

Week #8 - Logistic Regression As A Predictive Algorithm

Cheng Peng

West Chester University

Contents

1	Introduction	1
2	Cross-Validation in Predictive Modeling	2
2.1	The Logic of the Three-way Split	2
2.2	Cross-validation (CV)	2
2.3	Predictive Performance Measures	4
2.4	Measuring Global Performance - ROC Curve	5
3	Case Study - Diabetes Prediction with Logistic Regression	6
3.1	Data and Variable Descriptions	6
3.2	Research Question	7
3.3	Exploratory Analysis	7
3.4	Standardizing Numerical Predictor Variables	9
3.5	Data Split - Training and Testing Data	9
3.6	Candidate Models	9
3.7	Summary and Conclusion	12

1 Introduction

The logistic regression model as a member of the family of the generalized linear regression model has the following form.

Let Y be the binary response variable and $\{x_1, x_2, \dots, x_n\}$ be the set of predictor variables. If Y takes on either 1 or 0, the multiple logistic regression model is then defined as

$$\frac{E[Y]}{1 - E[Y]} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_k x_k$$

The success probability function

$$p(x_1, x_2, \dots, x_k) = P(Y = 1 | x_1, x_2, \dots, x_k) = \frac{\exp(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_k x_k)}{1 + \exp(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_k x_k)}$$

If Y takes on character values, R uses chooses the alphabetically higher value to model the above probability. For example, if $Y = \text{"diseased"} \text{ or } \text{"disease-free"}$, by default, the logistic regression will be defined as

$$p(x_1, x_2, \dots, x_k) = P(Y = \text{disease-free} | x_1, x_2, \dots, x_k) = \frac{\exp(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_k x_k)}{1 + \exp(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_k x_k)}$$

We can also redefine the factor level of the response variable to model the probability of the desired category.

In the previous module, we introduced the strategies for searching the final model using both manual and automatic variable selection approaches using likelihood-based performance metrics. However, in predictive modeling, the classical model search strategies might not work satisfactorily. It is a common practice to use data-driven and algorithm-based approaches to assess the predictive performance of a predictive model or an algorithm.

In the following sections, we assume that a set of several candidate models/algorithms have already been developed. The candidate models and algorithms include logistic regression models and other algorithms such as tree-based algorithms, neural nets, support vector machines, etc. The objective is to use various methods to choose the one with the best predictive power to implement in real-world applications.

2 Cross-Validation in Predictive Modeling

In classical statistical modeling, we evaluate the predictive performance of a model using the large sample theory developed based on the likelihood. However, in algorithm-based predictive modeling, we don't assume the distribution of the population or the underlying population is uncertain. In other words, there is no general theory that can derive the predictive performance. Therefore, data-driven methods are used to define the goodness of the model. The key is to hold up a portion of the sample as “unseen” observations to test the actual performance of the predictive models and algorithms.

2.1 The Logic of the Three-way Split

The general idea is to randomly partition the data into several subsets for different purposes. One of them will be used to build the model, one is used to validate the model, and one is used as “unseen real-world” data to report the “actual” performance of the final model.



Figure 1: Three-way splitting mechanism

With this three-way split, the **model selection** and the **true error rate computation** can be carried out simultaneously. One important observation is that the error rate estimate of the final model on validation data is, in general, underestimated since the validation set is used to select the final model. Therefore, a third independent part of the data, the **test data**, is needed to report the actual prediction performance.

After the performance of the final model was evaluated on the test set, all estimated parameters of the model based on the training data *must not* be changed any further.

The three-way splitting method may face practical challenges such as the insufficiency of the data and variation of the error estimate based on a single-step process of model development. To overcome these potential challenges, researchers and practitioners developed resampling-based cross-validation (CV) methods to combine training, validating, and testing in an iterative process. The following subsection outlines this

2.2 Cross-validation (CV)

Among the methods available for estimating prediction error, the most widely used is cross-validation. Contrary to most people who thought that CV was developed by machine learning researchers, it was developed by statisticians. The idea was initially formulated in the 1930s and formally published in statistics literature in the 1970s. Maybe due to computational constraints, the cross-validation methods have not been used by the statistics community. Recently, machine learning researchers dug this old but gold algorithm out for assessing the performance of predictive modeling and algorithms.

