)
```

```
##          Classifier.test
## tree.pred Low High
##      Low  339   35
##      High  50  306
```

```
mean(tree.pred!=Classifier.test)
```

```
## [1] 0.1164384
```

Random Forest

Bagging: m=p

```
bag.train_df <- randomForest(Classifier~. -SalePrice, data=train.train_df, mtry=79, importance=TRUE)
bag.train_df
```

```
##
## Call:
##  randomForest(formula = Classifier ~ . - SalePrice, data = train.train_df,      mtry = 79, importanc
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 79
##
```

```
##         OOB estimate of  error rate: 7.81%
## Confusion matrix:
##      Low High class.error
## Low  343   28  0.07547170
## High  29  330  0.08077994
```

Predict

```
bag.classifier <- predict(bag.train_df, newdata = test.train_df)
table(predict=bag.classifier, truth=Classifier.test)
```

```
##        truth
## predict Low High
##    Low  351   34
##    High  38  307
```

```
mean(bag.classifier!=Classifier.test)
```

```
## [1] 0.09863014
```

Random Forest: m=p/3

```
rf.train_df <- randomForest(Classifier~. -SalePrice, data=train.train_df, mtry=26, importance=TRUE)
rf.train_df
```

```
##
## Call:
##  randomForest(formula = Classifier ~ . - SalePrice, data = train.train_df,      mtry = 26, importanc
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 26
##
##         OOB estimate of  error rate: 7.53%
## Confusion matrix:
##      Low High class.error
## Low  346   25  0.06738544
## High  30  329  0.08356546
```

Predict

```
rf.classifier <- predict(rf.train_df, newdata = test.train_df)
table(predict=rf.classifier, truth=Classifier.test)
mean(rf.classifier!=Classifier.test)
```

Support Vector Machine

Basic SVM Model

```
svmfit <-svm(Classifier~.-SalePrice, data=train.train_df, kernel="linear", cost=10, scale=FALSE)
```

Tuning SVM Model

```
tune.out <-tune(svm, Classifier~. -SalePrice , data=train.train_df, kernel="linear", ranges=list(cost=c
summary(tune.out)
```

We see that cost=0.01 results in the lowest cross validation error rate

tune() function stores the best model obtained, which can be assessed as follows

```
bestmod <-tune.out$best.model
summary(bestmod)

svm_pred <-predict(bestmod, test.train_df)
table(predict=svm_pred, truth=Classifier.test)
mean(svm_pred!=Classifier.test)
```

Boosting

```
train.train_df$Classifier2 <- ifelse(train.train_df$Classifier=="High",1,0)
boost.train=gbm(Classifier2~. -SalePrice -Classifier, data=train.train_df, distribution="bernoulli", n.
```

Predicting

```
boost.pred=predict(boost.train, newdata=test.train_df, n.trees=5000)
boost.classifier=ifelse(boost.pred >0.5,1,0)
Classifier2.test <- ifelse(Classifier.test=="High",1,0)
table(predict=boost.classifier, truth=Classifier2.test)
mean(boost.classifier!=Classifier2.test)
```

Estimation of graphical models using lasso-related approach

group 9

4/12/2017

Packages

```
library(MASS)
library(glmnet)
library(glasso)
library(zoo)
```

Function to generate theta

We need to generate a pxp symmetric positive definite matrix B. We sample each elements in the upper triangular matrix from a bernoulli with $p = 0.1$. We sum B with its transpose and now we have a symmetric matrix with the diagonal elements equal to 0.
If a matrix is symmetrical and strictly diagonally dominant then it is positive definite. Hence we put the elements of the diagonal equals to the maximum sum of the row plus 1. To standardize the matrix we divide by its diagonal elements.

```
rtheta=function(p)
{
  B=matrix(0,p,p)
  B[upper.tri(B, diag = FALSE)]=rbinom((p^2-p)/2,1,0.1)/2
  B=B+t(B)
  k=max(apply(B,1,sum))+1
  diag(B)=rep(k,p)
  B=B/B[1,1]
  return(B)
}
```

Node-wise lasso approach

This function takes a matrix x and a vector lambda. x are the fitting data and lambda is a vector of penalization term. It returns a matrix which a vector of prediction for each lambda.
To estimate this matrix it use the package "glmnet" to implement the node-wise lasso regression.

