CHAPTER6: DIAGNOSIS

RESIDUALS AND HETEROGENEITY

I. Overview

Chapter 5 is about "**Diagnosis**". Diagnosis, in one sense, is to see if the assumptions that determine the theoretical validity of the model fit the empirical characteristics of the data. For example, when we use linear regression model, we use a whole set of subsequent methods such as the t-test and F-test to gain more understanding of the model, while these methods are built on the assumptions such as the normality of the errors. Identification of assumption violation certainly indicates some concerns, limiting the strength of our conclusion, but doesn't mean the model is not useful. The model is still useful, telling part of the truth. Actually, the model, together with the potential gap between the theoretical assumptions and the empirical data characteristics, should be taken as a whole, that jointly form the analytics practice. Many diagnostic tools are developed for maintaining a critical attitude towards the models which are essentially artificial representations/approximations of the reality, yet there is a big difference between being critical and being dismissive. An even more radical assertion was once pointed out in the seminar book[1] that even a model that doesn't fit generates knowledge, revealed not by the failed model but by the misfit of the model as a fact.

II. Residual Analysis in Regression

In this chapter, we take a pragmatism approach to present some of these concepts, by combining the background, theory, and R lab into one section.

Residual analysis: Diagnosis of regression models have been theorized and well articulated in a few monographs. Many interesting concepts have been developed in the literature, such as the **multicollinearity**, **heteroscedasticity**, **cook's distance**, **leverage**, and **Q-Q plot**, to name a few.

Let's use the final regression model we identified in Chapter 2 for an example. The following R code reproduces this final model:

```
AD <- read.csv('AD_bl.csv', header = TRUE)
AD$ID = c(1:dim(AD)[1])
str(AD)

# fit a full-scale model
AD_full <- AD[,c(1:16)]
lm.AD <- lm(MMSCORE ~ ., data = AD_full)
summary(lm.AD)

# Automatic model selection
lm.AD.F <- step(lm.AD, direction="backward", test="F")
```

The returned final model is summarized by calling the function `summary()`.

```
## MMSCORE ~ PTEDUCAT + FDG + AV45 + HippoNV + rs744373 + rs61093
2 +
##     rs3764650 + rs3865444
##
```

[1] *Jaynes, E.T. Probability theory: the logic of science. Cambridge Press, 2003.*

```
##             Df Sum of Sq    RSS    AIC F value    Pr(>F)    
## <none>                   1537.5 581.47                      
## - rs3764650  1     7.513 1545.0 581.99  2.4824  0.115750    
## - rs744373   1    12.119 1549.6 583.53  4.0040  0.045924 *  
## - rs610932   1    14.052 1551.6 584.17  4.6429  0.031652 *  
## - rs3865444  1    21.371 1558.9 586.61  7.0612  0.008125 ** 
## - AV45       1    50.118 1587.6 596.05 16.5591 5.467e-05 ***
## - PTEDUCAT   1    82.478 1620.0 606.49 27.2507 2.610e-07 ***
## - HippoNV    1   118.599 1656.1 617.89 39.1854 8.206e-10 ***
## - FDG        1   143.852 1681.4 625.71 47.5288 1.614e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The R package “ggfortify” provides a nice graphic bundle to show some important diagnostic figures.

```
# Conduct diagnostics of the model
# install.packages("ggfortify")
library("ggfortify")

autoplot(lm.AD.F, which = 1:6, ncol = 3, label.size = 3)
```

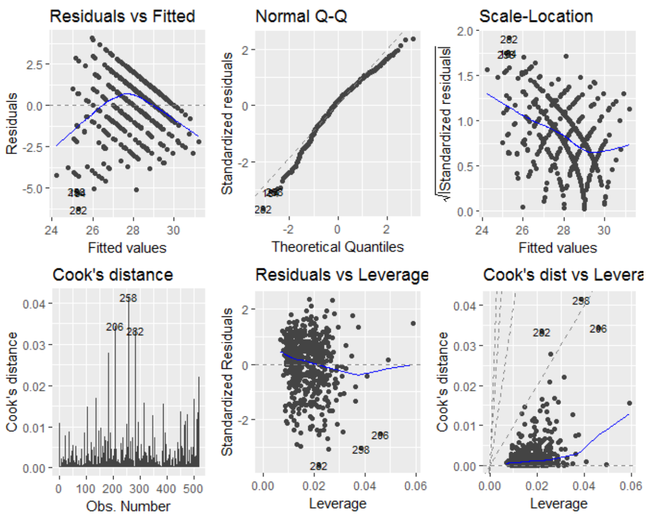

Figure 6.1: Diagnostic figures of regression model on the AD dataset

The diagnostic figures shown in Figure 6.1 are information-rich. One way to interpret them is to put them into a contrast with the way they suppose to be. For example, for the left figure in the first row, which is the scatterplot of the residuals versus fitted values of the outcome variable, it is supposed to show purely random distributions of the data points. In other words, any pattern that shows non-random characteristics, such as the curved relationship between the residuals and fitted values, and the unusual parallel lines of the data points, indicates deviance from the assumptions such as independence of the observations and constancy/homoscedasticity of the variance of the errors.

The Q-Q plot, as the middle figure in the first row, shows violation of the normality assumption of the error term. And some particularly violating data points such as the data points 282 and 256 are labelled.

The Cook's distance shown in the left figure in the second row, shows the influential data points that have larger than average influence on the parameter estimation. The Cook's distance of a data point is built on the idea of how much change will be induced on the estimated parameters if the data point is deleted.

The leverage of a data point, on the other hand, shows the influence of the data point in another way. Mathematically, the leverage of a data point is $\frac{\partial \hat{y}_i}{\partial y_i}$, reflecting how sensitive the prediction on the data point by the model is decided by the observed outcome value y_i. In other words, what data point will result in high leverage value? For data points that are surrounded by many close-by data points, their leverages won't be large, since the impact of removal of them will be compensated by other similar data points in the nearby. Thus, we could infer that the data points that sparsely occupy their neighbor areas will have large leverages. These data points could either be outliers that severely derivate from the linear trend represented by the majority of the data points, or could be valuable data points that align with the linear trend but lack neighbor data points, and thus, changes on their observations will generate a large impact on the predictions on the data points nearby their locations. Thus, it is important to note that, a data point that is influential doesn't necessary imply that it is bad. It only suggests that some more in-depth examination of the data point is needed.

While the information shown in Figure 6.1 is telling, we don't know how bad it is. In other words, we need a baseline version of these figures to establish an expectation so we can compare Figure 6.1 with. To do so, we can simulate a dataset while all the assumptions of the linear regression model are met, to get a sense what these diagnostic figures would look like. The R code in below shows how we can simulate a dataset with 100 samples from the regression model:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \varepsilon,\ \varepsilon \sim N(0,1).$$

```
# For comparison, let's simulate data
# from a model that fits the assumptions
x1 <- rnorm(100, 0, 1)
x2 <- rnorm(100, 0, 1)
beta1 <- 1
beta2 <- 1
mu <- beta1 * x1 + beta2 * x2
y <- rnorm(100, mu, 1)
lm.XY <- lm(y ~ ., data = data.frame(y,x1,x2))
summary(lm.XY)
```

We can see that the fitted model fairly reflects the underlying model.