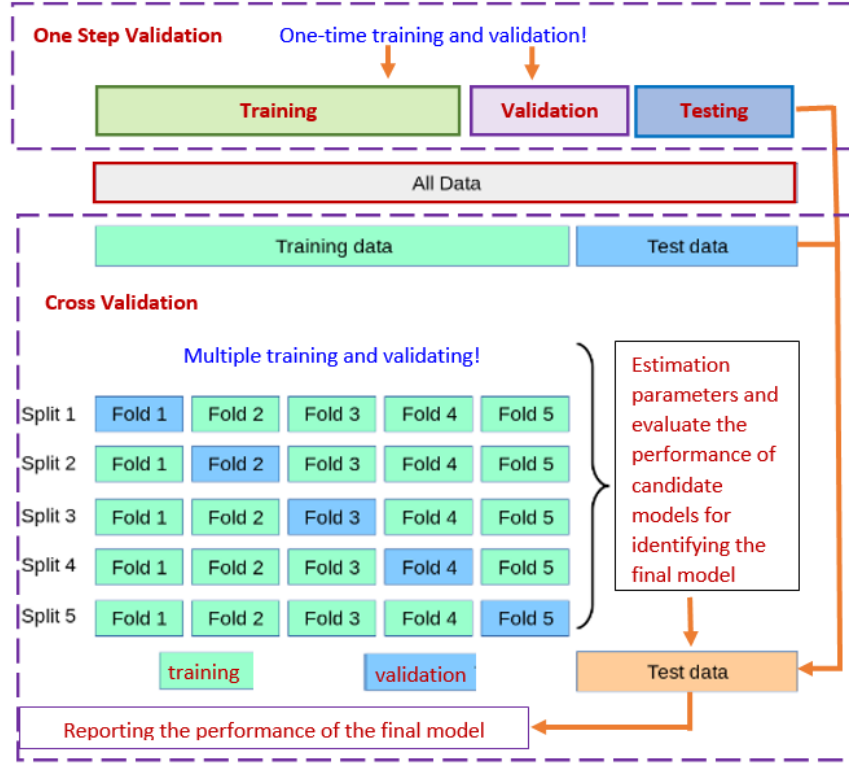


Figure 2: Three-way data splitting versus cross-validation

Essentially cross-validation includes techniques to split the sample into multiple training and test data sets. To obtain multiple training and testing subsets, we use various random sub-sampling methods to perform K data splits of the entire sample. Different random splits constitute different cross-validation methods. The most commonly used cross-validation method is k -fold cross-validation.

The following are steps for choosing the final working model from a set of candidate models using the cross-validation method.

- **Step 1:** Split the entire data into a training set and a testing set.
- **Step 2:** Perform the cross-validation in the training set and don't touch the test data.
 - *step 2.1.* Randomly split the training set into k subsets with equal size, say, F_1, F_2, \dots, F_k .
 - *step 2.2.* hold up F_i and combine $T_i = \{\dots, F_{i-1}, F_{i+1}, \dots\}$. Fit all candidate models to T_i and then use the fitted model to predict the response using the testing set F_i , for $i = 1, 2, \dots, k$.
 - *step 2.3.* Since the actual response values are available in the test set F_i , we can compare the predicted response value with the true response values to calculate the predictive error for each candidate model in each of the k -rounds.
- **Step 3:** Calculate the average of the predictive errors for each candidate model. The one with the smallest average predictive error is the winner.
- **Step 4:** Fit the winner to the entire training set $\{F_1, F_2, \dots, F_k\}$ to obtain the final working model.
- **Step 5:** Report the *actual* predictive error using the testing data set that has not been used in the above cross-validation process.

The above process is summarized in the following figure.

