CHAPTER 8: SCALABILITY

LASSO AND PCA

I. Overview

Chapter 7 is about "**Scalability**". It is to enhance our capacity to deal with large-scale problems – strength to be scalable. LASSO and PCA will be introduced in this chapter. LASSO stands for the Least Absolute Shrinkage and Selection Operator, which is a main representative method for feature selection. PCA stands for the Principle Component Analysis, which is a main representative method for dimension reduction. Both methods can reduce the dimensionality of the dataset, but follow different styles. LASSO, as a feature selection method, focuses on deletion of irrelevant or redundant features in the dataset. PCA, as a dimension reduction method, keeps all the features but combine them into a smaller number of aggregated new features.

II. LASSO

II.1 Rationale and Formulation

LASSO was invented in 1996[1] that was used to sparsify the linear regression model and allowed the regression model to select significant predictors automatically. The formulation of LASSO is

$$\widehat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}}\{\|\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_1\},$$

where $\boldsymbol{y} \in \mathbb{R}^{N\times 1}$ is the measurement vector of the response, $\mathbf{X} \in \mathbb{R}^{N\times p}$ is the data matrix of the N measurement vectors of the p predictors, $\boldsymbol{\beta} \in \mathbb{R}^{p\times 1}$ is the regression coefficient vector. Here, $\|\boldsymbol{\beta}\|_1 = \sum_{i=1}^{p}|\beta_i|$. Note that, here, the intercept (i.e., β_0) is not included, since we assume that the data is normalized (i.e., $\sum_{n=1}^{N} x_{nj}/N = 0$, $\sum_{n=1}^{N} x_{nj}^2/N = 1$ for $j = 1,2,\dots,p$ and $\sum_{n=1}^{N} y_n/N = 0$), and thus, the intercept is not needed.

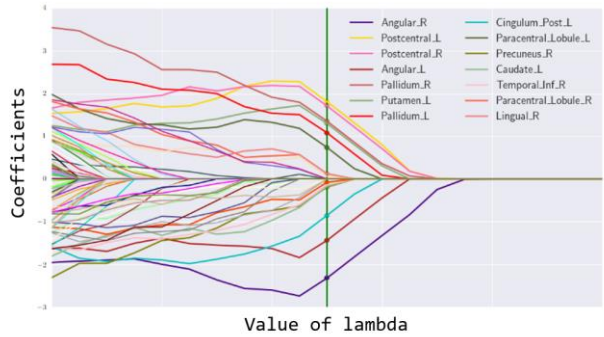

Figure 8.1: Path solution trajectory of the coefficients of LASSO, identifying brain regions that show longitudinal declines that can separate early-stage Alzheimer's patients from normal elderly.

[1] Tibshirani, R. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society (Series B)*, 1996.

It could be seen that LASSO embodies two major components in its formulation. The 1st term is the least squares loss function from linear regression, that is used to measure the goodness-of-fit of the model. The 2nd term is the sum of absolute values of the elements in $\boldsymbol{\beta}$, representing the model complexity, i.e., the smaller the $\|\boldsymbol{\beta}\|_1$, more zeros in $\boldsymbol{\beta}$, leading to a simpler model. The parameter, λ, is called the penalty parameter that is specified by user of LASSO. In other words, LASSO suggests the best model by an optimal balance between model fit and model complexity, and this balance could be flexibly tuned by tuning the parameter λ. Furthermore, as shown in Figure 8.1, LASSO can generate the path solution trajectory that visualizes the solutions of $\boldsymbol{\beta}$ for a continuum of values of λ, giving us a global sense of the relationships between variables. Also, model selection criteria such as Akaike Information Criteria (AIC) or cross-validation can be used to identify the best λ.

II.2 Theory and Method

Why LASSO uses the L_1 norm: The popularity of LASSO and its enormous impact on statistical/machine learning research in the last decade needs no exaggeration. Some researchers in optimization and operations research often found puzzling is why all of sudden LASSO was invented and gave birth to the area of sparse learning. To answer this question, LASSO is often compared with another similar model, called **Ridge regression**[1] that has been developed almost half a century ago.

The formulation of Ridge regression is

$$\widehat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}}\{\|\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2\},$$

where $\|\boldsymbol{\beta}\|_2 = \sum_{i=1}^{p}|\beta_i|^2$ is called the L_2 norm.

[1] Hoerl, A.E. and Kennard, R.W. Ridge regression: biased estimation for nonorthogonal problems. *Technometrics*, 1970.

At the first glance, it seems that the Ridge regression bears the same spirit of LASSO – they both penalize the magnitudes of the regression parameters. However, it has been noticed that, in the Ridge regression model, the regression parameters in $\boldsymbol{\beta}$ will not achieve exactly zero. Even if you impose a very large λ, many elements in $\boldsymbol{\beta}$ may be close to zero with a very tiny numerical magnitude, but not zero. Although the numerical magnitudes of these elements are close to zero, and thus, seems to be insignificant, they are still in the estimation system and generate impacts on the estimation of other regression parameters. Thus, it is often reported that when you run Ridge regression and LASSO regression, you may not observe the same set of predictors that are selected by both methods. Ridge regression is more often used as a stabilization strategy to handle the multicollinearity issue or any other issues that result in numerical instability in parameter estimation, while LASSO is used as a variable selection strategy.

Shooting algorithm to solve the optimization problem of LASSO: We could denote the objective function of LASSO as

$$L(\boldsymbol{\beta}) = \|\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_1.$$

To see how the LASSO can be iteratively solved, let's first consider a simple case where there is only one predictor. Then, the objective function becomes

$$L(\beta) = \|\boldsymbol{y} - \mathbf{X}\beta\|_2^2 + \lambda|\beta|.$$

To find the optimal solution, we can solve the equation as

$$\frac{\partial L(\beta)}{\partial \beta} = 0.$$

The complication is the L1-norm term, $|\beta|$, which has no gradient when $\beta = 0$. Thus, we can discuss different scenarios and identify the solutions.

- If $\beta > 0$, then $\frac{\partial L(\beta)}{\partial \beta} = 2\beta - 2\mathbf{X}^T\boldsymbol{y} + \lambda$. Thus, $\frac{\partial L(\beta)}{\partial \beta} = 0$ will lead to the solution that $\beta = \frac{(2\mathbf{X}^T\boldsymbol{y} - \lambda)}{2}$. But if $2\mathbf{X}^T\boldsymbol{y} - \lambda < 0$, this will result in a contradiction, and thereby, $\beta = 0$.
- If $\beta < 0$, then $\frac{\partial L(\beta)}{\partial \beta} = 2\beta - 2\mathbf{X}^T\boldsymbol{y} - \lambda$. Similarly as above, we can conclude that $\beta = \frac{(2\mathbf{X}^T\boldsymbol{y} + \lambda)}{2}$. But if $2\mathbf{X}^T\boldsymbol{y} + \lambda > 0$, this will result in a contradiction, and thereby, $\beta = 0$.
- If $\beta = 0$, then we have had the solution and no longer need the calculate the gradient.

In summary, we can derive the solution of β as

$$\hat{\beta} = \begin{cases} \frac{(2\mathbf{X}^T\boldsymbol{y} - \lambda)}{2}, & if 2\mathbf{X}^T\boldsymbol{y} - \lambda > 0 \\ \frac{(2\mathbf{X}^T\boldsymbol{y} + \lambda)}{2}, & if 2\mathbf{X}^T\boldsymbol{y} + \lambda < 0 \\ 0, & if\ \lambda \geq |2\mathbf{X}^T\boldsymbol{y}| \end{cases}.$$

Now we are ready to generalize this practice to general case with more predictors.

The spirit is to keep as much the easiness of solving for one predictor as we derived above as possible. Thus, revealing the resemblance of the general problem with our one-predictor special problem is important. Particularly, we decide to follow an iterative structure that updates each β_j at a time when fixing all the other parameters as their latest values. Thus, suppose that we are now at the tth iteration and we are trying to optimize for β_j, we can rewrite the general optimization problem's objective function as a function of β_j

$$L(\beta_j) = \left\| \boldsymbol{y} - \sum_{k \neq j} \mathbf{X}_{(:,k)} \beta_k^{(t-1)} - \mathbf{X}_{(:,j)} \beta_j \right\|_2^2 + \lambda \sum_{k \neq j} \left| \beta_k^{(t-1)} \right| + \lambda |\beta_j|.$$

Here, $\beta_k^{(t)}$ is the value of β_k in the tth iteration. The objective function above can be simplified as

$$L(\beta_j) = \left\|\boldsymbol{y} - \mathbf{X}_{(:,j)}\beta_j\right\|_2^2 + \lambda|\beta_j|,$$

which just resembles the structure as the one-predictor special case we discussed. Thus, we can readily derive that

$$\hat{\beta}_j^{(t)} = \begin{cases} q_j - \lambda/2\,, & if\, q_j - \lambda/2 > 0 \\ q_j + \lambda/2\,, & if\, q_j + \lambda/2 < 0 \\ 0, & if\ \lambda \geq |2q_j| \end{cases},$$

where $q_j = \mathbf{X}_{(:,j)}^T \left(\boldsymbol{y} - \sum_{k\neq j} \mathbf{X}_{(:,k)}\beta_k^{(t-1)}\right)$.

An Example to implement the Shooting algorithm: Here let's consider one exemplary data as shown in below.

Table 8.1: A dataset example for LASSO

X_1	X_2	Y
-0.707	0	-0.77
0	0.707	-0.33
0.707	-0.707	0.62

The dataset of Y is actually randomly sampled from the true model,

$$Y = 0.8X_1 + \varepsilon, \text{ where } \varepsilon \sim N(0,0.5).$$

Thus, it can be seen that the variable X_2 is irrelevant.

Now let's implement the Shooting algorithm for LASSO on this data. The objective function of LASSO on this case is

$$\sum_{n=1}^{N}\left[y_n - \left(\beta_1 x_{n,1} + \beta_2 x_{n,2}\right)\right]^2 + \lambda(|\beta_1| + |\beta_2|).$$

Note that, here, for simplicity, we don't need to include the offset parameter β_0 in the model as the predictors are standardized with mean as zero.

Suppose that we choose $\lambda = 0.88$. First, we initiate the regression parameters as $\hat{\beta}_1^{(0)} = 0$ and $\hat{\beta}_1^{(0)} = 0$.