```
##
## Call:
## lm(formula = y ~ ., data = data.frame(y, x1, x2))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.6475 -0.6630 -0.1171  0.7986  2.5074
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.0366     0.1089   0.336    0.738
## x1            0.9923     0.1124   8.825 4.60e-14 ***
## x2            0.9284     0.1159   8.011 2.55e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.088 on 97 degrees of freedom
## Multiple R-squared:  0.6225, Adjusted R-squared:  0.6147
## F-statistic: 79.98 on 2 and 97 DF,  p-value: < 2.2e-16
```

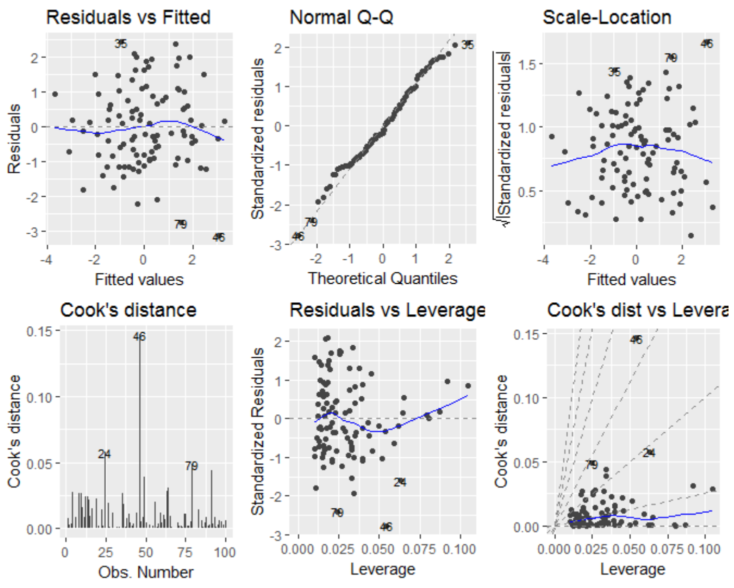

Figure 6.2: Diagnostic figures of regression model on a simulation dataset

Then, we can generate the same set of diagnostic figures. Many interesting contrasts could be observed. For example, from the left figure on the first row in Figure 6.2, we can see that, different from the one in Figure 6.1, now we don't see any significant non-random statistical pattern. The relationship between the residual and fitted values seems to be null. From the QQ-plot, we can also see that the normality assumption is held well. On the other hand, from the cook's distance and the leverage plot, some data points are observed to be influential just as what we can observe from Figure 6.1. As we know that we have simulated the data strictly following the assumptions of the linear regression model, this experiment shows that it is normal to expect some data points exhibiting abnormality according to the cook's distance and the leverage.

```r
# Conduct diagnostics of the model
library("ggfortify")
autoplot(lm.XY, which = 1:6, ncol = 3, label.size = 3)
```

Multicollinearity analysis: The diagnostic figures shown above are all about diagnosis on the data points. This perspective of looking at the dataset point by point (vertically) is the conventional way to define model diagnostics. There is another perspective which is to look at the dataset variable by variable (horizontally). In regression model, the problem of multicollinearity has been well known as a serious problem. Multicollinearity refers to the phenomenon that many predictor variables highly correlate with each other, resulting in great ambiguity in the model parameter estimation due to the ill condition of the matrix $\mathbf{X}^T\mathbf{X}$, i.e., small changes on $\mathbf{X}$ will result in large and unpredictable changes on the inverse matrix $\mathbf{X}^T\mathbf{X}$, which will eventually result in great instability of the parameter estimation in $\widehat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\boldsymbol{Y}$.

We can do a simple analysis to study this problem of multicollinearity. Consider a system that generates the observation following the relationships shown in below:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p + \varepsilon,\ \ \varepsilon \sim N(0, \sigma_\varepsilon^2),$$

$$x_1 = 2x_2 + \epsilon,\ \ \epsilon \sim N(0, 0.1\sigma_\varepsilon^2)$$

The system has the symptom of multilinearity as two of the variables are highly correlated. Thus, theoretically, we could value the regression model that is shown in above as the ground truth model equally as we value the following models:

$$y = \beta_0 + (2\beta_1 + \beta_2)x_2 + \beta_3 x_3 \ldots + \beta_p x_p,$$
$$y = \beta_0 + (\beta_1 + 0.5\beta_2)x_1 + \beta_3 x_3 + \cdots + \beta_p x_p,$$

$$y = \beta_0 + 1000x_1 + (\beta_2 + \beta_1 - 2000)x_2 + \beta_3 x_3 + \cdots + \beta_p x_p,$$

Thus, the problem of multilinearity creates this inherent ambiguity of the models that could be taken as faithful representation of how the data was generated. Consequently, it makes no sense that an estimation method, essentially as a reverse-engineering approach, that could recover the truth while the truth itself is ambivalent.

There are some methods that we can use to diagnose for multilinearity. As it is a condition that many variables are highly correlated with each other, we may present the correlations among the predictor variables. The R package "corrplot" is a package that has been widely used for visualizing correlation matrix.

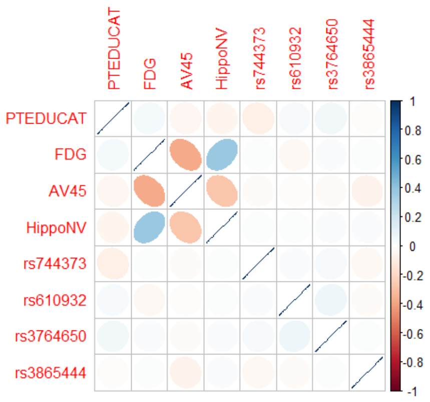

Figure 6.3: Correlations of the predictors in the regression model of `MMSCORE`

The following R code shows how to use "`corrplot`" to visualize the correlations among the predictors in the regression model we have built in Chapter 2 for predicting `MMSCORE`. Result is shown in Figure 6.3.

```
# Extract the covariance matrix of the regression parameters
Sigma = vcov(AD)
```

```
# Visualize the correlation matrix of the estimated regression pa
rameters
# install.packages("corrplot")
library(corrplot)

corrplot(cov2cor(Sigma), method="ellipse")
```

From Figure 6.3 we could observe that there is significant correlations between the variables, `FDG`, `AV45`, and `HippoNV`, indicating a concern for multicollinearity. On the other hand, it seems that the correlations are only moderate, and not all the variables are densely correlated with each other.

We can further visualize the correlation matrix of the estimated regression parameter:

```
Sigma = vcov(lm.AD.F)

corrplot(cov2cor(Sigma), method="ellipse")
```

Then we can observe a similar pattern in Figure 6.4 as shown in Figure 6.3.

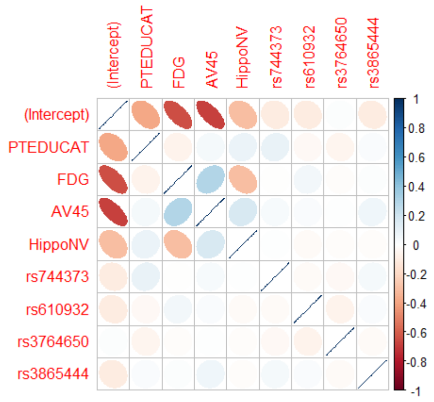

Figure 6.4: Correlations of the estimated parameters in the regression model of `MMSCORE`

The concern with multicollinearity, as we have discussed and illustrated using an analysis, is that it may result in unreliable model estimations, due to the inherent ambiguity and instability in the numerical operations in least square estimation. The `corrplot` could help to visually check the data for multicollinearity, but it could not answer the question whether we should worry about the model we have built. To answer this question, we could further use Bootstrap to introduce perturbation into the data and see if the models fitted on different bootstrapped samples will change. Recall that we have done this in Chapter 4, we could draw the conclusion that the multicollinearity issue is not severe here in the AD dataset.

Ranking of variable: A related topic to the multicollinearity problem is the ranking of features. For example, the R package "`leaps`" implements the exhaustive evaluation of all subsets regression, i.e., try every combination of variables with a minimum number of features in the regression model. The results will show which models achieve highest R-squared value, and which variables frequently appear on these models.

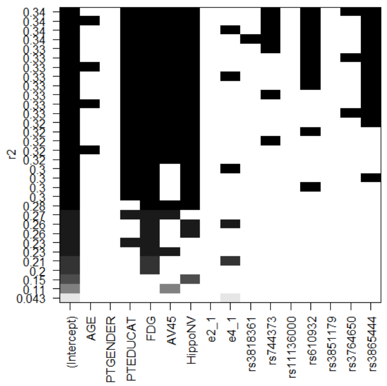

Figure 6.5: Regression models ranked by their R-squared and the constitutive features

The following R code implements this method on the AD dataset. Result is shown in Figure 6.5.

```
# Evaluate the variable importance by all subsets regression
# install.packages("leaps")
library(leaps)

leaps<-regsubsets(MMSCORE ~ ., data = AD,nbest=4)
# view results
summary(leaps)

# plot a table of models showing variables in each model.
# models are ordered by the selection statistic.
plot(leaps,scale="r2")
```

From Figure 6.5 we could observe that, the variables, `PTEDUCAT`, `FDG`, `AV45`, and `HippoNV`, are most important features. And `rs3865444` shows moderate significance. Other variables show less significance.

III. Diagnosis in Random Forests

Random forests make few assumptions about the data. It handles mixed categorical and numerical variables, nonlinearities, and variable interactions in a more automatic way than data models such as linear regression. However, random forests are complex and consist of multiple weak trees that are hard to interpret. Cross-valuation error or **out-of-bag (OOB)** error can be used to evaluate the random forests' model accuracy. However, to be able to trust random forests, it is desirable to extract interpretable insights from random forests. Here we introduce a few ways of diagnosis for random forest.

Variable Importance: Each variable's usefulness in predicting the outcome variable can be measured by the variable importance scores from random forests. There are two types of importance scores. The first one is the total decrease of node impurity across all tree nodes that are split by a variable (retrospectively). The second is measured by the accuracy decrease by permuting the variable (proactively).

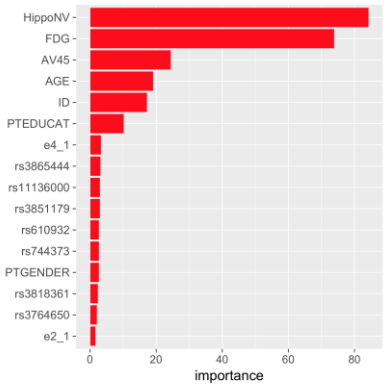

Figure 6.6: Important scores of variables in random forest

Here we plot the first type of importance score, the impurity gain importance score, from the random forest model built for the AD data. Result is shown in Figure 6.6.

```
library("RWeka")
library("randomForest")
library("RRF")
library("inTrees")
library("ggplot2")

path <- "../../data/AD_bl.csv"
data <- read.csv(path, header = TRUE)

target_indx <- which(colnames(data) == "DX_bl")
rm_indx <- which(colnames(data) %in% c("DX_bl", "TOTAL13", "MMSCO
RE"))
rf <- randomForest(data[, -rm_indx], as.factor(data[, target_ind
x]))
imp <- as.data.frame(rf$importance)
colnames(imp)[colnames(imp) == "MeanDecreaseGini"] <- "importance
"
imp <- imp[order(imp$importance, decreasing = FALSE), , drop = FA
LSE]
imp$feature <- rownames(imp)
imp$feature <- factor(imp$feature, levels = as.character(imp$feat
ure))
theme_set(theme_gray(base_size = 18))
ggplot(data = imp, aes(x = feature, y = importance)) + geom_bar(s
tat = "identity",
    aes(factor(feature)), fill = "red") + theme(axis.title.y = el
ement_blank(),
    axis.text.y = element_text(hjust = 1, size = 15)) + coord_fli
p()
```

Not a surprise, we can see from Figure 6.6 that the variables `HippoNV`, `FDG`, `AV45`, and `AGE`, are most important features in the random forest model. Interestingly, we can also see that the feature "`ID`" has the fourth largest importance score. In normal sense, this is a nuisance feature that is supposed to be random assignments of IDs to subjects. Thus, we might suspect that random forest is too powerful in extracting nonlinear patterns from data, thus it is tricked by noises in a dataset, a trade-off that we could overcome by more careful model selection and validation using cross-validation and in-depth analysis.

But on the other hand, this happens in many practical situations that some supposedly random assignments by humans are actually not the same as pure random noise, but rather encode some systematical patterns. As Prof. R.A. Fisher said, who is a pioneer in design of experiments (DOE) and modern statistics, "if one tries to think of numbers at random, one thinks of numbers very far from at random[1]". In addition, out-of-bag error doesn't reduce significantly when the variable `ID` is removed. This may indicate that the creation of the `ID` for the subjects probably contained certain information about the subjects.

In some other applications, we also find that, if we create a binary variable to indicate the missing data instances in a feature, sometimes this binary indicatory variable could be significant. Which means, the missing data itself as a fact is also informative to predict an outcome! This is not uncommon in healthcare applications. For instance, when a patient's condition is severe, this patient may lack measurements of many clinical variables. Thus, the missing values of these variables provide valuable

[1] Fisher, R.A. Cigarettes, cancer, and statistics. *The Centennial Review of Arts & Science*, 1958.

information in predicting if the patient's condition is severe. This issue is also referred as variable **leakage** in machine learing.

Partial dependency plot: Variable importance scores indicate whether a variable is informative in predicting the outcome variable, but do not provide information about how the outcome variable is influenced by the variables. Partial dependency plot can be used to visualize the relationship between the variables of interest and the outcome variable, averaged on other variables. For the AD data, we apply the partial dependency plots to the top two important variables. It is clear that the relationships between the outcome variable with both predictor variables are significant. And we could also see the orientation of both relationships, i.e., the larger the `HippoNV` or `FDG`, the more likely that the subjects belong to normal (the class "normal" is coded as -1).