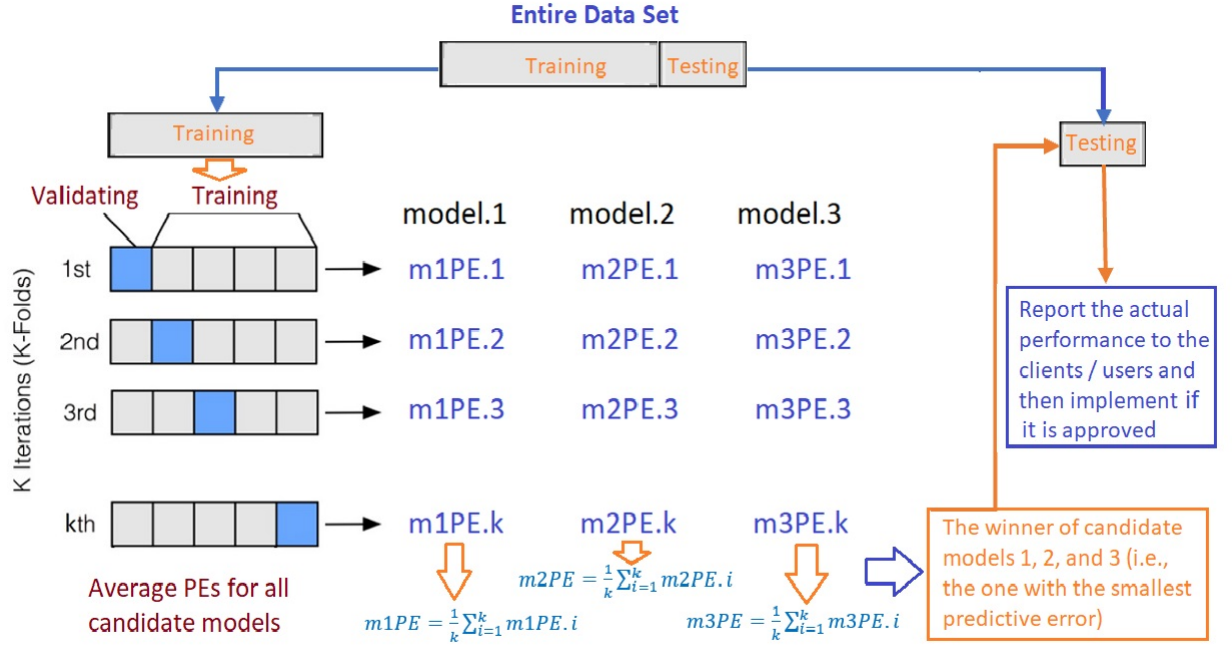


Figure 3: The workflow of the cross-validation process

Remarks on the cross-validation.

1. The proportions of the training and testing sets are determined by modelers. 75%:25% is a commonly used ratio.
2. The number of folds used in the cross-validation is the **hyper-parameter** or **tuning-parameter**. The 10-fold cross-validation is commonly used. The reason 10-fold is commonly used is that when $k \geq 10$, the predictive errors in the training and testing become stable.

2.3 Predictive Performance Measures

We introduced two sets of conditional probabilities using clinical language in the previous module. As an analogy, we can think about the logistic predictive model to be a **diagnostics test** for predicting a specific disease. Note that what the logistic model predicted is the probability of having a **disease**. In order to predict the value (T+ = “predicted disease” or T- = “predicted no-disease”) of the binary response, we need to use a cut-off probability to define “T+” and “T-”. In most software programs, 0.5 was used. If the success probability is greater than 0.5, the predicted value of the response will be “T+”, otherwise, the predicted response value will be “T-”.

Since each of the hold-up validating set has the true values of the response, we then can use corresponding predicted values based on a selected cut-off probability to make a two-way table in the following form

```
Positive.test = c("n11", "n21")
Negative.test = c("n12", "n22")
confusion.matrix = as.data.frame(cbind(Positive.test = Positive.test,
                                         Negative.test = Negative.test))
names(confusion.matrix)=c("T+", "T-")
row.names(confusion.matrix) = c("D+", "D-")
kable(confusion.matrix, caption = "Confusion matrix of the binary logistic predictive model")
```

Table 1: Confusion matrix of the binary logistic predictive model

	T+	T-
D+	n11	n12
D-	n21	n22

The above two-way contingency table is called the **confusion matrix** which can be used to estimate the following four performance measures introduced in the previous module.

- **Sensitivity:**

$$P(\widehat{T+} | D+) = n_{11} / (n_{11} + n_{12})$$

- **Specificity:**

$$P(\widehat{T-} | D-) = n_{22} / (n_{21} + n_{22})$$

- **False Negative Rate:**

$$P(\widehat{T-} | D+) = n_{12} / (n_{11} + n_{12})$$

- **False Positive Rate:**

$$P(\widehat{T+} | D-) = n_{21} / (n_{21} + n_{22})$$

Ideally, a good binary predictive model such as the logistic regression model should have a high sensitivity and specificity and a low false negative rate and false-positive rate (FPR, also called false alarm rate).