```
nodewise.lasso=function(x,lambda)
{
  #par.glmnet is a 3 dimensional array
  #its indexes are the variable x[i]
  #the estimated values for beta[i,j]
  #the levels if lambda

  p=ncol(x)
  par.glmnet=array(0,dim=c(p,p,length(lambda)))

  #We regress each X[i] on the remaining variables
  #we save the coefficients beta[i,j]
  #in the 3-dimensional array par.glmnet

  for(i in 1:p)
  {
    fit=glmnet(x[,-i],x[,i],lambda = lambda,intercept = FALSE)
    for(j in 1:p)
    {
      if(j<i) par.glmnet[i,j,]=coef(fit)[j+1,]
      if(j>i) par.glmnet[i,j,]=coef(fit)[j,]
    }
  }

  #We estimate 1 if b[i,j]!=0
  par.glmnet[par.glmnet!=0]=1

  #We arrange the 3-dimensional array into a matrix
  #For each lambda we save the estimated edges in a column of pred
  pred=matrix(0,p*(p-1)/2,length(lambda))

  #For each lambda we sum the pxp matrix
  #which contains 1 if b[i,j]!=0 with its transpose.
  #The new matrix cointains
  #0 if b[i,j]=b[j,i]=0
  #1 if b[i,j]=0 or b[j,i]=0 (not both of them)
  #2 if b[i,j]!=0 and b[j,i]!=0
  for(i in 1:length(lambda))
  {
    mat=par.glmnet[,,(length(lambda)-i+1)]+t(par.glmnet[,,(length(lambda)-i+1)])
    pred[,i]=mat[upper.tri(mat)]
  }
  return(pred)
}
```

Graphical LASSO approach

This function takes a matrix x and a vector lambda. x are the fitting data and lambda is a vector of penalization term. It use the function “glassopath” from the package “glasso”. This function allow to implement the graphical lasso regression for a vector of penalization term. It’s output it is a matrix containing a $p(p-1)/2$ vector of prediction for each lambda.

```
graphical.lasso=function(x,lambda)
{
  p=ncol(x)
  var.x=var(x)
  fit.glasso=glassopath(var.x,lambda,penalize.diagonal=FALSE,trace=0)

  #For each lambda we save the estimation for the matrix theta in the column of pred
  pred=matrix(0,p*(p-1)/2,length(lambda))
  for (i in 1:length(lambda))
  {
    fit=fit.glasso$wi[,,i]
    pred[,i]=as.vector(fit[upper.tri(fit)])
  }

  #we predict 1 if the element is != 0
  pred[pred!=0]=1
  return(pred)
}
```

TPR and FPR calculation

This function takes as input the estimated vertices, the real vertices and the vector of lambda and it evaluate the true and false positive rate.
Since our predictions are vector of 0 and 1, we can use some trick to evaluate the TPR and the FPR. In particular $TPR=#{E'=1 && E=1}/#{E=1}. We have that {E'=1 && E=1} if and only if {E'E=1}. Hence, to evaluate the TPR it is possible to count the 1 in the vector E'E and divide it by the number of 1 in E. The FPR it has been calculated in a similar way.
The multiplication between vectors are elements by elements (according to the R definition).

```
error.rate=function(pred,edges,lambda)
{
  tpr=rep(0,length(lambda))
  fpr=rep(0,length(lambda))
  tpr=apply(pred*edges,2,sum)/sum(edges)
  fpr=apply(pred*(1-edges),2,sum)/sum(1-edges)
  return(cbind(tpr,fpr))
}
```

AUROC evaluation

This function evaluate the AUROC with the trapezium approximation.

```
aur=function(tpr,fpr)
{
  o=order(fpr)
  a=sum(rollmean(tpr[o],2)*diff(fpr[o]))
  return(a)
}
```

Main function

This function recall all the previous function: it generate the set of vertices, it predicts the vertices with the 3 estimation methods, it plots the miss-classification rate, it evaluates TPR, FPR, ROC and AUROC and it returns them as a list.