```
randomForest::partialPlot(rf, data, HippoNV, "1")
randomForest::partialPlot(rf, data, FDG, "1")
```

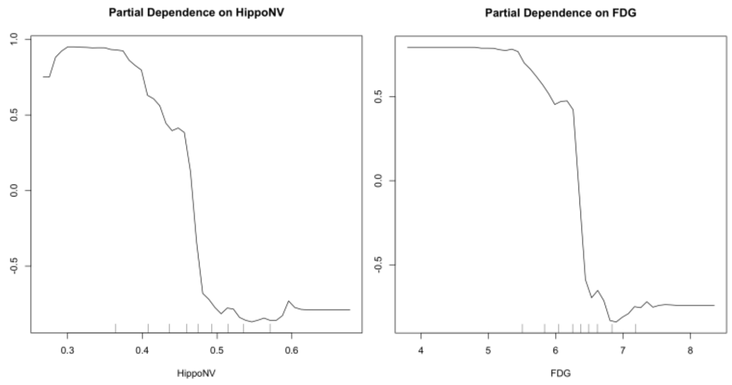

Figure 6.7: Partial dependency plots of variables in random forest

inTrees

Partial dependence plots provide how predictor variables interact with the outcome variable, while this interaction effect is averaged on other variables. However, it is difficult to visualize the synergistic effect of multiple variables on the outcome variable, and a more quantitative approach can be desirable at times. The inTrees framework can be used for this purpose, which will be discussed in detail in Chatper 10. In the framework of inTrees, rules can be extracted, cleaned, and summarized from random forests.

```
treeList <- RF2List(rf)  # transform rf object to an inTrees' for
mat
exec <- extractRules(treeList, data[, -rm_indx])  # R-executable
conditions

## 3695 rules (length<=6) were extracted from the first 100 trees.

class <- paste0("class_", as.character(data[, target_indx]))
rules <- getRuleMetric(exec, data[, -target_indx], class)
rules <- pruneRule(rules, data[, -target_indx], class)
rules <- selectRuleRRF(rules, data[, -target_indx], class)
rules <- presentRules(rules, colnames(data[, -target_indx]))
```

Here are the rules from inTrees applied to the AD data. `len` indicates the number of variable-value pairs in the condition of a rule, `freq` is the percentage of instances satisfying the condition, and `err` is the error rate of the rule. Without the need to choose which variables we would like to study, the selected rules from inTrees indicate all the important variable interactions it could identify, i.e., in the generated rules shown belown, it shows how `HippoNV` and `FDG` interact with the outcome variable.

```
##       len freq    err
##  [1,] "2" "0.3"   "0.00600000000000001"
##  [2,] "2" "0.503" "0.115"
##  [3,] "3" "0.464" "0.125"
##  [4,] "2" "0.373" "0.114"
##  [5,] "2" "0.335" "0.04"
##  [6,] "3" "0.286" "0.0679999999999999"
##  [7,] "3" "0.114" "0.0679999999999999"
##  [8,] "3" "0.209" "0.074"
##  [9,] "3" "0.176" "0.088"
## [10,] "4" "0.023" "0.333"
## [11,] "4" "0.017" "0"
## [12,] "4" "0.11"  "0.035"
## [13,] "4" "0.031" "0.125"
## [14,] "3" "0.099" "0.118"
##       condition
##  [1,] "FDG<=6.35981 & HippoNV<=0.47428125"

##  [2,] "FDG>6.323505 & HippoNV>0.401237706"

##  [3,] "PTEDUCAT>12.5 & AV45<=1.5079 & HippoNV>0.463772954"

##  [4,] "FDG<=6.464415 & HippoNV<=0.4766025375"

##  [5,] "FDG>6.35679 & HippoNV>0.4764150235"

##  [6,] "FDG>6.29287 & HippoNV>0.406667579 & rs3851179>0.5"

##  [7,] "AV45>1.17371 & HippoNV<=0.4713683765 & rs3851179<=0.5"

##  [8,] "FDG>5.69764 & HippoNV>0.4784356305 & rs3865444<=0.5"

##  [9,] "FDG>6.513695 & AV45>1.011585 & e4_1<=0.5"

## [10,] "FDG<=6.30134 & AV45<=1.25324 & HippoNV>0.507448786 & e4
_1>0.5"
## [11,] "PTEDUCAT<=12.5 & AV45>1.103075 & AV45<=1.1183375 & e4_1
<=0.5"
## [12,] "AV45>1.057595 & AV45<=1.740035 & HippoNV<=0.421919192 &
 rs744373<=0.5"
## [13,] "AGE<=69.55 & FDG<=6.30834 & rs744373>0.5 & rs3865444>0.
5"
## [14,] "HippoNV<=0.461434739 & rs744373<=0.5 & rs3851179<=0.5"

##       pred      impRRF
##  [1,] "class_1" "1"
##  [2,] "class_0" "0.158820055688818"
##  [3,] "class_0" "0.0882618347383523"
##  [4,] "class_1" "0.073427455249915"
##  [5,] "class_0" "0.0687467199669404"
##  [6,] "class_0" "0.0557289471858664"
##  [7,] "class_1" "0.030470587008693"
##  [8,] "class_0" "0.0255667487231907"
##  [9,] "class_0" "0.019899126787755"
## [10,] "class_1" "0.0179324928636556"
## [11,] "class_1" "0.0142529598174707"
## [12,] "class_1" "0.0133909560650111"
## [13,] "class_1" "0.0106427916981619"
## [14,] "class_1" "0.0101981643861376"
```

Residual analysis

For problems that have a continuous outcome variable, one can apply residual analysis to random forests through the "`plotmo`" R package. Here we perform residual analysis to the AD data where the variable `AGE` is used as the outcome variable. First, we plot the residual vs. fitted figure as shown in Figure 6.8. If the model fits the data well, the data points should spread around the horizontal line (ideally residual = 0). However, in Figure 6.8, there is a linear pattern between the fitted values and residuals. This indicates that the random forest model missed some linear relationship in the AD dataset.

```r
require(randomForest)
require(plotmo)
set.seed(1)
path <- "../../data/AD_hd.csv"
data <- read.csv(path, header = TRUE)
target <- data$AGE
rm_indx <- which(colnames(data) %in% c("AGE", "ID", "TOTAL13", "M
MSCORE"))
X <- data[, -rm_indx]
rf.mod <- randomForest(X, target)
plotres(rf.mod, which = 3)
```

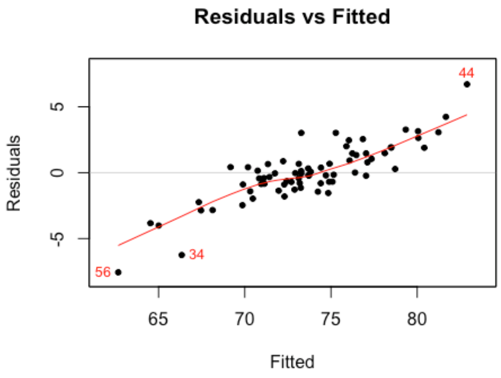

Figure 6.8: Residuals versus fitted in the random forest model

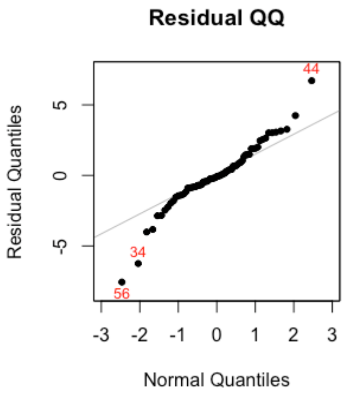

Figure 6.9: The Q-Q plot of residuals of the random forest model

Next, we plot the Q-Q plot that can be interpreted pretty much in the same way as in Figure 6.1. If the random forests fit the data well, the residuals should be pure noise, such that a straight line is expected. However, it can be seen that the residuals deviate from the straight line.