In the first iteration, we aim to update $\hat{\boldsymbol{\beta}}_1$. We can obtain that

$$\boldsymbol{y} - \mathbf{X}_{(:,2)}\hat{\beta}_2^{(0)} = \begin{bmatrix} -0.71 \\ -1.037 \\ 1.327 \end{bmatrix}.$$

Thus,

$$q_1 = \mathbf{X}_{(:,1)}^T \left(\boldsymbol{y} - \mathbf{X}_{(:,2)}\hat{\beta}_2^{(0)} \right) = 1.44.$$

As

$$q_1 - \lambda/2 = 1 > 0,$$

we know that

$$\hat{\beta}_1^{(1)} = q_1 - \lambda/2 = 1.$$

Similarly, we can update $\hat{\boldsymbol{\beta}}_2$. We can obtain that

$$\boldsymbol{y} - \mathbf{X}_{(:,1)}\hat{\beta}_1^{(0)} = \begin{bmatrix} -1.477 \\ -0.33 \\ -0.087 \end{bmatrix}.$$

Thus,

$$q_2 = \mathbf{X}_{(:,1)}^T \left(\boldsymbol{y} - \mathbf{X}_{(:,1)}\hat{\beta}_1^{(0)} \right) = -0.178.$$

As

$$\lambda \geq |2q_2|,$$

we know that

$$\hat{\beta}_2^{(1)} = 0.$$

Thus, on this simple example, with only one iteration, the LASSO method can identify the irrelevant variable and delete it from the model.

II.3 R Lab

In what follows, we apply LASSO on an extended AD dataset that has 329 variables. It includes 313 variables that are derived from the MRI images of the subjects, corresponding to the grey matter volumes of 313 brain regions. We have known that many of these variables are correlated with each other as our prior knowledge. Also, depending on the outcome to predict, not all the brain regions are useful. Thus, this extended AD dataset provides a good example for us to showcase the use of LASSO and PCA.

First, let's load the data into the R workspace:

```
# Chapter 8 Dataset of Alzheimer's Disease Objective: prediction
of
# diagnosis filename
AD <- read.csv("AD_hd.csv", header = TRUE)
```

This time, let's formulate an interesting prediction question: can we use the MRI readings to predict the age of the subject. Using the following R code, we may get a sense of the relationship of the variables by drawing the scatterplots of variables that correlate with the outcome variable "AGE" most strongly according to the Pearson correlation.

```
# Supplement the model with some visualization of the statistical
 patterns
# Scatterplot matrix to visualize the relationship between outcom
e variable
# with continuous predictors
require(ggplot2)
# install.packages('GGally')
require(GGally)
# draw the scatterplots and also empirical shapes of the distribu
tions of
# the variables
tempRank <- sort(abs(cor(AD[, 5], AD[, 17:329])), decreasing = TR
UE, index.return = TRUE)
p <- ggpairs(AD[, c(5, 16 + tempRank$ix[1:8])], upper = list(cont
inuous = "points"),
    lower = list(continuous = "cor"))
print(p)
```

Then we can see that, the correlations between the MRI variables with `AGE` are significant, and the correlations among the MRI variables are also strong, which confirms with our prior knowledge and also indicates significant redundancy in the variables.

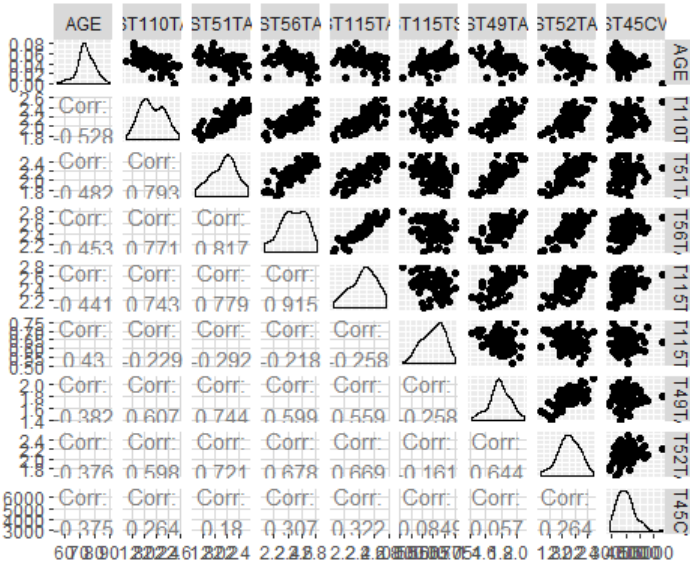

Figure 8.2: Scatterplots of some MRI variables

Now let's split the data into training and testing datasets.

```
AD[, 17:dim(AD)[2]] <- scale(AD[, 17:dim(AD)[2]])
# Use the glmnet R pacakge to build LASSO model split into traini
ng and test
# sets
AD$train <- ifelse(runif(nrow(AD)) < 0.8, 1, 0)
# separate training and test sets
trainset <- AD[AD$train == 1, -grep("train", names(AD))]
testset <- AD[AD$train == 0, -grep("train", names(AD))]
trainX <- as.matrix(trainset[, 17:dim(trainset)[2]])
testX <- as.matrix(testset[, 17:dim(testset)[2]])
trainY <- as.matrix(trainset[, 5])
testY <- as.matrix(testset[, 5])
```

The `glmnet` package can be used to implement the LASSO method:

```
# build model install.packages('glmnet')
require(glmnet)
fit = glmnet(trainX, trainY, nlambda = 100)
```

We can use the `plot()` function to see the path solution trajectory of the regression coefficients by LASSO for different values of lambda, as shown in Figure 8.3.

```
plot(fit, label = TRUE)
```

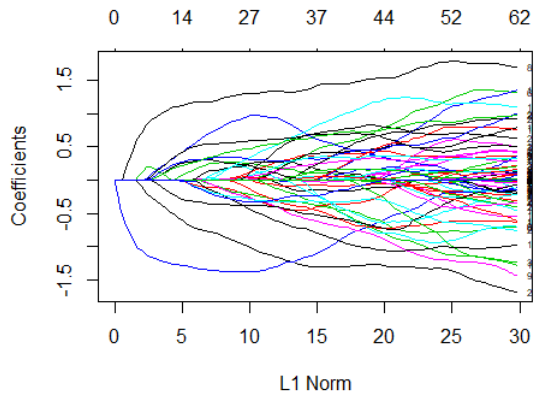

Figure 8.3: Path solution trajectory of the regression coefficients by LASSO

The numerical details of the trained LASSO regression models can also be seen by `print(fit)`. Here, we skip the output due to space limit. As we have seen, the glmnet trained not only one LASSO model, but actually 100 models by default. We could use `coef()` to query details of each of the models. For example, `coef(fit, s = 0.05)` queries the model that has `lambda = 0.05`.

We draw the histogram of the Pearson correlations of the selected variables in this model, which is shown in Figure 8.4.

```
# Check out the marginal correlations between the selected variab
les with
# the outcome
idx.var <- which(coef(fit, s = 0.05) != 0) - 1
tempData <- as.numeric(abs(cor(trainY, trainX[, idx.var])))
qplot(tempData, geom = "histogram")
```

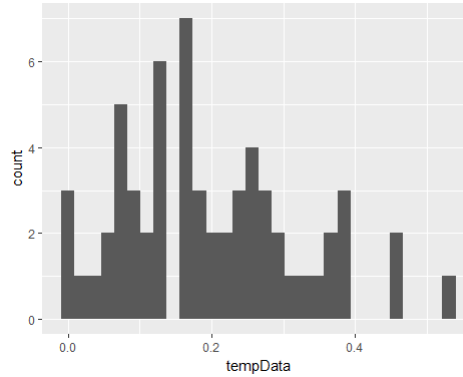

Figure 8.4: Histogram of the Pearson correlations of the selected variables with `AGE`

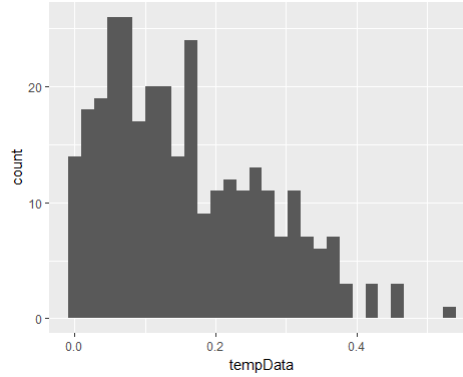

Figure 8.5: Histogram of the Pearson correlations of all variables with `AGE`

We can see that, the LASSO model is a multivariate method that selects variables not only based on their marginal correlations with the outcome, but also their syngeneic effects. Figure 8.5 shows the histogram of the Pearson correlations of all variables for a contrast.

```
# Compare with the overview of the correlations between variables
 with the
# outcome
tempData <- as.numeric(abs(cor(trainY, trainX)))
qplot(tempData, geom = "histogram")
```

The `predict()` function can be used to predict on the testing dataset using different models.

```
# Predict on the testing data
predict(fit, newx = testX, s = c(0.1, 0.2, 0.4))

##           1        2        3
## 3  81.24935 81.23730 78.97272
## 9  69.19876 69.64454 70.43511
## 17 69.89315 68.66627 67.34304
## 19 72.01627 70.19499 71.69000
## 29 73.75759 71.94729 71.05652
## 30 67.57181 67.22865 67.62802
## 31 69.72086 69.79635 71.41110
## 38 70.65147 72.04312 73.13276
## 39 84.74255 84.62814 84.28301
## 45 71.88253 71.29639 69.67876
## 48 74.83788 74.96710 72.61464
## 52 66.70134 69.83361 71.03180
## 53 72.42933 71.19085 72.60041
## 61 73.11982 73.33213 75.97945
## 62 75.87737 74.92556 75.84209
## 73 74.98227 75.70788 74.91021
```

On the other hand, to decide on the best model, glmnet implements 10-folder cross-validation procedure using `cv.glmnet()`.

```
# Use cross-validation to decide which model is best
cv.fit = cv.glmnet(trainX, trainY)
plot(cv.fit)
```

The result is shown in Figure 8.6, which reveals a certain optimal point on which we can decide the best model. It is also noticeable that the

optimality of the model is not very statistically significant, probably due to the small sample size, or enormous heterogeneity of the AD population, or other data issues.