```
one.time.run=function(p,n,lambda)
{
  theta=rtheta(p)
  #sigma is the inverse of theta
  sigma=solve(theta,sparse=TRUE)

  #Edges is the true response variable
  edges=as.vector(theta[upper.tri(sigma)])
  edges[edges!=0]=1

  #We generate a n random sample from a multivariate gaussian distribution with mean 0 and variance Sig
  x=mvrnorm(n,rep(0,p),sigma)

  #GLMNET package for Node-wise LASSO approach
  #Determining prediction for method 1 and 2
  #Method 1 predicts a vertex if pred=1 or pred=2
  #Method 2 predicts a vertex if pred=2

  pred=nodewise.lasso(x,lambda)
  #Method 1 predicts and edge if pred is equal to 1 or 2
  pred.1=matrix(0,p*(p-1)/2,length(lambda))
  pred.1[pred!=0]=1
  #Method 2 predicts and edge if pred is equal to 2
  pred.2=matrix(0,p*(p-1)/2,length(lambda))
  pred.2[pred==2]=1

  #GLASSO package for Graphical Lasso approach
  pred.3=graphical.lasso(x,lambda)

  #OUTPUT - Mis-classification rate
  #Graph
  if(p==100 && n==500)
  {
  par(mfrow=c(1,3))
  plot(lambda,2*apply(abs(pred.1-edges)/(p*(p-1)),2,sum),type="l",ylim=c(0,1),ylab = "Method 1",xlab="La
  mtext("Mis-classification rate", side = 3, line = 2, font=2)
  legend("topright",legend=c(p,n))
  plot(lambda,2*apply(abs(pred.2-edges)/(p*(p-1)),2,sum),type="l",ylim=c(0,1),ylab = "Method 2",xlab="La
  plot(lambda,2*apply(abs(pred.3-edges)/(p*(p-1)),2,sum),type="l",ylim=c(0,1),ylab = "Method 3",xlab="La
  par(mfrow=c(1,1))
  }

  #ROC and AUROC
  #positive.rate is a matrix cointaining the vector of true and false positive rate for each lambda

  positive.rate.1=error.rate(pred.1,edges,lambda)
  positive.rate.2=error.rate(pred.2,edges,lambda)
  positive.rate.3=error.rate(pred.3,edges,lambda)

  auroc.1=aur(positive.rate.1[,1],positive.rate.1[,2])
  auroc.2=aur(positive.rate.2[,1],positive.rate.2[,2])
  auroc.3=aur(positive.rate.3[,1],positive.rate.3[,2])
  li=list(list(pred.1,pred.2,pred.3),list(positive.rate.1,positive.rate.2,positive.rate.3),list(auroc.1
  return(li)
}
```

Main Function 2

This function is the same as the latter, but it doesn't plot any graph and it returns only the AUROC values.

```
no.graph.run=function(p,n,lambda)
{
  theta=rtheta(p)
  sigma=solve(theta,sparse=TRUE)

  edges=as.vector(theta[upper.tri(sigma)])
  edges[edges!=0]=1

  x=mvrnorm(n,rep(0,p),sigma)

  pred=nodewise.lasso(x,lambda)
  pred.1=matrix(0,p*(p-1)/2,length(lambda))
  pred.1[pred!=0]=1
  pred.2=matrix(0,p*(p-1)/2,length(lambda))
  pred.2[pred==2]=1

  pred.3=graphical.lasso(x,lambda)

  positive.rate.1=error.rate(pred.1,edges,lambda)
  positive.rate.2=error.rate(pred.2,edges,lambda)
  positive.rate.3=error.rate(pred.3,edges,lambda)

  auroc.1=aur(positive.rate.1[,1],positive.rate.1[,2])
  auroc.2=aur(positive.rate.2[,1],positive.rate.2[,2])
  auroc.3=aur(positive.rate.3[,1],positive.rate.3[,2])