```
plotres(rf.mod, which = 4)
```

Both two residual analysis figures show that the random forest model has underfitting problems, particularly at the two ends of the prediction spectrum. This is expected as the learning boundary of random forests are parallel to the axis. To better illustrate this, we simulate a dataset where the outcome variable has a linear relationship with one single variable. Random forests are applied to the simulated dataset. From the residual vs. fitted Figure as shown in Figure 6.10, we can see that there are three points substantially deviated from the horizontal line and are colored in red. From the Q-Q plot, it is clear that severe derivations happen at the two ends.

```
require(ggplot2)
set.seed(1)
X <- data.frame(X1 = runif(30, min = -1, max = 1))
target <- 0.5 * X$X1  # + 0.5 * X$X2
rf <- randomForest(X, target)
plotres(rf, which = 3)
```

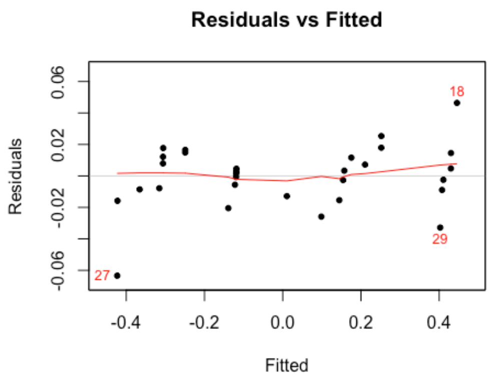

Figure 6.10: Residuals versus fitted in the random forest model fitted on the simulated dataset

```r
plotres(rf, which = 4)
```

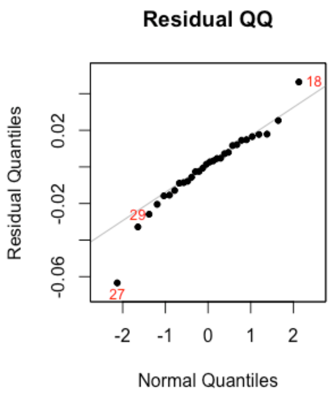

Figure 6.11: The Q-Q plot of residuals of the random forest model fitted on the simulated dataset

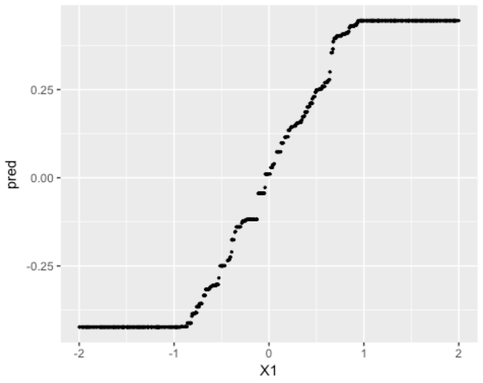

Figure 6.12: Underfitting patterns of random forest to capture linear relationships in dataset

Now we simulate a testing dataset with more data points, and use the trained random forests for prediction. The learning boundary is plotted in Figure 6.12. It can be seen that, at certain intervals, the predicted values remain constant. This is particularly clear at the two ends of the predictor variable, where the predictions have a lower bound and upper bound. This is different from linear regression where the prediction can further increase towards infinitely. This shows that the random forest model can have underfitting problem if there are linear patterns in the dataset.

```
testing <- data.frame(X1 = runif(1000, min = -2, max = 2))
target <- 0.5 * testing$X1  # + 0.5 * X$X2
pred <- predict(rf, testing, type = "response")
pred.data <- cbind(testing, target, pred)
ggplot(pred.data, aes(x = X1, y = pred)) + geom_point(size = 0.5)
```

IV. Clustering

IV.1 Rationale and Formulation

The residual analysis methods mentioned above have implicitly assumed that, the lack of fit of the model to the data is probably resulted from some outliers in the data that are quite different from a majority of the data. In many applications, the outliers are sparse, randomly distributed, and form no structure. But in some applications, you may find that the implied structure with a majority and a few outliers doesn't apply to the dataset. Rather, it is possible that there are a few majorities that make the dataset heterogeneous.

Thus, while it is not a usual habit in an analytics book to put clustering algorithm together with residual analysis, here, we highlight the utility of clustering method for better understanding of the structure embedded in data to build better prediction models.

Let's start with the Gaussian mixture model (GMM), that has been one of the most popular clustering model. GMM assumes that the data come from not just one distribution but a few. As shown in Figure 6.13, the data is sampled from a mix of 4 distributions.