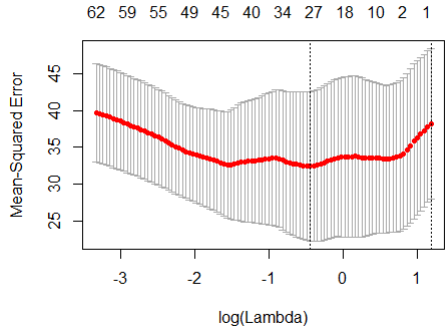

Figure 8.6: MSEs by cross-validation with different values of `lambda`

The best model can be queried by calling on the function `cv.glmnet` as shown in below (results omitted due to the space limit):

```
# To view the selected variables and the corresponding coefficien
ts
cv.fit$lambda.min

## [1] 0.6437308

coef(cv.fit, s = "lambda.min")
```

As LASSO has picked up the variables, we can further fit a regression model using these variables. Here, the reason that we need to re-fit the regression model is that, since LASSO uses the L_1 norm to push those insignificant predictors out of the model, it also results in shrinkage of the magnitudes of the predictors that are significant. This shrinkage is bias that makes the model less accurate in prediction. Usually, people tend to use LASSO for model selection only, e.g., to identify the predictors that are

significant. Then, with the identified predictors, a classic regression model is further applied on this reduced set of predictors to build the final model.

We follow this two-step strategy. As shown in below, it can be seen that these variables could lead to a model that has the `R-squared` as large as `0.9117`. It is also possible, from the results shown in below, that the model can be further simplified by deleting more insignificant variables.

```
# fit a linear regression model
trainX.reduced <- data.frame(trainX[, which(coef(cv.fit, s = "lam
bda.min") !=
    0) - 1])
tempData <- cbind(trainY, trainX.reduced)
lm.AD <- lm(trainY ~ ., data = tempData)
summary(lm.AD)

##
## Call:
## lm(formula = trainY ~ ., data = tempData)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.4283 -1.4383  0.4106  1.0739  3.3860
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 73.48975    0.42384 173.389  < 2e-16 ***
## ST103CV      0.68829    0.86621   0.795  0.43404
## ST103TA      0.82101    0.86085   0.954  0.34901
## ST106TA     -1.73112    0.63784  -2.714  0.01164 *
## ST106TS     -0.34535    0.60115  -0.574  0.57057
## ST110CV     -0.99903    0.75223  -1.328  0.19569
## ST110TA     -0.79230    0.64168  -1.235  0.22797
## ST113SA      0.06535    0.62399   0.105  0.91740
## ST118SA     -0.59835    0.69504  -0.861  0.39717
## ST119CV     -1.05435    0.83192  -1.267  0.21626
## ST119SA     -0.77869    0.77768  -1.001  0.32591
## ST128SV      1.63101    0.69748   2.338  0.02733 *
## ST129TS      0.98078    0.66285   1.480  0.15098
## ST130TS     -0.11623    0.58658  -0.198  0.84446
## ST14TS      -1.04771    0.55252  -1.896  0.06909 .
## ST16SV       0.34340    0.72793   0.472  0.64104
## ST17SV      -1.14047    0.57236  -1.993  0.05690 .
## ST26TS      -0.47550    0.56765  -0.838  0.40985
## ST35TS      -1.38430    0.53229  -2.601  0.01515 *
```

```
## ST39TS       1.34252    0.46659   2.877  0.00791 **
## ST42SV       0.96528    0.56439   1.710  0.09912 .
## ST44TS       0.61458    0.58486   1.051  0.30301
## ST45CV      -0.59449    0.55985  -1.062  0.29806
## ST59CV       2.02101    0.70023   2.886  0.00774 **
## ST62TS      -0.54468    0.46893  -1.162  0.25597
## ST74TS       0.10354    0.60269   0.172  0.86493
## ST7SV        0.81011    0.54650   1.482  0.15027
## ST83CV       0.80073    0.51498   1.555  0.13207
## ST83TA      -0.20721    0.79645  -0.260  0.79678
## ST85TS       0.84103    0.68633   1.225  0.23141
## ST98CV      -0.03397    0.71845  -0.047  0.96265
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.707 on 26 degrees of freedom
## Multiple R-squared:  0.9117, Adjusted R-squared:  0.8099
## F-statistic:  8.95 on 30 and 26 DF,  p-value: 1.203e-07
```

Note that, the `glmnet` package not only implements LASSO, but also other related models such as Ridge regression. For instance, via the R code below we can implement the Ridge regression by setting `alpha = 0`:

```
# Do a ridge regression instead
fit.ridge = glmnet(trainX, trainY, alpha = 0, nlambda = 100)
print(fit.ridge)
```

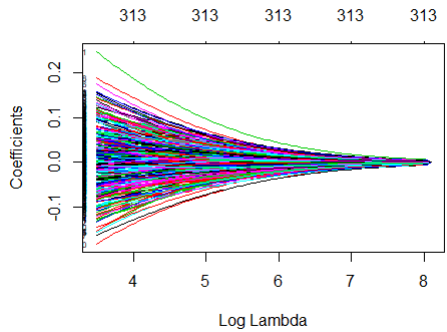

Figure 8.7: Path solution trajectory of the regression coefficients by Ridge regression

As shown in Figure 8.7, the path solution trajectory of the regression coefficients by Ridge regression has quite a different shape from the path solution trajectory of LASSO regression shown in Figure 8.3.

```
plot(fit.ridge, xvar = "lambda", label = TRUE)
```

We can also implement the idea of LASSO on logistic regression model. Here, we use the "DX_bl" as our outcome variable that has two classes, "NC" and "LMCI", denoting for the normal aging and mild cognitive impairment, respectively. Note that, classifying between NC and LMCI is very challenging, much more challenging than the classification between NC and AD that has been discussed in previous chapters, since the LMCIs are to certain degrees still normal individuals and are not clinically diagnosed with dementia yet.

```
# Fit a LASSO model for logistic regression
trainY <- as.matrix(trainset[, 2])
testY <- as.matrix(testset[, 2])
fit = glmnet(trainX, trainY, nlambda = 100, family = "binomial")
plot(fit, label = TRUE)
```

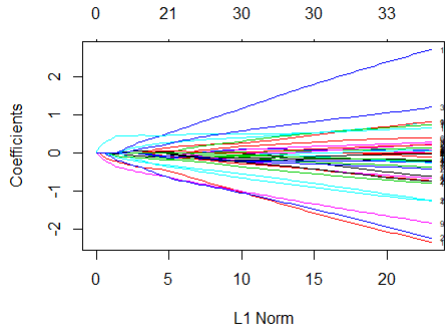

Figure 8.8: Path solution trajectory of the coefficients by logistic LASSO

Also, the cross-validation can be used to decide on the best model:

```
# Use cross-validation to decide which model is best
cv.fit = cv.glmnet(trainX, trainY, family = "binomial", type.meas
ure = "class")
plot(cv.fit)
```

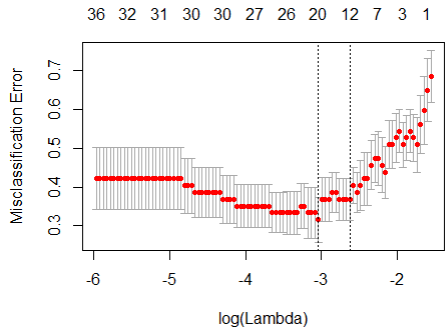

Figure 8.9: Classification errors by cross-validation with different values of lambda

As shown in Figure 8.9, a certain optimal point can be identified to decide on the best model. We can further output the corresponding coefficients of this optimal model via the R code below (results omitted due to space limit).

```
# To view the selected variables and the corresponding coefficien
ts
coef(cv.fit, s = "lambda.min")
```

Similarly in LASSO, we could use `predict()` to predict on the testing dataset using any model.

```
predict(cv.fit, newx = testX, s = "lambda.min")
```

```
##              1
## 3  -1.23155042
## 9  -0.32381239
## 17 -5.49829437
## 19 -0.47832582
## 29 -0.35677823
## 30 -0.82189141
## 31  0.62296595
## 38  0.52778295
## 39 -1.49146275
## 45 -2.54049855
## 48  1.07123484
## 52 -1.27364645
```

```
## 53  1.46493378
## 61 -1.21799365
## 62 -3.32616327
## 73  0.03668668
```

II.4 Remark

The shooting algorithm [1] has been widely used in many extension models of LASSO in the statistics community. The shooting algorithm is easy to use and has nice interpretation of each iteration. But it could be slow in very high-dimensional situations. Also, with more complex penalty terms such as those L_{21}-norm regularization or group regularization terms, the shooting algorithm may not work anymore. In machine learning community where the computational efficiency is of particular interest, many scalable algorithms such as the projection operator based methods have been developed. Interested readers can read more of these works[2] in this direction that provided closed form iterative updating rules by projection operator on a variety of regularization terms.

On the other hand, regarding why LASSO can produce sparse estimates of the regression parameters while Ridge regression could not, a deep reason was revealed in the "bible" book of statistical learning[3]. Here, we adopt an easy and more common sense explanation (which has also been a very famous example) as shown in the Figure 8.10. It shows the application of LASSO and Ridge regression models on a problem with 2 predictors. The contour plot corresponds to the least squares loss function which is shared by classic regression model, LASSO, and Ridge models. $\widehat{\boldsymbol{\beta}}$ in the

[1] Fu, WJ. Penalized regressions: the bridge versus the lasso. *Journal of Computational and Graphical Statistics*, 1998.

[2] http://www.yelab.net/software/SLEP/

[3] Hastie, T., Tibshiranim R. and Friedman, J. *The elements of statistical learning*, 2nd edition. Springer, 2009.

center of the contour is the least squares estimator of the regression parameters.

Figure 8.10 shows that why LASSO can generate sparse estimation of the model. As the objective function of LASSO consists of two terms, the optimal solution lies on the intersection of the geometric areas corresponding to the two terms. As LASSO uses L_1 norm, it results in those "sharp" corner points that mostly like to be the point of contacts of the two geometric areas. Those point of contacts are themselves sparse solutions, e.g., in Figure 8.10, the point of contact implies that $\beta_1 = 0$.

As a comparison, in Ridge regression, as the geometric area corresponding to the L_2 norm has no such "sharp" corner points, the model has no strong incentive for where to allocate the point of contacts of the two geometric areas. Thus, given the infinite number of potential point of contacts of the two geometric areas, it is expected that Ridge regression will not result in sparse solutions with exact zeros in $\widehat{\boldsymbol{\beta}}$.