The overall accuracy is defined by

$$\text{accuracy} = (n_{11} + n_{22}) / (n_{11} + n_{12} + n_{21} + n_{22})$$

and the predictive error (PE) is defined by

$$PE = (n_{12} + n_{21}) / (n_{11} + n_{12} + n_{21} + n_{22}).$$

The above PE was used in the iterations of the process of cross-validation for a binary predictive model. In the logistic regression model, the confusion matrix is dependent on the choice of the cut-off probability. In other words, the values of the above 6 measures are dependent on the choice of the cut-off probability. Therefore, the above performance metrics are called **local performance** measures.

Remarks

1. The default choice of 0.5 in most software programs may not be the optimal choice.
2. The “disease” probability is a continuous variable on $[0,1]$, we can search the optimal cut-off probability by plotting the cut-off probability against the accuracy and then find the optimal cut-off from the plot.

2.4 Measuring Global Performance - ROC Curve

In many practical applications, we may have different candidate models that behaved differently from different perspectives. For example, (1). how to report the performance of a medical diagnostic test if it is dependent on the age of the patient? (2). If there are two tests and both are dependent on the age of patients, how to compare the performance of the two tests? (3). sometimes the costs of false positive and false negative rates are very different in a specific application, how to minimize the cost due to errors? These questions can be addressed using the Receiver Operating Characteristics (ROC) analysis.

An ROC curve is the plot of sensitivity (i.e., TPR = True Positive Rate) against $(1 - \text{specificity}) = \text{False Positive Rate (FPR)}$. Drawing the ROC curve is straightforward. We only need to choose a sequence of

cut-off probability and then calculate the corresponding TPR and FPR. Then we can plot these points to get the ROC curve which is similar to the following figure.

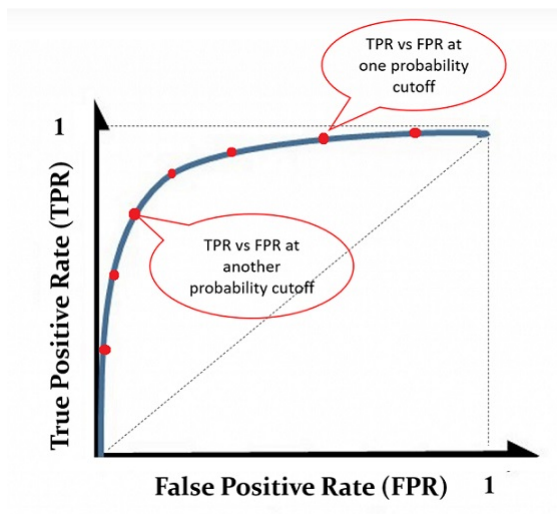


Figure 4: An illustrative ROC curve

Using the same set of cut-off probabilities, different candidate models have different ROC curves. The area under the curve (AUC) reflects the **global goodness of the model**. See the following illustrative curve.

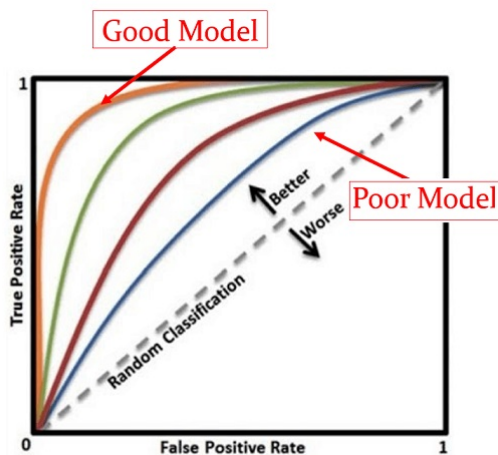


Figure 5: Illustrative comparison of multiple ROC curves associated with the corresponding candidate models

With the ROC curve, we can answer the questions in the opening paragraph of this section.

3 Case Study - Diabetes Prediction with Logistic Regression

In this case study, we still use the diabetes data that was used in the last module.

3.1 Data and Variable Descriptions

There are 9 variables in the data set.