```r
# Simulate a clustering structure
X <- c(rnorm(200, 0, 1), rnorm(200, 10,2), rnorm(200,20,1), rnorm
(200,40, 2))
Y <- c(rnorm(800, 0, 1))
plot(X,Y, ylim = c(-5, 5), pch = 19, col = "gray25")
```

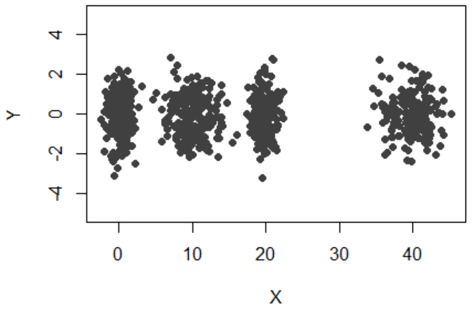

Figure 6.13: A mixture of four Gaussian distributions

This leads to the following formulation of data-generating mechanism. Suppose that there are M distributions mixed together. For each data point $\boldsymbol{x}_n$, the probability that it comes from the mth distribution is denoted as π_m, while $\sum_{m=1}^{M} \pi_m = 1$. In GMM, we assume that all the distributions are Gaussian distributions, i.e., such that we denote the mth distribution as $N(\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)$. The task of GMM is to learn the unknown parameters of the distributions $\{\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m, m = 1,2,\dots,M\}$ and the probability vector $\boldsymbol{\pi}$ that includes the elements $\{\pi_m, m = 1,2,\dots,M\}$. For simplicity in the presentation, let's use $\boldsymbol{\Theta}$ to denote all these parameters.

IV.2 Theory/Method

To learn these parameters from data, first, we need to derive the likelihood function. Frist, we realize that, if we have known which distribution the data point $\boldsymbol{x}_n$ was sampled, it would be straightforward to derive the likelihood function. Following this idea, we invent a binary indicator variable, denoted as z_{nm}, while $z_{nm} = 1$ indicates that $\boldsymbol{x}_n$ was sampled from the mth distribution. Then, the complete log-likelihood function is:

$$l(\boldsymbol{\Theta}) = \log \prod_{n=1}^{N} p(\boldsymbol{x}_n | z_{nm} = 1; \boldsymbol{\Theta}),$$
$$= \log \prod_{n=1}^{N} p(\boldsymbol{x}_n, z_{nm} | \boldsymbol{\Theta}),$$
$$= \log \prod_{n=1}^{N} \prod_{m=1}^{M} [p(\boldsymbol{x}_n | z_{nm} = 1, \boldsymbol{\Theta}) p(z_{nm} = 1)]^{z_{nm}},$$
$$= \sum_{n=1}^{N} \sum_{m=1}^{M} [z_{nm} \log p(\boldsymbol{x}_n | z_{nm} = 1, \boldsymbol{\Theta}) + z_{nm} \log \pi_m].$$

Meanwhile, we can derive that

$$p(\boldsymbol{x}_n | z_{nm} = 1; \boldsymbol{\Theta}) = (2\pi)^{-p/2} |\boldsymbol{\Sigma}_m|^{-1/2} \exp\left\{-\frac{1}{2}(\boldsymbol{x}_n - \boldsymbol{\mu}_m)^T \boldsymbol{\Sigma}_m^{-1} (\boldsymbol{x}_n - \boldsymbol{\mu}_m)\right\}.$$

Thus,

$$l(\boldsymbol{\Theta}) = \sum_{n=1}^{N} \sum_{m=1}^{M} \left[z_{nm} \log\left((2\pi)^{-p/2} |\boldsymbol{\Sigma}_m|^{-1/2} \exp\left\{-\frac{1}{2}(\boldsymbol{x}_n - \boldsymbol{\mu}_m)^T \boldsymbol{\Sigma}_m^{-1} (\boldsymbol{x}_n - \boldsymbol{\mu}_m)\right\}\right) + z_{nm} \log \pi_m\right].$$

To optimize for $\boldsymbol{\Theta}$, we need to overcome the challenge that z_{nm}s are latent and unknown. Here, an intuitive proposal could be:

1. Even we don't know z_{nm}, but we can estimate it if we have known $\boldsymbol{\Theta}$. For instance, it is easy to know that $p(z_{nm} = 1 | \mathbf{X}, \boldsymbol{\Theta}) =$

$\frac{p(x_n|z_{nm}=1,\mathbf{\Theta})\pi_m}{\sum_{k=1}^{M} p(x_n|z_{nk}=1,\mathbf{\Theta})\pi_k}$. Thus, given $\mathbf{\Theta}$, the best estimate of z_{nm} could be the expectation of z_{nm} as $\langle z_{nm}\rangle_{p(z_{nm}|\mathbf{X},\mathbf{\Theta})} = 1 \cdot p(z_{nm} = 1|\mathbf{X},\mathbf{\Theta}) + 0 \cdot p(z_{nm} = 0|\mathbf{X},\mathbf{\Theta}) = \frac{p(x_n|z_{nm}=1,\mathbf{\Theta})\pi_m}{\sum_{k=1}^{M} p(x_n|z_{nk}=1,\mathbf{\Theta})\pi_k}$.

2. We can fill in $l(\mathbf{\Theta})$ with the estimated z_{nm} and optimize it to update $\mathbf{\Theta}$. Feed this updated back to Step 1 and repeat the iterations, until all the parameters in the iterations don't change significantly.

To do step 2, we need to derive the estimated $l(\mathbf{\Theta})$, which is denoted as $\langle l(\mathbf{\Theta})\rangle_{p(\mathbf{Z}|\mathbf{X},\mathbf{\Theta})}$. It can be seen that

$$\langle l(\mathbf{\Theta})\rangle_{p(\mathbf{Z}|\mathbf{X},\mathbf{\Theta})} = \sum_{n=1}^{N}\sum_{m=1}^{M}\left[\langle z_{nm}\rangle_{p(z_{nm}|\mathbf{X},\mathbf{\Theta})} \log p(\boldsymbol{x}_n|z_{nm} = 1,\mathbf{\Theta}) + \langle z_{nm}\rangle_{p(z_{nm}|\mathbf{X},\mathbf{\Theta})} \log \pi_m\right].$$

To optimize for the parameters $\{\boldsymbol{\mu}_m, \mathbf{\Sigma}_m, m = 1,2,\dots,M\}$, we take derivatives of $\langle l(\mathbf{\Theta})\rangle_{p(\mathbf{Z}|\mathbf{X},\mathbf{\Theta})}$ regarding to these parameters and put them equal to zero:

$$\frac{\partial\langle l(\mathbf{\Theta})\rangle_{p(\mathbf{Z}|\mathbf{X},\mathbf{\Theta})}}{\partial\boldsymbol{\mu}_m} = \sum_{n=1}^{N}\langle z_{nm}\rangle_{p(z_{nm}|\mathbf{X},\mathbf{\Theta})}\frac{\partial \log p(x_n|z_{nm}=1,\mathbf{\Theta})}{\partial\boldsymbol{\mu}_m} = \mathbf{0}.$$

We can derive that

$$\frac{\partial \log p(x_n|z_{nm}=1,\mathbf{\Theta})}{\partial\boldsymbol{\mu}_m} = \frac{\partial \log\left((2\pi)^{-p/2}|\mathbf{\Sigma}_m|^{-1/2}\exp\left\{-\frac{1}{2}(x_n-\boldsymbol{\mu}_m)^T\mathbf{\Sigma}_m^{-1}(x_n-\boldsymbol{\mu}_m)\right\}\right)}{\partial\boldsymbol{\mu}_m} = -\frac{1}{2}\frac{\partial(x_n-\boldsymbol{\mu}_m)^T\mathbf{\Sigma}_m^{-1}(x_n-\boldsymbol{\mu}_m)}{\partial\boldsymbol{\mu}_m} = (\boldsymbol{x}_n - \boldsymbol{\mu}_m)^T\mathbf{\Sigma}_m^{-1}.$$

Thus, putting these together we can have

$$\sum_{n=1}^{N}\langle z_{nm}\rangle_{p(z_{nm}|\mathbf{X},\mathbf{\Theta})}(\boldsymbol{x}_n - \boldsymbol{\mu}_m)^T\mathbf{\Sigma}_m^{-1} = \mathbf{0}.$$

This gives us the equation to estimate $\boldsymbol{\mu}_m$ as

$$\boldsymbol{\mu}_m = \frac{\sum_{n=1}^{N}\langle z_{nm}\rangle_{p(z_{nm}|\mathbf{X},\mathbf{\Theta})}x_n}{\sum_{n=1}^{N}\langle z_{nm}\rangle_{p(z_{nm}|\mathbf{X},\mathbf{\Theta})}}.$$

Similarly, we take derivatives of $\langle l(\mathbf{\Theta})\rangle_{p(\mathbf{Z}|\mathbf{X},\mathbf{\Theta})}$ regarding $\mathbf{\Sigma}_m$ and put them equal to zero:

$$\frac{\partial\langle l(\mathbf{\Theta})\rangle_{p(\mathbf{Z}|\mathbf{X},\mathbf{\Theta})}}{\partial\mathbf{\Sigma}_m} = \sum_{n=1}^{N}\langle z_{nm}\rangle_{p(z_{nm}|\mathbf{X},\mathbf{\Theta})}\frac{\partial \log p(x_n|z_{nm}=1,\mathbf{\Theta})}{\partial\mathbf{\Sigma}_m} = \mathbf{0}.$$

We can derive that

$$\frac{\partial \log p(\boldsymbol{x}_n|z_{nm}=1,\boldsymbol{\Theta})}{\partial \boldsymbol{\Sigma}_m} =$$

$$\frac{\partial \log\left((2\pi)^{-p/2}|\boldsymbol{\Sigma}_m|^{-1/2}\exp\left\{-\frac{1}{2}(\boldsymbol{x}_n-\boldsymbol{\mu}_m)^T\boldsymbol{\Sigma}_m^{-1}(\boldsymbol{x}_n-\boldsymbol{\mu}_m)\right\}\right)}{\partial \boldsymbol{\Sigma}_m} =$$

$$\frac{1}{2}\frac{\partial\left\{|\boldsymbol{\Sigma}_m|^{-1/2}-(\boldsymbol{x}_n-\boldsymbol{\mu}_m)^T\boldsymbol{\Sigma}_m^{-1}(\boldsymbol{x}_n-\boldsymbol{\mu}_m)\right\}}{\partial \boldsymbol{\Sigma}_m} = \frac{1}{2}[\boldsymbol{\Sigma}_m-(\boldsymbol{x}_n-\boldsymbol{\mu}_m)(\boldsymbol{x}_n-\boldsymbol{\mu}_m)^T].$$

Thus, putting these together we can have

$$\sum_{n=1}^{N}\langle z_{nm}\rangle_{p(z_{nm}|\mathbf{X},\boldsymbol{\Theta})}[\boldsymbol{\Sigma}_m-(\boldsymbol{x}_n-\boldsymbol{\mu}_m)(\boldsymbol{x}_n-\boldsymbol{\mu}_m)^T]=\mathbf{0}.$$

This gives us the equation to estimate $\boldsymbol{\Sigma}_m$ as

$$\boldsymbol{\Sigma}_m=\frac{\sum_{n=1}^{N}\langle z_{nm}\rangle_{p(z_{nm}|\mathbf{X},\boldsymbol{\Theta})}(\boldsymbol{x}_n-\boldsymbol{\mu}_m)(\boldsymbol{x}_n-\boldsymbol{\mu}_m)^T}{\sum_{n=1}^{N}\langle z_{nm}\rangle_{p(z_{nm}|\mathbf{X},\boldsymbol{\Theta})}}.$$

Lastly, in order to optimize for $\{\pi_m, m=1,2,\ldots,M\}$, we face with the problem that $\sum_{m=1}^{M}\pi_m=1$. To address this, we introduce the Lagrange multiplier λ and optimize for

$$\langle l(\boldsymbol{\Theta})\rangle_{p(\mathbf{Z}|\mathbf{X},\boldsymbol{\Theta})}-\lambda\left(\sum_{m=1}^{M}\pi_m-1\right).$$

We take derivatives of it regarding π_m and put them equal to zero:

$$\frac{\partial\left[\langle l(\boldsymbol{\Theta})\rangle_{p(\mathbf{Z}|\mathbf{X},\boldsymbol{\Theta})}-\lambda\left(\sum_{m=1}^{M}\pi_m-1\right)\right]}{\partial\pi_m}=\frac{\partial\langle l(\boldsymbol{\Theta})\rangle_{p(\mathbf{Z}|\mathbf{X},\boldsymbol{\Theta})}}{\partial\pi_m}-\lambda=0.$$

It is known that

$$\frac{\partial\langle l(\boldsymbol{\Theta})\rangle_{p(\mathbf{Z}|\mathbf{X},\boldsymbol{\Theta})}}{\partial\pi_m}=\frac{1}{\pi_m}\sum_{n=1}^{N}\langle z_{nm}\rangle_{p(z_{nm}|\mathbf{X},\boldsymbol{\Theta})}.$$

Thus, for $m=1,2,\ldots,M$ we arrive at

$$\sum_{n=1}^{N}\langle z_{nm}\rangle_{p(z_{nm}|\mathbf{X},\boldsymbol{\Theta})}-\lambda\pi_m=0.$$

Adding these M equations together, we have

$$\sum_{m=1}^{M}\sum_{n=1}^{N}\langle z_{nm}\rangle_{p(z_{nm}|\mathbf{X},\boldsymbol{\Theta})}-\lambda\sum_{m=1}^{M}\pi_m=0.$$

Since $\sum_{m=1}^{M}\pi_m=1$, we can get that

$$\lambda=\sum_{m=1}^{M}\sum_{n=1}^{N}\langle z_{nm}\rangle_{p(z_{nm}|\mathbf{X},\boldsymbol{\Theta})}=N.$$

Thus,

$$\pi_m=\frac{\sum_{n=1}^{N}\langle z_{nm}\rangle_{p(z_{nm}|\mathbf{X},\boldsymbol{\Theta})}}{N}.$$

IV.3 R Lab

The R package “`Mclust`” could be used to implement the GMM model while the underlying algorithm is the EM algorithm. Again, using the simulated data with four clusters, the following R code is to identify clusters.