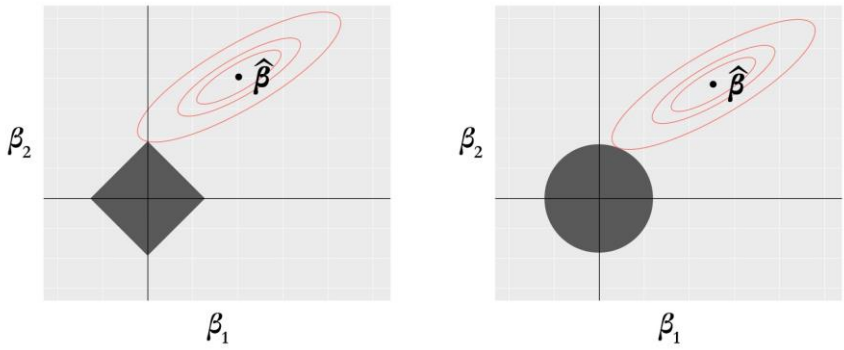

Figure 8.10: Why LASSO could generate sparse estimates while Ridge tends to not

Following this idea, the L_1 norm is later extended to L_q norm for $q \leq 1$. For any $q \leq 1$, we could generate those "sharp" corner points to enable sparse solutions of $\widehat{\boldsymbol{\beta}}$. The advantage of using $q < 1$ is to reduce bias in the model. Recall that, we have mentioned earilier, that LASSO will lead to bias in parameter estimation. Using $q < 1$ is a good approach to reduce this bias, while it still produces "sharp" corner points but penalizes less on the significant predictors. The cost of using $q < 1$ is that it will result in nonconcave penalty terms, making the overall objective function of the sparse model nonconcave.

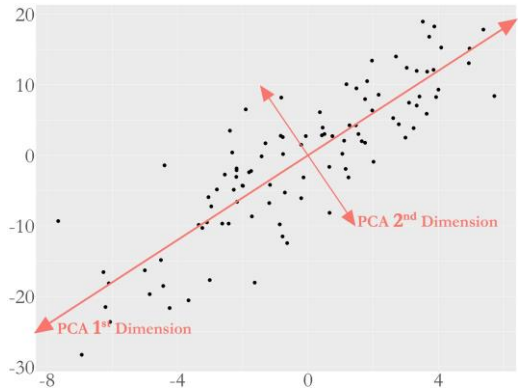

Figure 8.11: Illustration of the principal components in a dataset with 2 variables; the main variation source is represented by the 1st PC dimension

III. Principal Component Analysis

III.1 Rationale and Formulation

The PCA method is built on the assumption that, for a multivariate dataset that has many variables, the dimensionality of the dataset is smaller than it appears to be. In other words, for example, for one dataset that has 10 variables, we may have the impression that there are ten independent sources of variation that infuse uncertainty into the data. But, PCA is built

on the idea that the underlying independent sources of variations are only a few (e.g., 2 or 3 for 10 variables could be usual). The question is how to identify the intrinsic sources of the variation.

As shown in Figure 8.11, PCA pursues this idea of identifying the intrinsic sources of the variation in the framework of linear models. The characteristic shape of the dataset shown in Figure 8.11 indicates that, although the data points are located in a two-dimensional space, the data points are not totally randomly scattered all over the place. Rather, there is a force that orients these data points towards one direction (or, in the same effect, you may say there is a force that pushes the data points towards one narrow zone). To recover these forces, the PCA seeks linear combinations of the original variables to pinpoint the directions towards which the underlying forces are pushing the data points. In other words, another assumption of PCA is that the relationship between the underlying dimensions and the variables (surface dimensions) is linear.

III.2 Theory and Method

The idea shown in Figure 8.11 reveals the principle to guide the estimation of the linear weights to combine the variables. This leads to the following formulation:

$$\boldsymbol{w}_{(1)} = \arg \max_{\boldsymbol{w}_{(1)}^T \boldsymbol{w}_{(1)}=1} \left\{ \sum_{n=1}^{N} \boldsymbol{x}_{(i)} \cdot \boldsymbol{w}_{(1)} \right\},$$

where there are N samples and p variables, $\boldsymbol{x}_{(n)} \in R^{1\times p}$ is the nth sample, and $\boldsymbol{w}_{(1)} \in R^{p\times 1}$ is the linear weights vector of the first PC. Note that the constraint $\boldsymbol{w}_{(1)}^T \boldsymbol{w}_{(1)} = 1$ is to control the scale of the vector – without which an infinite number of solutions would exist. This also indicates that the absolute magnitudes of the weights are meaningless. Only the relative magnitudes are useful.

A more succinct form could be:

$$\boldsymbol{w}_{(1)} = \arg \max_{\boldsymbol{w}_{(1)}^T \boldsymbol{w}_{(1)}=1} \left\{ \boldsymbol{w}_{(1)}^T \mathbf{X}^T \mathbf{X} \boldsymbol{w}_{(1)} \right\},$$

where $\mathbf{X} \in R^{N \times p}$ is usually called the data matrix that concatenate all the N samples into a matrix. $\mathbf{X}^T\mathbf{X}$ is actually the sample covariance matrix.

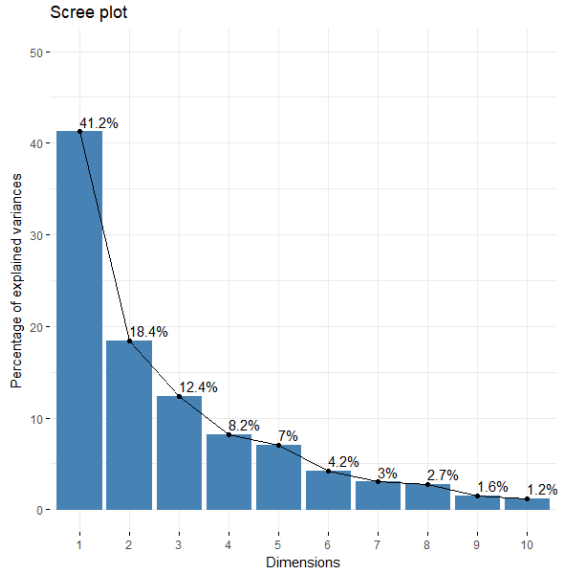

Figure 8.12: Scree plot that shows the first 5 PCs maybe significant, while the first PC is definitely a major variation source, together with the second and third PCs as other main variations sources

To identify the second PC, we could follow the principle of iteration. The idea is rather simple. As the first PC represents one variance source, and the original data $\mathbf{X}$ contains a linear aggregation of multiple variance sources, why not remove the first variance source from $\mathbf{X}$, then create a new data that contains the remaining variance sources? Then, the procedure for finding $\boldsymbol{w}_{(1)}$ could be readily used for finding $\boldsymbol{w}_{(2)}$, since with $\boldsymbol{w}_{(1)}$ removed, $\boldsymbol{w}_{(2)}$ is the largest variance source now.

This process could be generalized as:

- In order to find the kth PC, we could create a dataset as $\mathbf{X}_{(k)} = \mathbf{X} - \sum_{s=1}^{k-1} \mathbf{X}\boldsymbol{w}_{(s)}\boldsymbol{w}_{(s)}^T$.
- Then, we solve $\boldsymbol{w}_{(k)} = \arg\max_{\boldsymbol{w}_{(k)}^T\boldsymbol{w}_{(k)}=1} \left\{\boldsymbol{w}_{(k)}^T\mathbf{X}_{(k)}^T\mathbf{X}_{(k)}\boldsymbol{w}_{(k)}\right\}$ for identifying $\boldsymbol{w}_{(k)}$.

In practice, we need to decide how many PCs are needed to represent the dataset. In theory, for a dataset with p variables, there are p PCs that could be extracted if the dataset has more sample size than the number of variables. But it is often the case that only the first few PCs are needed since these few PCs could explain away majority of the variation in the data. The scree plot as shown in Figure 8.12 is a common tool in practice, that draws the eigenvalues of the PCs to look for the change point beyond which the PCs maybe statistically insignificant.

We could use the following example to practice this procedure. The dataset is shown in the Table below:

Table 8.2: A dataset example for PCA

X_1	X_2
-1	0
3	3
3	5
-3	-2
3	4
5	6
7	6
2	2

First, we can calculate the sample covariance matrix as

$$\mathbf{S} = \mathbf{X}^T\mathbf{X} = \begin{bmatrix} 115 & 118 \\ 118 & 130 \end{bmatrix}.$$

We can obtain the $\boldsymbol{w}_{(1)}$ by

$$\boldsymbol{w}_{(1)} = \arg \max_{\boldsymbol{w}_{(1)}^T \boldsymbol{w}_{(1)}=1} \left\{ \boldsymbol{w}_{(1)}^T \mathbf{S} \boldsymbol{w}_{(1)} \right\}.$$

The lagrangian form is

$$\boldsymbol{w}_{(1)}^T \mathbf{S} \boldsymbol{w}_{(1)} - \lambda_1 \boldsymbol{w}_{(1)}^T \boldsymbol{w}_{(1)}.$$

By taking the derivative of the lagrangian form with regards to $\boldsymbol{w}_{(1)}$, it is not hard to arrive at the equation:

$$\mathbf{S} \boldsymbol{w}_{(1)} - \lambda_1 \boldsymbol{w}_{(1)} = 0.$$

Thus, this is an eigenvalue problem of the matrix $\mathbf{S}$. We can solve it as $\lambda_1 = 240.74$ and $\boldsymbol{w}_{(1)} = [0.68, 0.73]$. Further, we can get that $\lambda_2 = 4.26$ and $\boldsymbol{w}_{(2)} = [-0.73, 0.68]$.

III.3 R Lab

We apply PCA on the AD data that has been used in the R lab of LASSO. There are many packages in R that can conduct PCA. Here, we use the function `PCA()` in the "`FactoMineR`" package.

```
# Implement principal component analysis on the AD data
# install.packages('factoextra')
require(factoextra)
require(FactoMineR)
require(ggfortify)
tempData <- AD[, c(17:dim(AD)[2])]
# Conduct the PCA analysis
pca.AD <- PCA(tempData, graph = FALSE, ncp = 10)
```

The construct `pca.AD` has the eigenvalues and the principal components. We can use `fviz_screeplot()` to visualize the contributions of the principal components, as shown in Figure 8.13.

```
# Examine the contributions of the PCs in explaining the variatio
n in data
fviz_screeplot(pca.AD, addlabels = TRUE, ylim = c(0, 50))
```

It can be seen from Figure 8.13 that the first PC could explain away 16.7% of the total variation and the second PC could explain away 12.7% of the total variation. It seems that there is a change point at the third PC, showing that the following PCs *could* be insignificant.