1. **pregnant**: Number of times pregnant

2. **glucose**: Plasma glucose concentration (glucose tolerance test)
3. **pressure**: Diastolic blood pressure (mm Hg)
4. **triceps**: Triceps skinfold thickness (mm)
5. **insulin**: 2-Hour serum insulin (μ U/ml)
6. **mass**: Body mass index (weight in kg/(height in m)²)
7. **pedigree**: Diabetes pedigree function
8. **age**: Age (years)
9. **diabetes**: Class variable (test for diabetes)

I load the data from R **library{mlbench}** in the following code. For convenience, I delete all records with missing values and keep only the records with complete records in this case study. The final analytic data set has 392 records. This

```
library(mlbench)
data(PimaIndiansDiabetes2)           # load the data to R work-space
diabetes.0 = PimaIndiansDiabetes2    # make a copy of the data for data cleansing
diabetes = na.omit(diabetes.0)       # Delete all records with missing components
#head(diabetes)
```

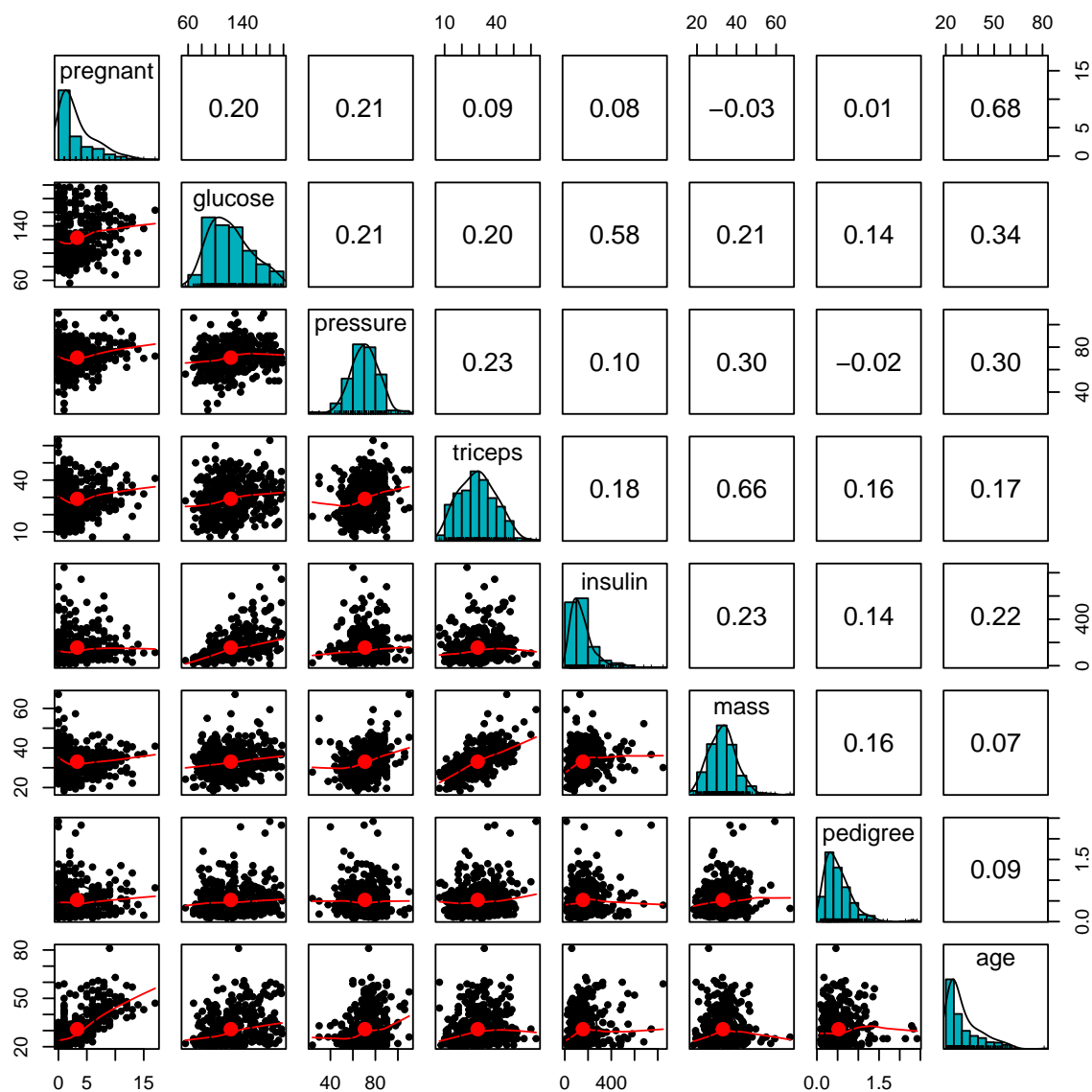
3.2 Research Question

The objective of this case study is to build a logistic regression model to predict diabetes using various risk factors associated with the individual patient.

