Homework 4, STAT 542

Wenke Huang

Question 1

1.1 Quick Introduction

In this case, we would like to fit the slided inverse regression which is discussed in the lecture not `SIR`. Here, we are about to choose the numbers of slices $H = 10$ which is a fixed number and will not be tuned here. The code we use to generate the data set has been offered by Professor Zhu. We will use this data set to perform our following analysis.

1.2 Model Fitting

First, we calculate the sample covariance matrix $\hat{\Sigma}_x$ for the data set $\mathbf{X}$. Second, we subtract column means from the $\mathbf{X}$ to make each variable mean 0. Then, we multiply $\hat{\Sigma}_x^{-1/2}$ on the right. Here, we can get ourselves a new matrix $\mathbf{Z}$. Sort the dataset ($\mathbf{Z}$, $\mathbf{Y}$) by the observed Y values. Divide the dataset (according to the sorted values of Y) into H slices as equally as possible. In this case, we can easily find out that each slice will be assigned to 50 (which is equal to 500/10) observations evenly. Within each slice h (h = 1,2,. . . ,10), we compute the sample column mean of Z_h. We also compute the sample column mean $\bar{z}$ of our data set Z. Then, we compute the covariance matrix for the slice means of Z, weighted by the slice sizes: $\mathbf{M} = \frac{\sum_{h=1}^{H} n_h(\bar{z}_h - \bar{z})(\bar{z}_h - \bar{z})^T}{n}$. Finally, we perform PCA on $\mathbf{M}$ and obtain the K largest eigenvectors, $\hat{\alpha}_1$, $\hat{\alpha}_2$,. . . , $\hat{\alpha}_K$. After we get them, we transform them back to $\hat{\beta}_1 = \hat{\Sigma}_x^{-1/2}\hat{\alpha}_1$, $\hat{\beta}_2 = \hat{\Sigma}_x^{-1/2}\hat{\alpha}_2$,. . . , $\hat{\beta}_K = \hat{\Sigma}_x^{-1/2}\hat{\alpha}_K$. In this case, we set K to be 2.

Table 1: Results of both Own and Package Codes

	Dir1.own	Dir2.own	Dir1.dr	Dir2.dr
X1	-0.2500124	0.7665312	0.2558937	0.5957737
X2	-0.6485292	-0.3857784	0.6525792	-0.3007017
X3	-0.2189409	0.2353363	0.2247071	0.2865794
X4	0.6718652	0.4644759	-0.6758045	0.3720583
X5	0.0117874	0.1249510	-0.0050473	0.3366031
X6	-0.0072603	-0.0605469	0.0034366	-0.1914790
X7	-0.0261229	-0.3950544	0.0265353	-0.2234455
X8	-0.0064671	0.1102010	0.0079486	0.1200475
X9	0.0258084	0.2842068	-0.0248198	0.2203536
X10	0.0043018	-0.3361487	-0.0057682	-0.2668637

1.3 Further Test and Conclusion

The above table include results from both our own and `dr()` codes. However, we find that they don't match perfectly. Here, we will use an another method to test our own codes.

First of all, we generate a new set of data by changing line 6 of the code to $n = 50000$. Then, we perform our code on the data with number of slices $H = 10$ and obtain the first two directions $\hat{\beta}_1$ and $\hat{\beta}_2$, which is same with the previous part. In the next step, we perform a canonical correlation analysis (CCA) between the true directions `cbind(b1, b2)` (one 10×2 matrix) and your estimated directions `cbind`($\hat{\beta}_1$, $\hat{\beta}_2$). If our estimated directions form the same column space as the true directions, then the canonical correlations should be very close to 1. The results will be shown below.

Table 2: Results of CCA (n=50000)

	cc1	cc2
Canonical Correlations	0.9999923	0.998986

We find that the canonical correlations are extremely close to 1. We can conclude that our code should be correct. Then, we change n back to 500. This time, we will perform `dr()` and our own personal code on the data set. Then we perform CCA between the first two directions obtained by both two methods.

Table 3: Results of CCA (n=500)

	cc1	cc2
Canonical Correlations	0.9999999	0.945165

The above table shows us that the canonical correlations are close to 1. Hence, we can conclude that our code is correct in this case.