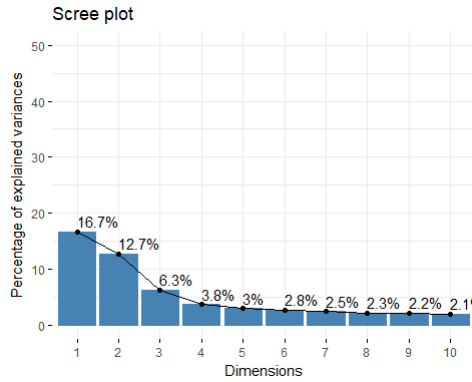

Figure 8.13: Scree plot of the PCA analysis on the AD dataset

We could also show the numerical details of the loadings of the variables in the PCs.

```
# Examine the loadings of the variables in the PCs
var <- get_pca_var(pca.AD)
head(var$contrib)

##                Dim.1      Dim.2       Dim.3      Dim.4      Dim.5
## ST101SV 5.217488e-01 0.00543192 0.022513018 0.00695649 0.55635835
## ST102CV 6.331461e-01 0.26061413 0.167089523 0.02972375 0.11789451
## ST102SA 1.029105e+00 0.00198550 0.011535559 0.22360153 0.17723377
## ST102TA 1.080058e-02 1.06804755 0.237858049 0.38424318 0.00483804
## ST102TS 5.979293e-05 0.06038710 0.299593782 0.52436240 0.02805072
## ST103CV 8.089453e-02 0.05503171 0.001267605 0.54330434 3.08447026
##               Dim.6       Dim.7       Dim.8       Dim.9       Dim.10
## ST101SV 1.835299776 0.498169844 0.055960987 0.018902630 0.0517726064
## ST102CV 0.037225860 0.182728808 0.039477823 0.029123146 0.1602108140
```

```
## ST102SA 0.336057060 0.006782785 0.008738713 0.009108956 0.0002884433
## ST102TA 0.167160010 0.221663964 0.273441045 0.003510227 0.2088207321
## ST102TS 0.008515869 0.167631015 0.074972768 0.022843632 1.9489628313
## ST103CV 0.175682030 0.822144415 1.592278638 0.251205409 0.1727957251
```

It is also helpful to visualize the contributions of the variables to the PCs by figures. For example, Figures 8.14 and 8.15 show the contributions of the variables to the first and second PC, respectively.

```
fviz_contrib(pca.AD, choice = "var", axes = 1, top = 20)
fviz_contrib(pca.AD, choice = "var", axes = 2, top = 20)
```

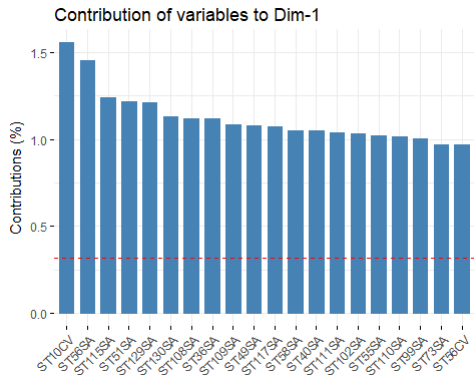

Figure 8.14: Contributions of the variables for the first PC

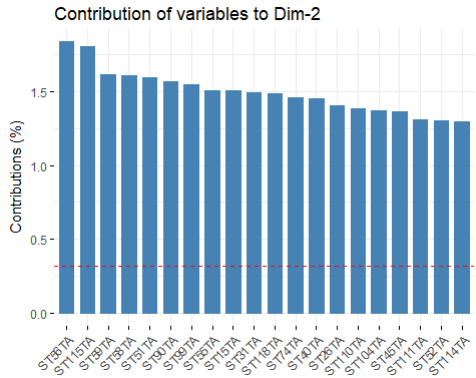

Figure 8.15: Contributions of the variables for the second PC

Figure 8.16 is another way to visualize the contributions of the variables to the PCs.

```
fviz_pca_var(pca.AD, col.var = "contrib", select.var = list(contr
ib = 20), gradient.cols = c("#00AFBB",
    "#E7B800", "#FC4E07"), repel = TRUE  # Avoid text overlapping
)
```

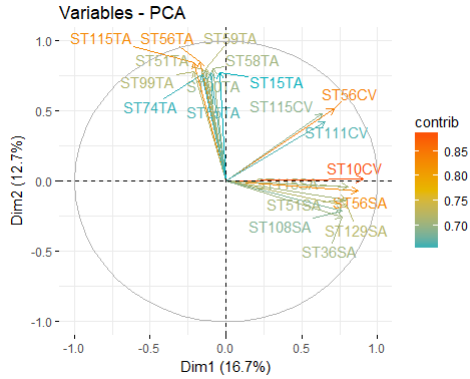

Figure 8.16: Loadings of the top variables in the first and second PCs

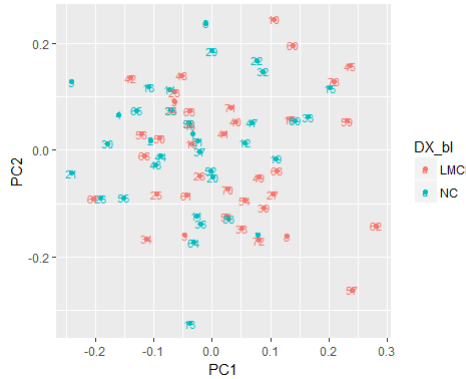

Figure 8.17: Scatterplot of the subjects in the space defined by the first and second PCs

With the identified PCs, we can visualize the distribution of the data points in this new space spanned by the PCs. Sometimes, it may reveal some inherent structure of the dataset. For example, for a classification problem, it is hoped that the data points from different classes would cluster around different centers in the space spanned by the PCs. In our case, as shown in Figure 8.17, this cluster structure is not perfect but seems to be on the borderline of significance, as it is not entirely like a pure random pattern.

```
# Examine the projection of data points in the new space defined
by PCs
autoplot(prcomp(tempData), data = AD, colour = "DX_bl", label = T
RUE, label.size = 3)
```

The PCs can be taken as new variables. For example, we can build regression models using the PCs to predict outcome variables. Here, we use `AGE` as the outcome, and first fit a regression model with 10 PCs.

```
# fit a regression model using the PCs
tempData <- data.frame(cbind(AD[, 5], pca.AD$ind$coord))
names(tempData) <- c("AGE", "PC1", "PC2", "PC3", "PC4", "PC5", "P
C6", "PC7",
    "PC8", "PC9", "PC10")
lm.AD <- lm(AGE ~ ., data = tempData)
summary(lm.AD)
```

It seems that the first PC is not significant, while the second, the third, and the fifth PCs are significant. It is not unusual to see that the first PC is insignificant, as the first PC sometimes may embody a variation source that is not correlated with the outcome variable. It is always a challenge to interpret the results of PCA, particularly, to interpret the physical correspondence of the PCs. On the other hand, we can see that the `R-squared` by this model is `0.3672`, and the `p-value` is as small as 0.0008235, indicating the overall model is significant. Further variable selection would be conducted to prune the model and reduce its complexity.

```
##
## Call:
## lm(formula = AGE ~ ., data = tempData)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -17.3377  -2.5627   0.0518   2.6820  11.1772
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 73.68767    0.59939 122.938  < 2e-16 ***
## PC1          0.04011    0.08275   0.485 0.629580
## PC2         -0.31556    0.09490  -3.325 0.001488 **
## PC3          0.50022    0.13510   3.702 0.000456 ***
## PC4          0.14812    0.17462   0.848 0.399578
## PC5          0.47954    0.19404   2.471 0.016219 *
## PC6         -0.29760    0.20134  -1.478 0.144444
## PC7          0.10160    0.21388   0.475 0.636440
## PC8         -0.25015    0.22527  -1.110 0.271100
## PC9         -0.02837    0.22932  -0.124 0.901949
## PC10         0.16326    0.23282   0.701 0.485794
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.121 on 62 degrees of freedom
## Multiple R-squared:  0.3672, Adjusted R-squared:  0.2651
## F-statistic: 3.598 on 10 and 62 DF,  p-value: 0.0008235
```

III.4 Remark

While PCA has been widely used, it is often criticized as a black box model or lack interpretability since it is always not easy to connect the identified principal components with physical entities. But, probably, we may stop worry about validity of the PCA method in many applications and focus on the significance it can reveal. After all, when studying real-world systems that we haven't known what we don't know yet, we have to make bold hypothesis and make the leap over the gaps. This is probably one of the reasons why PCA has been applied in many real world applications. The massive practices of PCA in many areas have formed a convention, or a myth – some critical statisticians may say – that formulistic rubrics have been invented to help beginners to quickly jump in to the vehicle of PCA

and start to convert their very challenging data into PCA patterns, then further convert these patterns into formulated sentences such as "the variables that have larger magnitudes in the first 3 PCs correspond to the brain regions in hippocampus areas, indicating that these brain regions manifest significant functional connectivity to deliver the verbal function", or "we have identified 5 significant PCs, and the genes that show dominant magnitudes in the linear weights vector are all related to T-cell production and immune functions – thereby each of the PC indicates a biological pathway that consists of these constitutional genes working together to produce specific types of proteins". You may also hear from some financial analysts who presented such a result: "through PCA on 100 stocks, we found that the first PC consists of 10 stocks as their weights are significantly larger than the other stocks. This may indicate that there is strong correlation between these 10 stocks and you may consider this fact when you define your investment strategy".

IV. Variable Selection by Random Forests

III.1 Rationale and Formulation

As we have seen that, both LASSO and PCA are linear models, which are not suitable if there are nonlinear relationships in the dataset. For nonlinear variable selection, the random forests have been commonly used. Recall that the random forests consist of decision nodes that are defined by splits on some variables. This can provide information about the variables' importance in the random forests. Also, random forests are powerful in capturing nonlinear and predictive information from data, and therefore, provide data-driven characterization of variable importance. Third, random forests require little data preprocessing. They can handle different scales of the continuous variables since the impurity gain is calculated based on the outcome variable, and can work with both categorical and numerical variables. Given these advantages, methods have been developed to conduct variable selection using random forest models.