3.3 Exploratory Analysis

We first make the following pairwise scatter plots to inspect the potential issues with predictor variables.

```
library(psych)
pairs.panels(diabetes[, -9],
             method = "pearson", # correlation method
             hist.col = "#00AFBB",
             density = TRUE, # show density plots
             ellipses = TRUE # show correlation ellipses
             )
```



From the correlation matrix plot, we can see several patterns in the predictor variables.

- All predictor variables are unimodal. But **pregnant** and **age** are significantly skewed. we discretize **pregnant** and **age** in the following.

```
preg = diabetes$pregnant
grp.preg = preg
grp.preg[preg %in% c(4:6)] = "4-6"
grp.preg[preg %in% c(7:9)] = "7-9"
grp.preg[preg %in% c(10:17)] = "10+"
##
age = diabetes$age
##
grp.age = age
grp.age[age %in% c(21:24)] = "21-25"
```



```
grp.age[age %in% c(25:30)] = "25-30"
grp.age[age %in% c(31:40)] = "31-40"
grp.age[age %in% c(41:50)] = "41-50"
grp.age[age %in% c(51:99)] = "50 +"
## added to the diabetes data set
diabetes$grp.age = grp.age
diabetes$grp.preg = grp.preg
```

A moderate correlation is observed in several pairs of variables: **age** v.s. **pregnant**, **glucose** v.s. **insulin**, and **triceps** v.s. **mass**. We will not drop any of these variables. We will standardize these variables. Part of the correlation may be removed after they are standardized.

3.4 Standizing Numerical Predictor Variables

Since this is a predictive model, we don't worry about the interpretation of the coefficients. The objective is to identify a model that has the best predictive performance.

```
## standardizing numerical variables
diabetes$sd.glucose = (diabetes$glucose-mean(diabetes$glucose))/sd(diabetes$glucose)
diabetes$sd.pressure = (diabetes$pressure-mean(diabetes$pressure))/sd(diabetes$pressure)
diabetes$sd.triceps = (diabetes$triceps-mean(diabetes$triceps))/sd(diabetes$triceps)
diabetes$sd.insulin = (diabetes$insulin-mean(diabetes$insulin))/sd(diabetes$insulin)
diabetes$sd.mass = (diabetes$mass-mean(diabetes$mass))/sd(diabetes$mass)
diabetes$sd.pedigree = (diabetes$pedigree-mean(diabetes$pedigree))/sd(diabetes$pedigree)
## drop the original variables except for the response variable
sd.diabetes = diabetes[, -(1:8)]
```

Remark: If the final model is used for real-time prediction, we need to write a separate function to perform all variable transformations and modifications before feeding it to the model for prediction.

3.5 Data Split - Training and Testing Data

We **randomly** split the data into two subsets. 70% of the data will be used as training data. We will use the training data to search the candidate models, validate them and identify the final model using the cross-validation method. The 30% of the hold-up sample will be used for assessing the performance of the final model.

```
## splitting data: 80% training and 20% testing
n <- dim(sd.diabetes)[1]
train.n <- round(0.8*n)
train.id <- sample(1:n, train.n, replace = FALSE)
## training and testing data sets
train <- sd.diabetes[train.id, ]
test <- sd.diabetes[-train.id, ]
```

3.6 Candidat Models

In the previous module, we introduced full and reduced models to set up the scope for searching for the final model. In this case study, we use the full, reduced, and final models obtained based on the step-wise variable selection as the three candidate models.

For convenience, we use 0.5 as the common cut-off for all three models to define the predicted. In a real application, you may want to find the optimal cut-off for each candidate model in the cross-validation process.