  return(cbind(auroc.1,auroc.2,auroc.3))
}
```

Main Program

Parameters settings

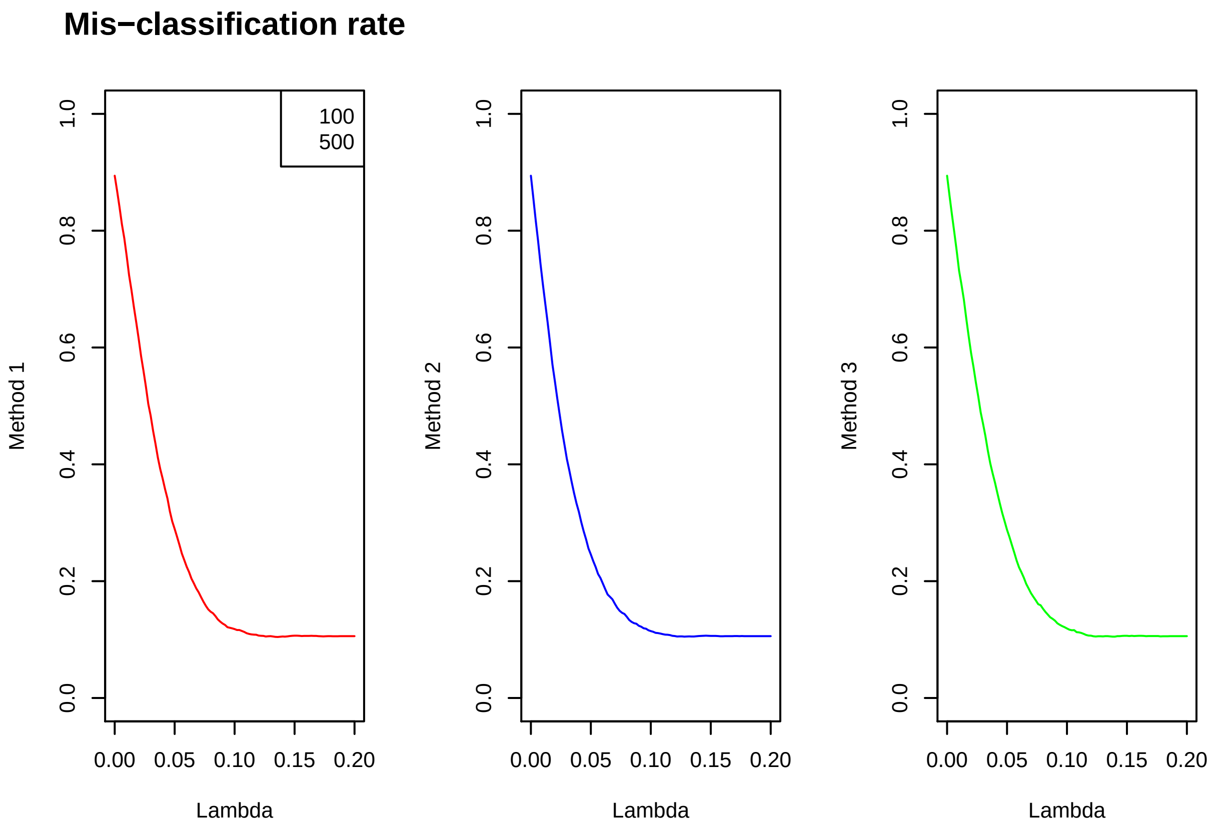
```
#The seed is the delivery date
set.seed(06122017)
#We define the vector lambda for the penality term
lambda=seq(0.0,0.2,by=0.002)
```

Estimation with different p and n

Result is a list. The following instruction explain how to recall the informations:

$result.p.n[[i]][[j]]$
j=1,2,3 is the method used
i=1 => prediction
i=2 => true positive rate and false negative rate i=3 => auroc

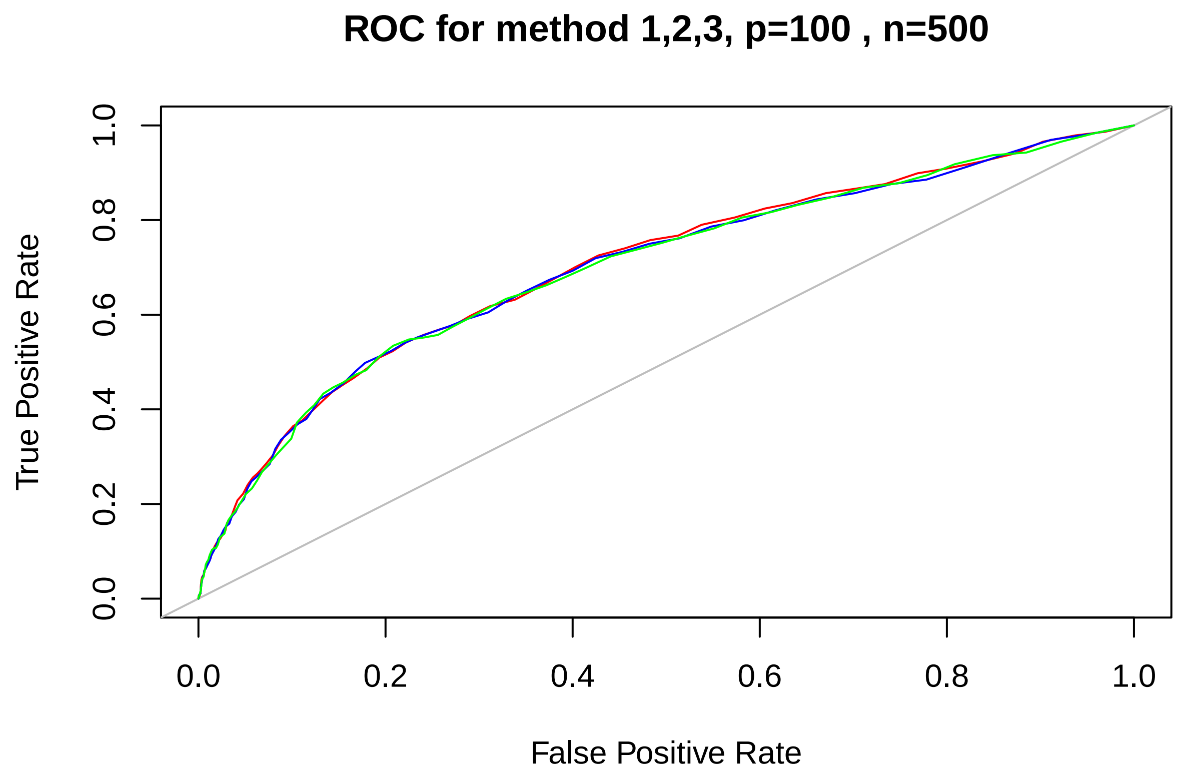
```
result.50.500=one.time.run(50,500,lambda)
result.100.200=one.time.run(100,200,lambda)
result.100.500=one.time.run(100,500,lambda)
```