III.2 Theory and Method

Variable importance scores: The importance score of a variable can be measured based on the Impurity gain. For classification problems, the Gini index for the data points at a node is defined as

$$Gini = \sum_{c=1}^{C} p_c(1 - p_c),$$

where C is the number the classes in the data set, and p_c is the proportion of the data instances of class c.

Assuming that a variable is used for splitting the node into n children nodes. The Gini gain of the variable can be calculated as

$$\nabla\, Gini = Gini - \sum_{i=1}^{n} w_i * Gini_i,$$

where $Gini$ is the Gini index at the node to be split; w_i and $Gini_i$ are the percentage of data instances of the entire dataset and the Gini index at the i^{th} node, respectively.

Then, the importance score of a variable can be calculated as

$$\frac{1}{ntree} \sum_{i=j_1}^{j_m} \nabla\, Gini_i,$$

where j_1, …, j_m are the nodes where the variable j is used for splitting, $\nabla\, Gini_i$ is the Gini gain at node i, and $ntree$ is the number of trees in the random forest model.

In what follows, we show how this can be done using a small data example.

Consider the following data example shown in Table 8.3. Note that X_1 and X_2 are identical, and thus one of them is redundant.

Table 8.3: A dataset example for RF

ID	X_1	X_2	X_3	X_4	Class
1	1	1	0	1	C0
2	0	0	0	1	C1
3	1	1	1	1	C1
4	0	0	1	1	C1

Assume that a random forest model is built on the data set with two trees, show in Figure 8.18 and Figure 8.19, respectively. The Gini index at each node is also shown in the Figures.

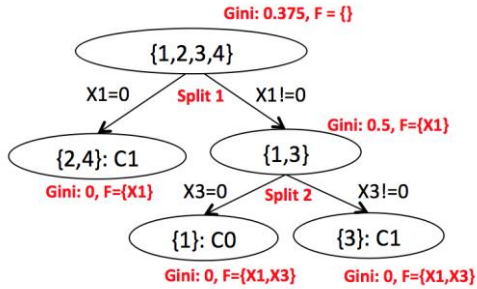

Figure 8.18: Tree 1 in a random forest model

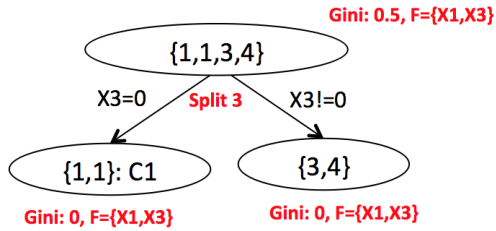

Figure 8.19: Tree 2 in a random forest model

Now, we calculate the importance score for each variable.

At split 1, the Gini gain for X_1 is calculated as

$$0.375 - 0.5*0 - 0.5*0.5 = 0.125.$$

At split 2, the Gini gain for X_3 is 0.5.

At split 3, the Gini gain for X_2 is $0.5 - 0.25*0 + 0.75*0.44 = 0.17$.

At split 4, the Gini gain for X_3 is 0.44.

Therefore, the importance score of X_1, X_2, X_3 and X_4, are 0.125/2 = 0.0625, 0.17/2=0.085, (0.5+0.44)/2=0.47, and 0 (there is not split using X_4), respectively.

Note that the example above is about classification problems. For regression problems, the mean squared error can be used as the impurity measure:

$$MSE = \sum_i (y_i - \bar{y})^2,$$

where y_i is the value of the outcome variable of the i^{th} data instance at a node, and $\bar{\bar{y}}$ is the average of the outcome variable of all the data instance at the node.

Regularized random forests: We can see that, in this calculation, the Gini gains are added equally across all the nodes, and a split that perfectly separates 2 data points has the same Gini gain as a split that perfectly separates 100 data points. Thus, the variable importance score built on this Gini gain can be sensitive to noise. In addition, since the importance score of a variable depends on only the splits where the variable is used, the existence of correlation or redundancy between the variables can make this concept of importance score misleading. In the illustrative example abovementioned, since X_1 and X_2 are identical, their importance scores should be the same. However, this is not the case, as in building the trees, the random forest model randomly selects either of them to split the nodes. This results in a dilution effect in the estimation of their importance scores. Thus, the true importance score of either X_1 or X_2 should be the sum of the obtained importance scores from an established random forest model.

This is just a case of two redundant variables. As the number of highly correlated variables increases, the importance scores for each are expected to further decrease.

Thus, the importance scores of the variables provided by the random forests do not consider variables redundancy. In the illustrative example, both X_1 and X_2 are used for splitting nodes and generating impurity gain, but only one of them is needed for the prediction task as they are essentially

the same. To overcome this limitation, we introduce the Regularized random forests (RRF)[1] that can generate a relevant and non-redundant variable subset.

The RRFs are built sequentially. A key difference between RRF and ordinary random forests is that the RRF uses the regularized impurity gain for evaluating the splitting criteria. The regularized impurity gain of variable X_i at a node is calculated as

$$Gain'(X_i) = \begin{Bmatrix} \lambda \cdot Gain(X_i) & X_i \notin F \\ Gain(X_i) & X_i \in F \end{Bmatrix},$$

where $Gain(X_i)$ is an ordinary impurity gain, e.g., Gini index gain and reduction of mean square error; F is a feature subset including features used to split the previous nodes and is an empty set at the first node of the first tree; $\lambda \in (0,1]$ is referred as the **coefficient** and is used to penalize the gain if X_i is used in previous split.

In RRF, if a variable X_i that is not present in F produces more information gain than the variables in F, it will be used for splitting the node and further added into F. Also, not like in random forests where the number of features M is randomly selected and tested at each node, in RRF, all features from F and a subset of features randomly selected from $\bar{F}$ (features that do not belong to F) are tested. The size of the subset can be set to the minimum of M and size of $\bar{F}$.

We illustrate how the RRF can be built with the data example shown in Table 8.3. Here, we set $\lambda = 0.8$, and $M = 1$.

First, look at the tree shown in Figure 8.18. At split 1, F is an empty set. Assuming that, still, X_1 is used for testing the split. The regularized Gini gain for $X_1 = 0$ is calculated as

$$0.8 * (0.375 – 0.5*0 – 0.5*0.5) = 0.8 * 0.125 = 0.1.$$

After split 1, $F = \{X_1\}$.

---

[1] *Deng, H. and Runger, G. Gene selection with guided regularized random forest. Pattern recognition, 2013.*

At split 2, suppose X_3 is selected for testing the split as it is not in F yet. The regularized Gini gain for $X_3 = 0$ is 0.8 * 0.5 = 0.4. As all the other variables in F should be tested, we can also get that the regularized Gini gain for X_1 is 0. Therefore, X_3 is still the best variable for splitting the node.

After split 3, $F = \{X_1, X_3\}$.

It can be seen that the first tree grown by RRF is the same as the one from the earlier section. Now consider the second example as shown in Figure 8.19.

At split 3, suppose X_2 is still used for testing the node. As X_2 is not in F yet, the regularized Gini gain of $X_2 = 0$ is 0.8 * (0.5 – 0.25*0 + 0.75*0.44) = 0.8 * 0.17 = 0.136. The regularized Gini gain for $X_1 = 0$ is 0.5 – 0.25*0 + 0.75*0.44 = 0.17 as X_1 is in F and is not penalized. The regularized Gini gain for $X_3 = 0$ is 0.5.

Therefore, X_3 is used for splitting the node. Both children nodes have only one class and so are made as leaf nodes.

It can be seen from the second tree building process, the redundant feature, X_2, is penalized and has less gain than its identical variable X_1. Also, X_3 is now used for splitting the node as it has the strongest impurity gain. Thus, the variable subset selected from the two trees are $F = \{X_1, X_3\}$.

While the RRF is a remedy to overcome redundancy of variables, the guided regularized random forests (GRRF) can further enhance RRF when the sample size is small. This is because that, since a tree recursively splits the training data points, the number of data points can be small when the tree reaching a certain depth. The evaluation criterion may not be accurate when the number of data points is small and could add noise to variable selection. The GRRF can be used to reduce the chance an irrelevant or redundant variable being selected when the number of instances is small.

In GRRF, instead having one λ for all variables, each variable X_i can have its own λ_i:

$$Gain'(X_i) = \begin{Bmatrix} \lambda_i \cdot Gain(X_i) & X_i \notin F \\ Gain(X_i) & X_i \in F \end{Bmatrix},$$

where λ_i is

$$\lambda_i = (1-\gamma)\lambda_0 + \gamma * w_i,$$

where λ_0 controls the base regularization, $w_i \in [0,1]$ is a prior of importance of each variable v_i, and $\gamma \in [0,1]$ controls the weight from the prior. Note RRF is a special case of GRRF when $\gamma = 0$. w_i can be determined by prior knowledge about the variables, or can be generated from the normalized importance scores (between 0 and 1) from random forests. The importance scores aggregate the impurities gains from all trees, and therefore, are expected to be less noisy than the impurity gain calculated only from a single node.

Suppose at the first split of the first tree shown in Figure 8.18, two variables X_1 and X_2 are selected for testing, where $w_1 = 0.6$, $w_2 = 1$, $\lambda_0 = 0.9$, and $\gamma = 0.5$. Then, the impurity gain in GRRF for X_1 is calculated as

(0.5*0.9 + 0.5*0.6)*0.125.

And the impurity gain for X_2 is

(0.5*0.9 + 0.5*1)*0.125.

Therefore, even the original impurity gain for the two variables is the same, with a prior weight, X_2 is preferred to split the node.

III.3 R Lab

We use the extended AD dataset. Further, we add redundant variables, i.e., the number of variables in this manipulated dataset are 4 times of the number of original variables. We then use all the features to predict the age as a classification problem, i.e., we discretize the variable "AGE" to create a binary variable by its mean value.