```
# use GMM to identify the clusters
require(mclust)

XY.clust <- Mclust(data.frame(X,Y))
summary(XY.clust)

plot(XY.clust)
```

Then, we can obtain:

```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust VVI (diagonal, varying volume and shape) model with 4 c
omponents:
##
##  log.likelihood   n df       BIC       ICL
##        -3666.07 800 19 -7459.147 -7459.539
##
## Clustering table:
##   1   2   3   4
## 199 201 200 200
```

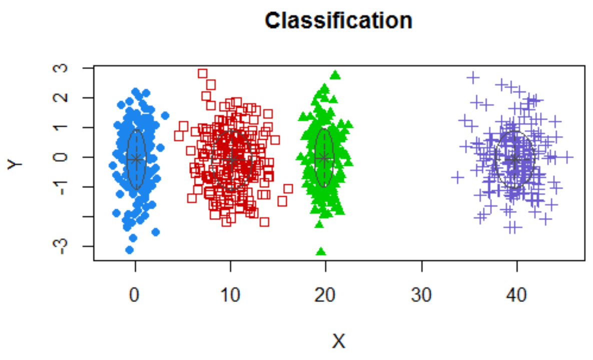

Figure 6.14: Clustering results of the simulated data

Note that, here, we didn't specify how many clusters should `Mclust` find. It seems that, by using model selection criteria such as `BIC` which balances model fit and model complexity (here refers to the number of clusters), mclust correctly identified the four clusters. It can also been seen that, for each cluster, the data points are almost 200.

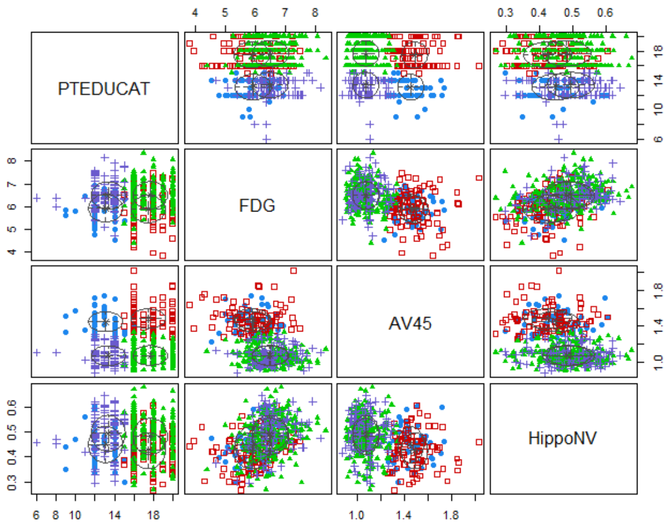

Figure 6.15: Clustering results of the AD data

Now let's implement GMM on the AD data using `Mclust`. Result is shown in Figure 6.15.