```
result.100.1000=one.time.run(100,1000,lambda)
result.150.500=one.time.run(150,500,lambda)
```

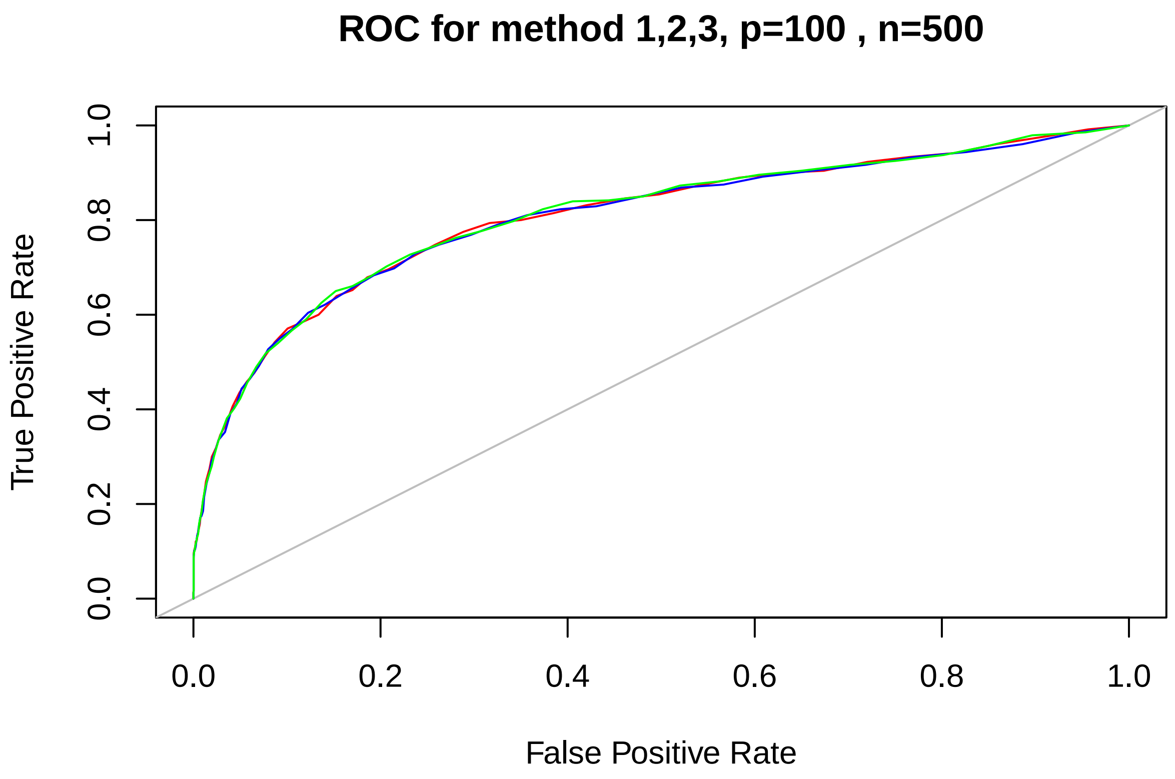
ROC curve across methods, p=100, n=500