First, we apply random forests to this data set. The importance scores of the variables are plotted in Figure 8.20. The variable names are omitted due to limited space.

```
require(inTrees)
require(randomForest)
require(RRF)
set.seed(1)
theme_set(theme_gray(base_size = 18))
```

```
path <- "../../data/AD_hd.csv"
data <- read.csv(path, header = TRUE)
data$AGE <- as.factor(dicretizeVector(data$AGE, K = 2))
target <- data$AGE
rm_indx <- which(colnames(data) %in% c("AGE", "ID", "TOTAL13", "M
MSCORE"))
X <- data[, -rm_indx]
X1 <- cbind(X, X, X)
colnames(X1) <- paste0("Y", 1:ncol(X1))
for (i in 1:ncol(X1)) {
    perc <- 0.1
    index <- sample(nrow(X1), floor(nrow(X1) * perc))
    X1[, i][sort(index)] <- (X1[, i])[index]
}
X <- cbind(X, X1)
rf <- randomForest(X, target)
imp <- as.data.frame(rf$importance)
colnames(imp)[colnames(imp) == "MeanDecreaseGini"] <- "importance
"
imp <- imp[order(imp$importance, decreasing = TRUE), , drop = FAL
SE]
imp$variable <- rownames(imp)
imp$variable <- factor(imp$variable, levels = as.character(imp$va
riable))
ggplot(data = imp, aes(x = variable, y = importance)) + geom_bar
(stat = "identity",
    aes(factor(variable)), fill = "red") + theme(axis.text.x = el
ement_blank())
```

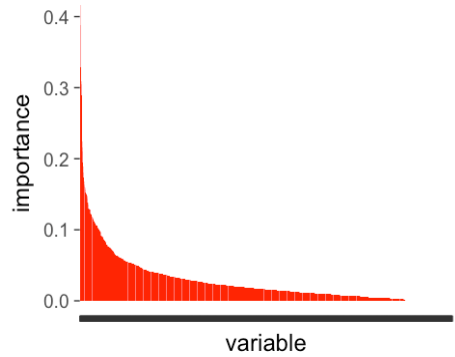

Figure 8.20: The important score of the variables by RF

From Figure 8.20 we can see a ranking of all variables in terms of their predictive powers. The top variables are ST62TA, ST59TS, ST56TA, ST58CV, and ST26TS. However, as we have demonstrated on the exemplary dataset shown in Table 8.3, the important scores of the variables are actually diluted by the redundancy of the variables. Thus, the observation that a large number of variables have non-zero importance scores in Figure 8.20 just amplifies the suspicion that there may be many redundant variables.

Now, let's apply the RRF to the data. The importance scores from the RRF are plotted in Figure 8.21.

```
rrf <- RRF(X, target)
imp <- as.data.frame(rrf$importance)
colnames(imp)[colnames(imp) == "MeanDecreaseGini"] <- "importance
"
imp <- imp[order(imp$importance, decreasing = TRUE), , drop = FAL
SE]
imp$variable <- rownames(imp)
imp$variable <- factor(imp$variable, levels = as.character(imp$va
riable))
ggplot(data = imp, aes(x = variable, y = importance)) + geom_bar
(stat = "identity",
    aes(factor(variable)), fill = "red") + theme(axis.text.x = el
ement_blank())
```

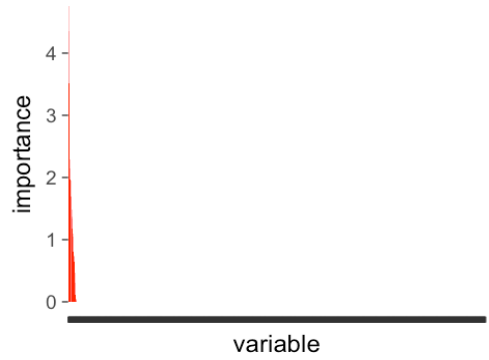

Figure 8.21: The important score of the variables by RRF

Clearly, as shown in Figure 8.21, a much smaller number of variables have non-zero importance scores, compared to ordinary random forests.

This demonstrates the superior capacity of RRF to deal with redundant variables than regular RF models.

Now, let's apply the GRRF to this dataset. The importance scores from the GRRF are shown in Figure 8.22. Similarly, the number of variables with non-zero importance scores is much smaller than ordinary random forests.

```
rf <- randomForest(X, target)
impRF <- rf$importance
impRF <- impRF[, "MeanDecreaseGini"]
imp <- impRF/(max(impRF))  #normalize the importance scores into
[0,1]
gamma <- 0.1
coefReg <- (1 - gamma) * 1 + gamma * imp  # each variable has a c
oefficient, which depends on the importance score from the ordina
ry RF and the parameter: gamma
grrf <- RRF(X, target, flagReg = 1, coefReg = coefReg)

imp <- as.data.frame(grrf$importance)
colnames(imp)[colnames(imp) == "MeanDecreaseGini"] <- "importance
"
imp <- imp[order(imp$importance, decreasing = TRUE), , drop = FAL
SE]
imp$variable <- rownames(imp)
imp$variable <- factor(imp$variable, levels = as.character(imp$va
riable))
ggplot(data = imp, aes(x = variable, y = importance)) + geom_bar
(stat = "identity",
    aes(factor(variable)), fill = "red") + theme(axis.text.x = el
ement_blank())
```

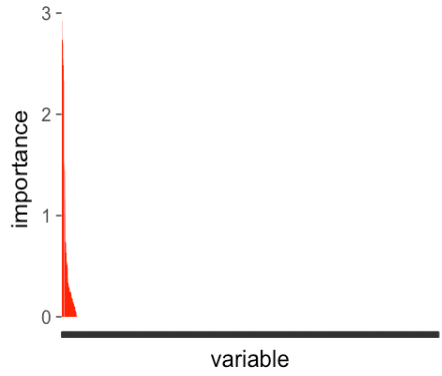

Figure 8.22: The important score of the variables by RRF

The previous figures illustrate that both RRF and GRRF use a much smaller number of variables to predict, compared to ordinary random forests. Now we evaluate the quality of the variable subset by the classification error.

Here is the procedure of evaluating the variable selection method. The dataset is split into training and testing sets with a 2:1 ratio. Different variable selection methods (e.g., by RF, RRF, and GRRF) are applied to the training set, and then, variable subsets are selected. Then, ordinary random forests are trained on the reduced training set, and applied to the testing set such that we can obtain the classification error. Each of the following experiments is run 50 times to get a robust estimate of the error rate. Note that, here, we first create the indices for the testing set (50 replicates), so that the later experiments can all consistently use the same indices.

The first variable selection method is to select the top K variables that have the top K importance scores from random forests. We study this method by changing K from 1 to 200. Results are shown in Figure 8.23.

```
set.seed(1)
testing.indices <- NULL
for (i in 1:50) {
    testing.indices <- rbind(testing.indices, sample(nrow(data),
floor(1 * nrow(data)/3)))
}

err.mat.rf <- NULL
for (K in c(1, (1:10) * 10, 150, 200)) {
    pred <- NULL
    for (i in 1:nrow(testing.indices)) {

        testing.ix <- testing.indices[i, ]
        X.training <- X[-testing.ix, ]
        target.training <- target[-testing.ix]
        X.testing <- X[testing.ix, , drop = FALSE]
        target.testing <- target[testing.ix]

        rf <- randomForest(X.training, target.training)
        impRF <- rf$importance
        impRF <- impRF[, "MeanDecreaseGini"]
```

```
        ix <- order(impRF, decreasing = TRUE)

        X.training.new <- X.training[, ix[1:K], drop = FALSE]
        rf <- randomForest(X.training.new, target.training)

        target.pred <- predict(rf, X.testing)

        error <- length(which(as.character(target.pred) != target.
testing))/length(target.testing)
        err.mat.rf <- rbind(err.mat.rf, c(K, error))
    }
}
err.mat.rf <- as.data.frame(err.mat.rf)
colnames(err.mat.rf) <- c("num_features", "error")

ggplot() + geom_boxplot(data = err.mat.rf %>% mutate(num_features
 = as.factor(num_features)),
    aes(y = error, x = num_features)) + geom_point(size = 3)
```

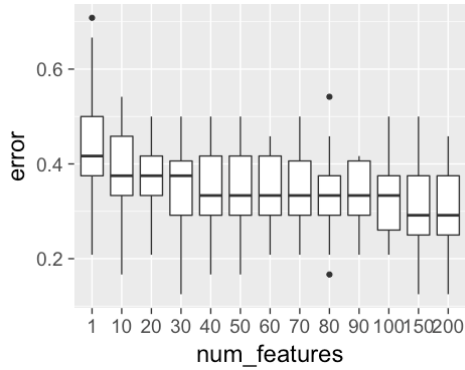

Figure 8.23: The error rates of the RF models with different number of features

From Figure 8.23, it can be seen that the error rates decrease as the number of variables increases. This makes sense as additional information is provided to the model to predict the outcome variable as more variables are added.

Next, we use RRF to conduct variable selection. This can be done by changing the coefficient parameter (`coefReg`) in RRF. The number of

selected variables should increase as the coefficient increases, and the error rates should decrease accordingly. Results are shown in Figure 8.24 and Figure 8.25.

```
set.seed(1)
err.mat.rrf <- NULL
for (coefI in c(0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 1)) {
    pred <- NULL
    for (i in 1:nrow(testing.indices)) {
        testing.ix <- testing.indices[i, ]
        X.training <- X[-testing.ix, ]
        target.training <- target[-testing.ix]
        X.testing <- X[testing.ix, , drop = FALSE]
        target.testing <- target[testing.ix]

        rrf <- RRF(X.training, target.training, coefReg = coefI)

        X.training.new <- X.training[, rrf$feaSet, drop = FALSE]
        rf <- randomForest(X.training.new, target.training)

        target.pred <- predict(rf, X.testing)
        # pred <- c(pred, as.character(target.pred) )
        error <- length(which(as.character(target.pred) != target.
testing))/length(target.testing)
        err.mat.rrf <- rbind(err.mat.rrf, c(coefI, length(rrf$fea
Set), error))
    }
    # error <- length(which(pred != target))/length(pred) err.mat.
rrf <-
    # rbind(err.mat.rrf, c(coefI, mean(num.fea.v), error))
}
err.mat.rrf <- as.data.frame(err.mat.rrf)
colnames(err.mat.rrf) <- c("coef", "num_features", "error")
# err.mat.rrf <- err.mat.rrf %>% mutate(coef=as.factor(coef))
ggplot() + geom_boxplot(data = err.mat.rrf %>% mutate(coef = as.f
actor(coef)),
    aes(y = error, x = coef)) + geom_point(size = 3)
```