Question 2

2.1 Quick Introduction

In this case, we will perform boosting on the email span dataset data(spam) in the `ElemStatLearn` package. Both exponential and logistic likelihood loss will be discussed, and number of trees and shrinkage factor will be tuned. The possible values of `n.trees` will be `300, 350` and `400` while the possible values of `shrinkage` will be `0.01, 0.1` and `0.99`. 10-folds cross-validation will be used to find out the best tuning parameters and a better model of these two. The possible tuning parameters `interaction.depth` and `n.minobsinnode` are set to be fixed values `1` and `10`, respectively.

2.2 Model Fitting

Boosting produces a sequence of learniers: $F_T(x) = \sum_{t=1}^{T} f_t(x)$. At the t-th iteration, given previously estimated $f_1, f_2, ...f_{t-1}$, we estimate a new function $h(x)$ to minimize the loss: $\min \sum_{i=1}^{n} L(y_i, \sum_{k=1}^{t-1} f_k(x_i) + h(x_i))$. Boosting is an additive model, but its slightly different from generalized additive model, in which each weak learner only involves one variable, and only p number of functions are used and added up. In boosting, each $f_t(x)$ may involve all variables in general, and we may fit a large number (T) of functions.

AdaBoost is a special case of this framework with `Exponential loss` for classification. First, we initiate the weights $w_i^{(1)} = 1/n, i = 1, 2, ..., n$. For $t = 1$ to T, we repeat the following procedures (a) - (d). (a), Fit a classifier $f_t(x) \in \{-1, 1\}$ to the weighted data, with individual weights $w_i^{(t)}$. (b) Compute $\epsilon_t = \sum_i w_i^{(t)} \mathbf{1}\{y_i \neq f_t(x_i)\}$. (c) Compute $\alpha_i = \frac{1}{2} log \frac{1-\epsilon_t}{\epsilon_t}$. (d) Then we update the weights, $w_i^{(t+1)} = \frac{w_i^{(t)}}{Z_t} exp\{-\alpha_t y_i f_t(x_i)\}$. Our final model will be $F_T(x) = \sum_{t=1}^{T} \alpha_t f_t(x)$. The rule of output the classification is: $sign(F_T(x))$.

For logistic likelihood loss, we assume y is the label of data and x is a feature vector. The classification framework can be formalized as: $\arg\min \sum_i L(y_i, f(\mathbf{x}_i))$, where f is a hypothesis function and L is loss function. For logsitic regression, we have the following instantiation: $f(\mathbf{x}) = \beta^T \mathbf{x}$, $L(y, f(\mathbf{x})) = \log(1 + \exp(-yf(\mathbf{x})))$ where $y \in \{\pm 1\}$.

2.3 Comparation and Conclusion

Table 4: Results of Model Fitting

shrinkage.exp	ntrees.exp	Accuracy.exp	shrinkage.log	ntrees.log	Accuracy.log
0.01	300	0.8993641	0.01	300	0.9002309
0.10	300	0.9441388	0.10	300	0.9417494
0.99	300	0.9317503	0.99	300	0.9332706
0.01	350	0.9030588	0.01	350	0.9047942
0.10	350	0.9443557	0.10	350	0.9430523
0.99	350	0.9345750	0.99	350	0.9339261
0.01	400	0.9061000	0.01	400	0.9082734
0.10	400	0.9460939	0.10	400	0.9428340
0.99	400	0.9337064	0.99	400	0.9345769

Table 5: Best Tuning Parameters of Both Methods

n.trees	shrinkage	Loss	Accuracy
400	0.1	Exponential Loss	0.9460939
350	0.1	Logistic Likelihood Loss	0.9430523

The results of both of the two methods are shown above. `.exp` indicates `Exponential Loss` while `.log` indicates `Logistic Likelihood Loss`. It is obvious that the method boosting with `Exponential Loss` is better in this case because its cross-validation accuracy is a little higher than the other one. And its corresponding best tuning parameters of `n.trees` and `shrinkage` are 400 and 0.1, respectively. What's more, there are still some other parameters like `interaction.depth` and `n.minobsinnode` which are set to be fixed values which have been discussed ahead in this case.