- 5-fold Cross-Validation

Since our training data is relatively small, I will use 5-fold cross-validation to ensure the validation data set has enough diabetes cases.

```
## 5-fold cross-validation
k=5
fold.size = round(dim(train)[1]/k)
## PE vectors for candidate models
PE1 = rep(0,5)
PE2 = rep(0,5)
PE3 = rep(0,5)
for(i in 1:k){
  ## Training and testing folds
  valid.id = (fold.size*(i-1)+1):(fold.size*i)
  valid = train[valid.id, ]
  train.dat = train[-valid.id,]
  ## full model
  candidate01 = glm(diabetes ~grp.preg + sd.glucose +sd.pressure+ sd.triceps + sd.insulin +
    sd.mass + sd.pedigree + grp.age, family = binomial(link = "logit"),
    data = train.dat)
  ## reduced model
  candidate03 = glm(diabetes ~ sd.insulin + sd.mass + sd.pedigree,
    family = binomial(link = "logit"),
    data = train.dat)
  ##
  candidate02 = stepAIC(candidate01,
    scope = list(lower=formula(candidate03),upper=formula(candidate01)),
    direction = "forward", # forward selection
    trace = 0 # do not show the details
  )
  ## predicted probabilities of each candidate model
  pred01 = predict(candidate01, newdata = valid, type="response")
  pred02 = predict(candidate02, newdata = valid, type="response")
  pred03 = predict(candidate03, newdata = valid, type="response")
  ## confusion matrix: ftable() will
  confusion.matrix01 = ftable(as.vector((pred01>0.5)),(valid$diabetes=="pos"))
  confusion.matrix02 = ftable(as.vector((pred02>0.5)),(valid$diabetes=="pos"))
  confusion.matrix03 = ftable(as.vector((pred03>0.5)),(valid$diabetes=="pos"))
  PE1[i] = (confusion.matrix01[1,2] + confusion.matrix01[2,1])/length(pred01)
  PE2[i] = (confusion.matrix02[1,2] + confusion.matrix02[2,1])/length(pred02)
  PE3[i] = (confusion.matrix03[1,2] + confusion.matrix03[2,1])/length(pred03)
}
avg.pe = cbind(PE1 = mean(PE1), PE2 = mean(PE2), PE3 = mean(PE3))
kable(avg.pe, caption = "Average of prediction errors of candidate models")
```

Table 2: Average of prediction errors of candidate models

PE1	PE2	PE3
0.2190476	0.2190476	0.2793651

The average predictive errors show that candidate models 1 and 2 have the same predictive error. Since model 2 is simpler than model 1, we choose model 2 as the final predictive model.

The actual accuracy of the final model is given by

```
pred02 = predict(candidate02, newdata = test, type="response")
confusion.matrix02 = ftable(as.vector((pred02>0.5)),(test$diabetes=="pos"))
accuracy = (confusion.matrix02[1,1] + confusion.matrix02[2,2])/length(pred02)
kable(accuracy, caption="The actual accuracy of the final model")
```

Table 3: The actual accuracy of the final model

x
0.7435897

Therefore, the final model has an accuracy rate given in the above table.

Remark: Since we used a random split method to define the training and testing data when re-running the code, the performance metrics will be slightly different.

- **Selecting the final model with ROC**

We first estimate the TPR (true positive rate, sensitivity) and FPR (false positive rate, 1 - specificity) at each cut-off probability for each of the three candidate models using the following R function.

```
## A function to extract false positive and false negative rates
TPR.FPR=function(pred){
  prob.seq = seq(0,1, length=15)
  pn=length(prob.seq)
  true.lab=as.vector(test$diabetes)
  TPR = NULL
  FPR = NULL
  ##
  for (i in 1:pn){
    pred.lab = as.vector(ifelse(pred >prob.seq[i],"pos", "neg"))
    TPR[i] = length(which(true.lab=="pos" & pred.lab=="pos"))/length(which(true.lab=="pos"))
    FPR[i] = length(which(true.lab=="neg" & pred.lab=="pos"))/length(which(true.lab=="neg"))
  }
  cbind(FPR = FPR, TPR = TPR)
}
```