It is known that the maximum number of features are selected when the coefficient becomes 1. As shown in Figure 8.24, when the coefficient is around 0.95, the smallest error rates could be obtained. Also, from Figure 8.25, we can see that, indeed the number of selected variables increases when the coefficient increases. Thus, RRF provides a continuum of variable

selection controlled by the parameter `coefReg`, providing convenience in model tuning and cross-validation.

```
ggplot() + geom_boxplot(data = err.mat.rrf %>% mutate(coef = as.f
actor(coef)),
    aes(y = num_features, x = coef)) + geom_point(size = 3)
```

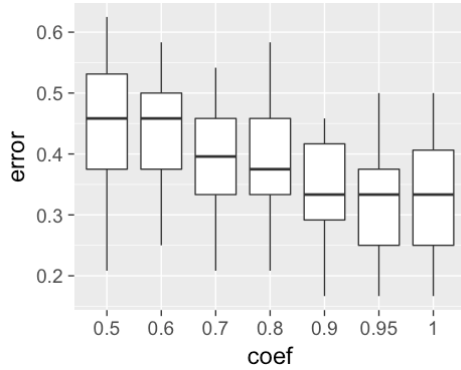

Figure 8.24: The error rates of the RF models with different values of coef

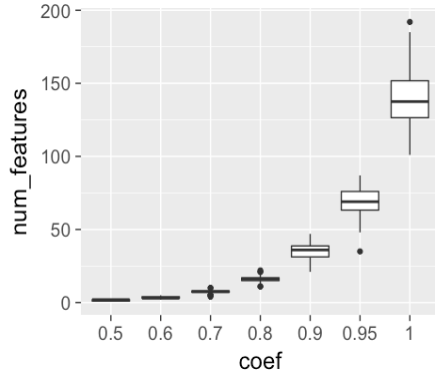

Figure 8.25: Number of features versus different values of coef

Furthermore, we conduct the variable selection using GRRF. We use different γ in GRRF and check the number of features and error rates. The weights come from the importance scores from random forests. As shown in Figure 8.26 and Figure 8.27, we can observe that the number of features decreases as γ increases, and the error rates increase accordingly.

```
set.seed(1)
err.mat.grrf <- NULL
for (gammaI in c(0.4, 0.3, 0.2, 0.1, 0.05, 0)) {
    pred <- NULL
    num.fea.v <- NULL

    for (i in 1:nrow(testing.indices)) {
        testing.ix <- testing.indices[i, ]
        X.training <- X[-testing.ix, ]
        target.training <- target[-testing.ix]
        X.testing <- X[testing.ix, , drop = FALSE]
        target.testing <- target[testing.ix]

        rf <- randomForest(X.training, target.training)
        impRF <- rf$importance
        impRF <- impRF[, "MeanDecreaseGini"]
        imp <- impRF/(max(impRF))
        coefReg <- (1 - gammaI) * 1 + gammaI * imp

        grrf <- RRF(X.training, target.training, flagReg = 1, coe
fReg = coefReg)

        # num.fea.v <- c(num.fea.v, length(grrf$feaSet))
        X.training.new <- X.training[, grrf$feaSet, drop = FALSE]

        rf <- randomForest(X.training.new, target.training)

        target.pred <- predict(rf, X.testing)
        # pred <- c(pred, as.character(target.pred) )
        error <- length(which(as.character(target.pred) != target.
testing))/length(target.testing)
        err.mat.grrf <- rbind(err.mat.grrf, c(gammaI, length(grrf
$feaSet), error))
    }
}

err.mat.grrf <- as.data.frame(err.mat.grrf)
colnames(err.mat.grrf) <- c("gamma", "num_features", "error")
```

```
# err.mat.grrf <- err.mat.grrf %>% mutate(gamma=as.factor(gamma))
ggplot() + geom_boxplot(data = err.mat.grrf %>% mutate(gamma = as.
factor(gamma)),
    aes(y = error, x = gamma)) + geom_point(size = 3)
```

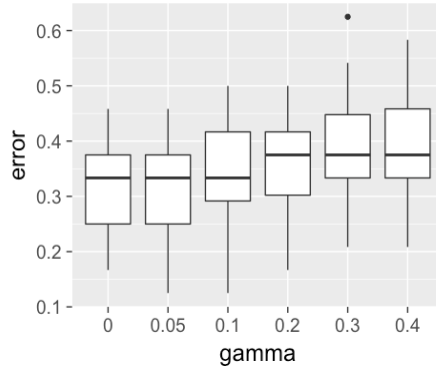

Figure 8.26: The error rates of the RF models with different values of gamma in RRF

```
ggplot() + geom_boxplot(data = err.mat.grrf %>% mutate(gamma = as.
factor(gamma)),
    aes(y = num_features, x = gamma)) + geom_point(size = 3)
```

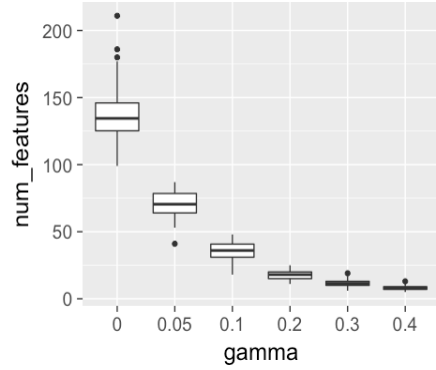

Figure 8.27: Number of features versus different values of gamma

Now, we compile the results from all the methods, and plot the average error rates at each number of features in Figure 8.28. For RRF and GRRF, the average number of features and average error rates of each parameter setting are used. It can be seen that between 40 and 80 features, the RRF and GRRF methods have lower error rates than the RF model that uses the top K features according to the importance scores generated by the RF model. As K increases, the error rate of using the top K variables according to the RF importance scores continues to decrease, lower than using the variables selected from RRF or GRRF. This means that RRF and GRRF can miss some informative variables. However, since this dataset is small, and the use of cross-validation makes it even smaller, the difference may not be as significant. To verify this hypothesis, we also plot the average errors with +/- 1 standard deviation in Figure 8.29. It can be seen that all the methods have similar error rate ranges when a certain number of variables is selected. However, an advantage for RRF and GRRF is that they can efficiently determine the number of variables needed by changing the coefficient or γ.

```
err.mat.rf <- as.data.frame(err.mat.rf)
err.mat.rf$method <- "RF"
err.mat.rrf <- as.data.frame(err.mat.rrf)
err.mat.rrf$method <- "RRF"
err.mat.grrf <- as.data.frame(err.mat.grrf)
err.mat.grrf$method <- "GRRF"
err.mat.rrf.summary <- err.mat.rrf %>% group_by(coef, method) %>%
 summarize(num_features = mean(num_features),
    sd = sd(error), error = mean(error), upper = error + sd, lowe
r = error -
        sd) %>% ungroup()
err.mat.grrf.summary <- err.mat.grrf %>% group_by(gamma, method)
 %>% summarize(num_features = mean(num_features),
    sd = sd(error), error = mean(error), upper = error + sd, lowe
r = error -
        sd) %>% ungroup()
err.mat.rf.summary <- err.mat.rf %>% group_by(num_features, metho
d) %>% summarize(sd = sd(error),
    error = mean(error), upper = error + sd, lower = error - sd)
 %>% ungroup()
err.mat <- rbind(err.mat.rf.summary[, c("num_features", "error",
"method", "lower",
```

```
    "upper")], err.mat.rrf.summary[, c("num_features", "error", "
method", "lower",
    "upper")], err.mat.grrf.summary[, c("num_features", "error",
"method", "lower",
    "upper")])

ggplot(err.mat, aes(x = num_features, y = error, group = method,
colour = method)) +
    geom_line(linetype = "dashed") + geom_point()
```

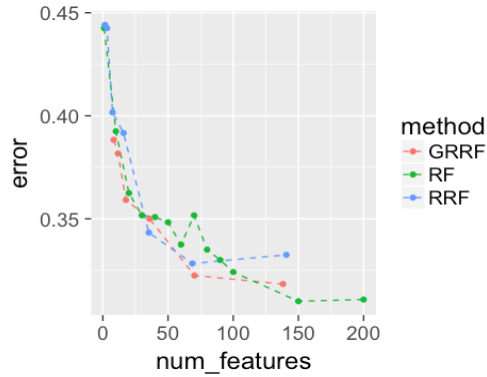

Figure 8.28: The error rates of the RF models with different number of features by RF, RRF, and GRRF

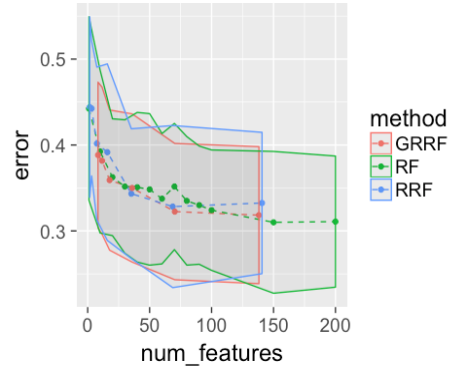

Figure 8.29: The error rates (and their upper and lower bounds) of the RF models with different number of features by RF, RRF, and GRRF

```
ggplot(err.mat, aes(x = num_features, y = error, group = method,
colour = method)) +
    geom_line(linetype = "dashed") + geom_point() + geom_ribbon(d
ata = err.mat,
    aes(ymin = lower, ymax = upper), alpha = 0.05)
```

In addition, a more extensive study on more datasets has found that the RF model with the variables selected from RRF or GRRF can significantly outperform the RF model that uses all the features. This provides evidence that the RRF and GRRF indeed can provide better variable selection results in this data, which is consistent with theoretical arguments illustrated using the exemplary dataset shown in Table 8.3, and the simulation results we have shown in Figures 8.20 – 8.22.

IV. Exercises

Data analysis

1. Find 5 classification datasets from the UCI data repository or R datasets. Use LASSO to select variables. Then, based on the reduced dataset, conduct a detailed analysis using the logistic regression model, SVM, decision tree, random forest, and AdaBoost. Conduct model selection and validation. Use cross-validation to select the best models.
2. Repeat 1, but use random forest to select variables. Compare the final models with the ones based on LASSO.
3. Repeat 1, but use PCA to identify the top PCs to replace the original variables. Compare the final models with the ones based on LASSO.
4. Find 5 regression datasets from the UCI data repository or R datasets. Repeat 1 and 3.

Programming

5. Write your own R script to implement the shooting model. Compare your results with `glm()`.

6. Write your own R script to implement the PCA model. Compare your results with `pca()`.