```
plot(x = c(0, 1),y = c(0, 1),type = "n",main = "ROC for method 1,2,3, p=100 , n=500",xlab = "False Posi
abline(0,1,col="grey")
lines(result.100.500[[2]][[1]][,2],result.100.500[[2]][[1]][,1],col="red")
lines(result.100.500[[2]][[2]][,2],result.100.500[[2]][[2]][,1],col="blue")
lines(result.100.500[[2]][[3]][,2],result.100.500[[2]][[3]][,1],col="green")
```



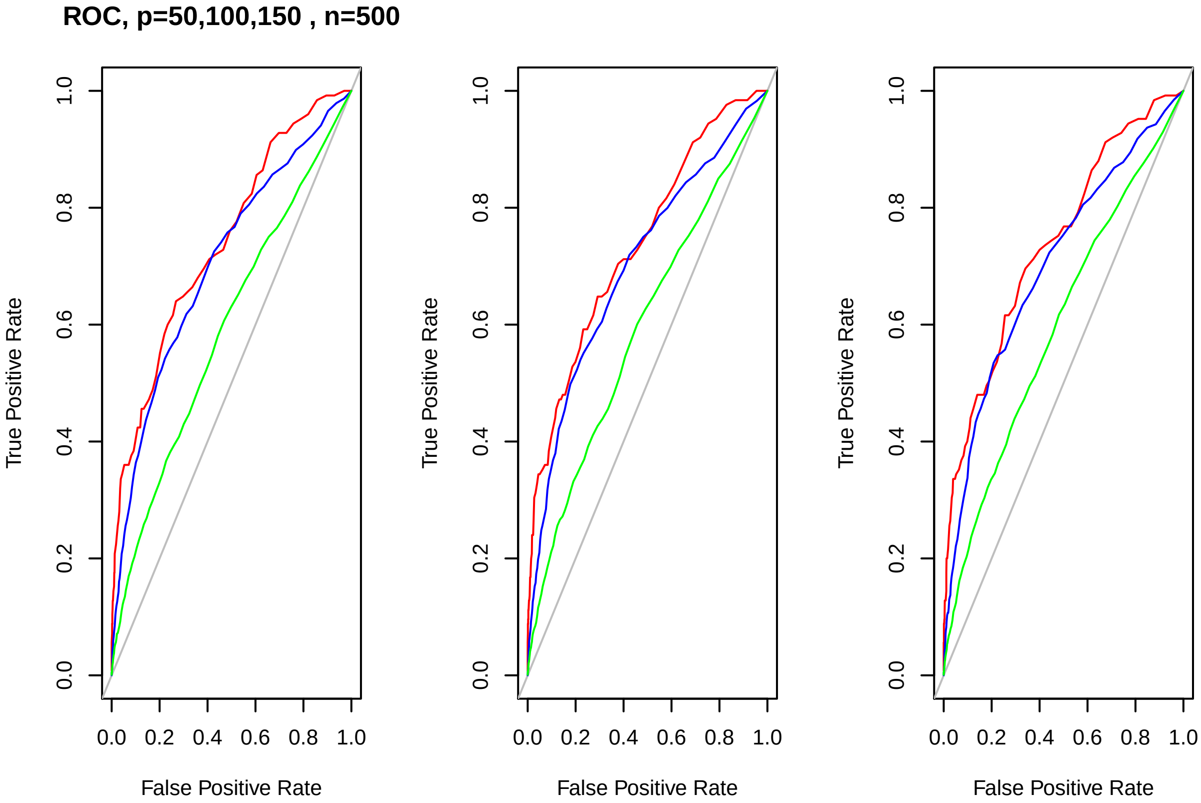
ROC curve across methods, p=100, n=1000