```
# install.packages("mclust")
require(mclust)
AD.Mclust <- Mclust(AD[,c(3,4,5,6,10,12,14,15)])
summary(AD.Mclust)

AD.Mclust$data = AD.Mclust$data[,c(1:4)]
# plot(AD.Mclust)

## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
```

```
## Mclust EEI (diagonal, equal volume and shape) model with 4 com
ponents:
##
##  log.likelihood   n df       BIC       ICL
##       -3235.874 517 43 -6740.414 -6899.077
##
## Clustering table:
##   1   2   3   4
##  43 253  92 129
```

Interestingly, it seems that in the AD data, four clusters are identified as well. And results are shown in in Figure 6.15. It seems a reasonable clustering result, although the boundaries between clusters are not as distinct as the boundaries in Figure 6.14. In real applications, particularly for those applications for which we haven't known enough, clustering is more like an exploration tool. It could generate suggestive results, but probably not confirmative conclusions.

IV.4 Remarks

Clustering-based prediction models: As the existence of clustering structure in a dataset violates the assumption of many prediction models such as the regression model that assume the data come from a homogeneous distribution, evidences in the literature and practices have shown that it will increase performance of prediction if we could identify the clusters and build prediction models separately for the clusters. In the literature, some algorithms have been developed to integrate clustering and prediction models jointly. For example, the Treed Regression method[1] is one of the earlier examples that propose to build a tree to stratify the dataset and create regression models on the leaves. Similarly, the logistic model trees model[2] also builds the tree to allocate data points into different leaves and build different logistic regression model for each leaf. Motivated

[1] *Alexander, W. and Grimshaw, S. Treed regression. Journal of computational and graphical statistics, 1996.*

[2] *Landwehr, N., Hall, M. and Frank, E. Logistic model trees, Machine learning, 2004.*

by this line of thoughts, more models have been developed with different combination of tree models and prediction models (or other types of statistical models) on the leaves[1][2].

The EM algorithm: The iterative two-step algorithm is actually the idea of the Expectation-Maximization (EM) algorithm. The EM algorithm is commonly used to solve for this type of problems that involve latent variables. The idea of the EM algorithm in GMM is to follow the iterative two-steps as shown in below:

1. The E-step: Derive the posterior distribution of $\mathbf{Z}$ as $p(\mathbf{Z}|\mathbf{X}, \mathbf{\Theta})$. Calculate the expectation of $l(\mathbf{\Theta})$ according to this distribution, i.e., denoted as $\langle l(\mathbf{\Theta})\rangle_{p(\mathbf{Z}|\mathbf{X},\mathbf{\Theta})}$.
2. The M-step: obtain $\mathbf{\Theta}$ by maximizing $\langle l(\mathbf{\Theta})\rangle_{p(\mathbf{Z}|\mathbf{X},\mathbf{\Theta})}$.

The power of the EM algorithm draws on the Jensen's inequality. The Jensen's inequality says that, let f be a convex function defined on an interval I. If $x_1, x_2, \dots x_n \in I$ and $\gamma_1, \gamma_2, \dots \gamma_n \geq 0$ with $\sum_{i=1}^{n} \gamma_i = 1$, then $f(\sum_{i=1}^{n} \gamma_i x_i) \leq \sum_{i=1}^{n} \gamma_i f(x_i)$.

Below we show how it work:

$$\log p(\mathbf{X}; \mathbf{\Theta}) = \log \int p(\mathbf{X}, \mathbf{Z}; \mathbf{\Theta}) d\mathbf{Z},$$

$$= \log \int Q(\mathbf{Z}) \frac{p(\mathbf{X},\mathbf{Z};\mathbf{\Theta})}{Q(\mathbf{Z})} d\mathbf{Z},$$

$$\geq \int Q(\mathbf{Z}) \log \frac{p(\mathbf{X},\mathbf{Z};\mathbf{\Theta})}{Q(\mathbf{Z})} d\mathbf{Z},$$

$$= \int Q(\mathbf{Z}) \log p(\mathbf{X}, \mathbf{Z}; \mathbf{\Theta})\, d\mathbf{Z} - \int Q(\mathbf{Z}) Q(\mathbf{Z}) d\mathbf{Z}.$$

The EM algorithm proposed to use $Q(\mathbf{Z}) = p(\mathbf{Z}|\mathbf{X}, \mathbf{\Theta})$. At each M-step, it can be see that, while the goal is to maximize $\log p(\mathbf{X}; \mathbf{\Theta})$, we could maximize its lower bound $\int Q(\mathbf{Z}) \log p(\mathbf{X}, \mathbf{Z}; \mathbf{\Theta})\, d\mathbf{Z}$, such that the objective

[1] *Gramacy, R. and Lee, H. Bayesian treed gaussian process models with an application to computer modeling, Journal of American statistical association, 2008.*

[2] *Liu, H., Chen, X., Lafferty, J. and Wasserman, L. Graph-valued regression, NIPS 2009.*

function $\log p(\mathbf{X}; \boldsymbol{\Theta})$ is improved with guarantee that the new objective function won't decrease along the iterations.

IV. Exercises

Data analysis

1. Find five regression datasets from the UCI data repository or R dataset. Conduct a detailed analysis using the linear regression model. Conduct model selection and validation. Conduct residual analysis of your final models, and comment on your results.
2. Find five classification datasets from the UCI data repository or R datasets. Conduct a detailed analysis using the logistic regression model. Conduct model selection and validation. Conduct residual analysis of your final models, and comment on your results.
3. For the five classification datasets you have selected from the UCI data repository or R datasets, conduct a detailed analysis using the random forest model. Conduct model selection and validation. Conduct residual analysis of your final models, and comment on your results.

Derivation

4. Derive a decision tree model that builds logistic regression model in its leaf nodes. You can get some help by reading this article[1]. Name this hybrid decision tree model!

Programming

5. Write your own R script to implement the hybrid decision tree model you have developed.

[1] https://cran.r-project.org/web/packages/rpart/vignettes/usercode.pdf

6. Simulate a dataset that fits this model. Then, build your hybrid decision tree model, logistic regression model, and random forest model on this simulated dataset. Compare their performances.
7. Repeat 6 on some other datasets you select from the UCI data repository or R dataset.