The ROC curves of the three candidate models are given below.

```
## candidate models
## full model
candidate01 = glm(diabetes ~grp.preg + sd.glucose +sd.pressure+ sd.triceps + sd.insulin +
                  sd.mass + sd.pedigree + grp.age, family = binomial(link = "logit"),
                  data = train)

## reduced model
candidate03 = glm(diabetes ~ sd.insulin + sd.mass + sd.pedigree,
                  family = binomial(link = "logit"),
                  data = train)

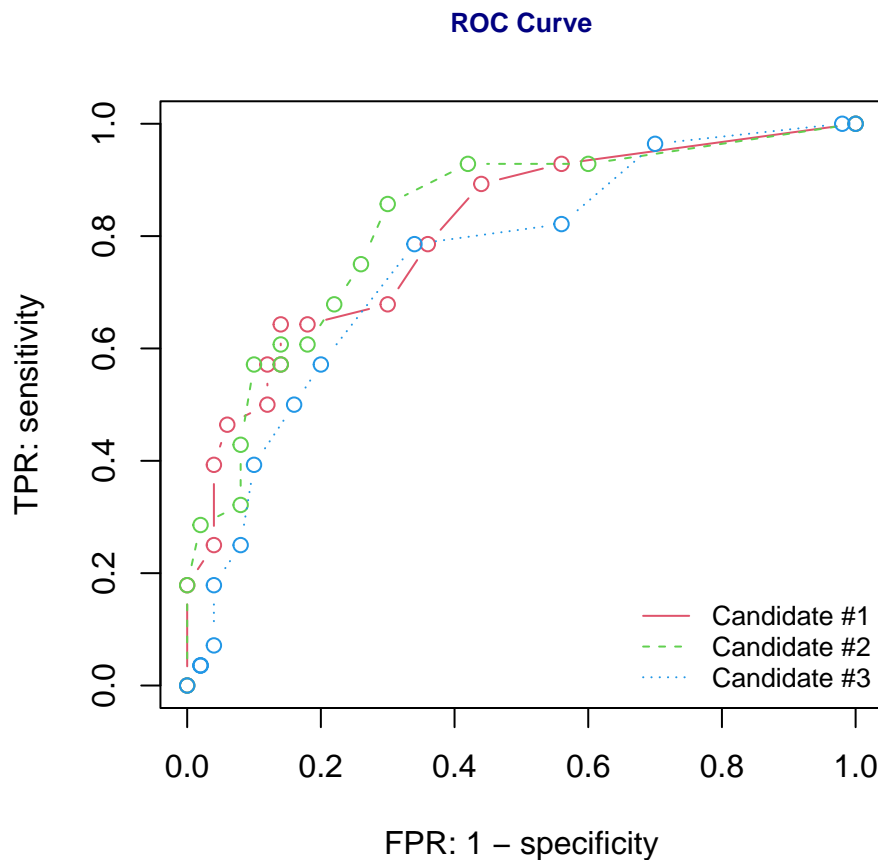
##
candidate02 = stepAIC(candidate03,
                      scope = list(lower=formula(candidate03),upper=formula(candidate01)),
                      direction = "forward", # forward selection
                      trace = 0 # do not show the details
                      )

## predicted probabilities
pred01 = predict(candidate01, newdata = test, type="response")
```

```

pred02 = predict(candidate02, newdata = test, type="response")
pred03 = predict(candidate03, newdata = test, type="response")
####
## ROC curve
plot(TPR.FPR(pred01)[,1], TPR.FPR(pred01)[,2],
     type="b", col=2, lty=1, xlim=c(0,1), ylim=c(0,1),
     xlab = "FPR: 1 - specificity",
     ylab = "TPR: sensitivity",
     main = "ROC Curve",
     cex.main = 0.8,
     col.main = "navy")
lines(TPR.FPR(pred02)[,1], TPR.FPR(pred02)[,2], type="b", col=3, lty=2)
lines(TPR.FPR(pred03)[,1], TPR.FPR(pred03)[,2], type="b", col=4, lty=3)
legend("bottomright", c("Candidate #1", "Candidate #2", "Candidate #3"),
      col=2:4, lty=1:3, cex = 0.8, bty="n")

```



We can see from the ROC curve that candidate model 02 has a better global fit than the full and reduced models. Based on the area under the ROC curve, we still choose model #2 as the final working model.

3.7 Summary and Conclusion

The case study focused on predicting diabetes. For illustrative purposes, we used three models as candidates and use both cross-validation and ROC curve to select the final working model. Both cross-validation and

ROC curve yielded the same result.