```
plot(x = c(0, 1),y = c(0, 1),type = "n",main = "ROC for method 1,2,3, p=100 , n=500",xlab = "False Posi
abline(0,1,col="grey")
lines(result.100.1000[[2]][[1]][,2],result.100.1000[[2]][[1]][,1],col="red")
lines(result.100.1000[[2]][[2]][,2],result.100.1000[[2]][[2]][,1],col="blue")
lines(result.100.1000[[2]][[3]][,2],result.100.1000[[2]][[3]][,1],col="green")
```



ROC curves, same n, p changes

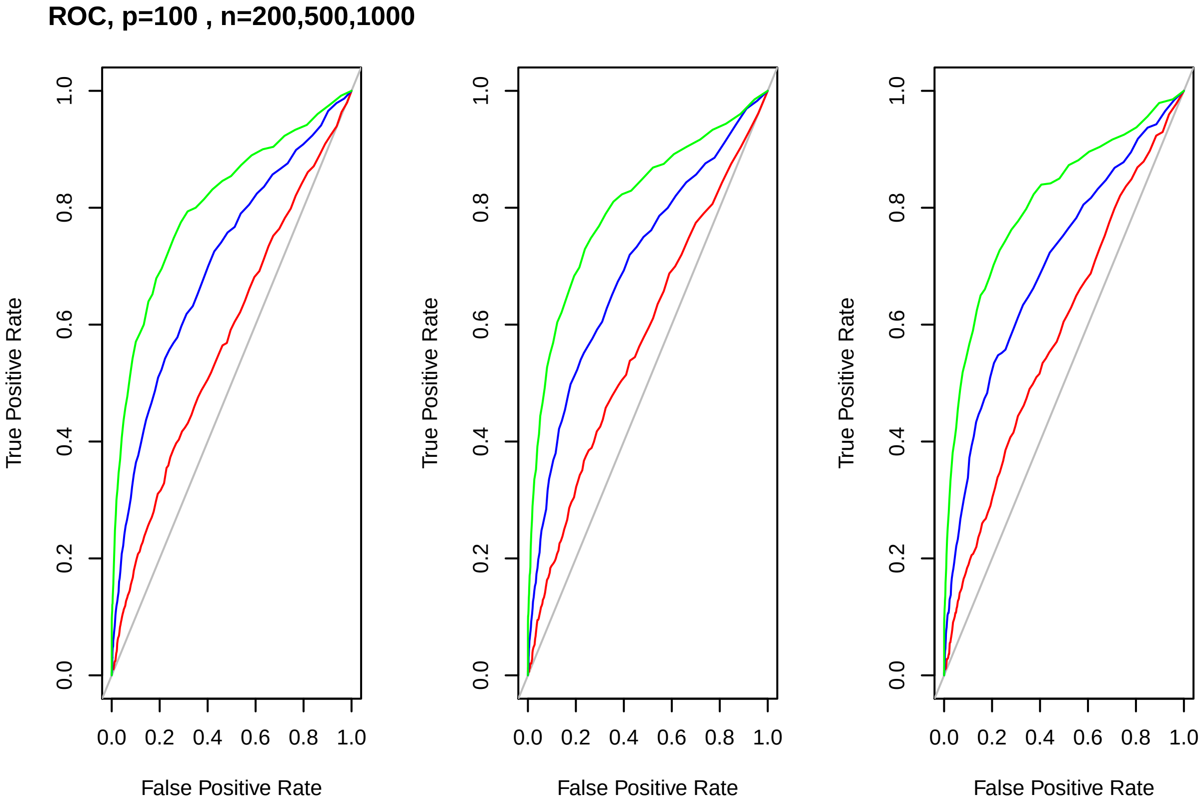
```
par(mfrow=c(1,3))
plot(x = c(0, 1),y = c(0, 1),type = "n",main = "ROC, p=50,100,150 , n=500",xlab = "False Positive Rate"
abline(0,1,col="grey")
lines(result.50.500[[2]][[1]][,2],result.50.500[[2]][[1]][,1],col="red")
lines(result.100.500[[2]][[1]][,2],result.100.500[[2]][[1]][,1],col="blue")
lines(result.150.500[[2]][[1]][,2],result.150.500[[2]][[1]][,1],col="green")
plot(x = c(0, 1),y = c(0, 1),type = "n",xlab = "False Positive Rate", ylab = "True Positive Rate")
abline(0,1,col="grey")
lines(result.50.500[[2]][[2]][,2],result.50.500[[2]][[2]][,1],col="red")
lines(result.100.500[[2]][[2]][,2],result.100.500[[2]][[2]][,1],col="blue")
lines(result.150.500[[2]][[2]][,2],result.150.500[[2]][[2]][,1],col="green")
plot(x = c(0, 1),y = c(0, 1),type = "n", xlab = "False Positive Rate", ylab = "True Positive Rate")
abline(0,1,col="grey")
lines(result.50.500[[2]][[3]][,2],result.50.500[[2]][[3]][,1],col="red")
lines(result.100.500[[2]][[3]][,2],result.100.500[[2]][[3]][,1],col="blue")
lines(result.150.500[[2]][[3]][,2],result.150.500[[2]][[3]][,1],col="green")
```



```
par(mfrow=c(1,1))
```

ROC curves, same p, n changes

```
par(mfrow=c(1,3))
plot(x = c(0, 1),y = c(0, 1),type = "n",main = "ROC, p=100 , n=200,500,1000",xlab = "False Positive Rate
abline(0,1,col="grey")
lines(result.100.200[[2]][[1]][,2],result.100.200[[2]][[1]][,1],col="red")
lines(result.100.500[[2]][[1]][,2],result.100.500[[2]][[1]][,1],col="blue")
lines(result.100.1000[[2]][[1]][,2],result.100.1000[[2]][[1]][,1],col="green")
plot(x = c(0, 1),y = c(0, 1),type = "n",xlab = "False Positive Rate", ylab = "True Positive Rate")
abline(0,1,col="grey")
lines(result.100.200[[2]][[2]][,2],result.100.200[[2]][[2]][,1],col="red")
lines(result.100.500[[2]][[2]][,2],result.100.500[[2]][[2]][,1],col="blue")
lines(result.100.1000[[2]][[2]][,2],result.100.1000[[2]][[2]][,1],col="green")
plot(x = c(0, 1),y = c(0, 1),type = "n",xlab = "False Positive Rate", ylab = "True Positive Rate")
abline(0,1,col="grey")
lines(result.100.200[[2]][[3]][,2],result.100.200[[2]][[3]][,1],col="red")
lines(result.100.500[[2]][[3]][,2],result.100.500[[2]][[3]][,1],col="blue")
lines(result.100.1000[[2]][[3]][,2],result.100.1000[[2]][[3]][,1],col="green")
```



```
par(mfrow=c(1,1))
```

50 times

We run the code fifty times to evaluate mean, variance and stardard deviation of the AUROC.

```
p=100
n=500
set.seed(06122017)
auroc=matrix(0,50,3)
for(t in 1:50)
{
  auroc[t,]=no.graph.run(p,n,lambda)
}
```

```
apply(auroc,2,mean)
```

```
## [1] 0.6512545 0.6500842 0.6533424
```

```
apply(auroc,2,sd)
```

```
## [1] 0.02899642 0.02859026 0.02875512
```

```
apply(auroc,2,var)
```

```
## [1] 0.0008407924 0.0008174031 0.0008268572
```