统计机器学习 Homework 03

姓名: 雍崔扬
学号: 21307140051

Problem 1

Part (1)

Write the Newton-Raphson algorithm to estimate logistic regression, i.e., derive the equation:

$$\frac{\partial^2 \ell(\beta)}{\partial \beta \partial \beta^\top} = -\sum_i x_i x_i^\top p(x_i;\beta)\{1 - p(x_i;\beta)\}$$

Solution:
给定训练集 $D_{\text{train}} = \{(x^{(i)}, y_i)\}_{i=1}^n$,
用 Logistic 回归模型对每个样本 $x^{(i)}$ 进行预测,
输出其标签为 1 的后验概率，记为 $\hat{y}_i = \sigma(\beta^{\text{T}} x^{(i)})\ \ (i = 1, \dots, n)$.

而真实条件概率可以表示为 $\begin{cases} p_r(y_i = 1|x^{(i)}) = y_i \\ p_r(y_i = 0|x^{(i)}) = 1 - y_i \end{cases}$
使用**交叉熵损失函数**: (简单起见，不考虑正则化项)

$$\begin{aligned} l(\beta) &= -\sum_{i=1}^n \{p_r(y_i = 1|x)\log(\hat{y}_i) + p_r(y_i = 0|x)\log(1 - \hat{y}_i)\} \\ &= -\sum_{i=1}^n \{y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\} \\ &(\text{where } \hat{y}_i = \sigma(\beta^{\text{T}} x^{(i)})\ \ (i = 1, \dots, n)) \end{aligned}$$

则我们有:

$$
\begin{aligned}
\frac{\partial}{\partial\beta^\top}l(\beta) &= -\sum_{i=1}^{n}\{y_i\frac{\partial}{\partial\beta^\top}\log(\hat{y}_i)+(1-y_i)\frac{\partial}{\partial\beta^\top}\log(1-\hat{y}_i)\} \\
&= -\sum_{i=1}^{n}\{y_i\frac{1}{\hat{y}_i}\frac{\partial}{\partial\beta^\top}\hat{y}_i+(1-y_i)(-\frac{1}{1-\hat{y}_i})\frac{\partial}{\partial\beta^\top}\hat{y}_i\} \quad (\text{note that } \frac{\partial}{\partial\beta}\hat{y}_i=\hat{y}_i(1-\hat{y}_i)x^{(i)}) \\
&= -\sum_{i=1}^{n}\{y_i\frac{\hat{y}_i(1-\hat{y}_i)}{\hat{y}_i}x^{(i)}-(1-y_i)\frac{\hat{y}_i(1-\hat{y}_i)}{1-\hat{y}_i}x^{(i)}\} \\
&= -\sum_{i=1}^{n}\{y_i(1-\hat{y}_i)x^{(i)}-(1-y_i)\hat{y}_ix^{(i)}\} \\
&= -\sum_{i=1}^{n}(y_i-\hat{y}_i)x^{(i)} \\
&(\text{where } \hat{y}_i=\sigma(\beta^{\mathrm{T}}x^{(i)}) \;\; (i=1,\ldots,n)) \\
&= -X^{\mathrm{T}}(y-\hat{y}) \\
&(\text{where } X=[x^{(1)},\ldots,x^{(n)}]^{\mathrm{T}},\hat{y}=[\hat{y}_1,\ldots,\hat{y}_n]^{\mathrm{T}}=\sigma(X\beta))
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial^2}{\partial\beta\partial\beta^\top}l(\beta) &= -\sum_{i=1}^{n}x^{(i)}\frac{\partial}{\partial\beta}(y_i-\hat{y}_i) \quad (\text{note that } \frac{\partial}{\partial\beta}\hat{y}_i=\hat{y}^{(i)}(1-\hat{y}^{(i)})(x^{(i)})^{\mathrm{T}}) \\
&= -\sum_{i=1}^{n}x^{(i)}\{-\hat{y}_i(1-\hat{y}_i)(x^{(i)})^{\mathrm{T}}\} \\
&= \sum_{i=1}^{n}\hat{y}_i(1-\hat{y}_i)x^{(i)}(x^{(i)})^{\mathrm{T}} \\
&(\text{where } \hat{y}_i=\sigma(\beta^{\mathrm{T}}x^{(i)}) \;\; (i=1,\ldots,n)) \\
&= \frac{1}{n}X^{\mathrm{T}}\mathrm{diag}\{\hat{y}\odot(1_n-\hat{y})\}X \\
&(\text{where } X=[x^{(1)},\ldots,x^{(n)}]^{\mathrm{T}},\hat{y}=[\hat{y}_1,\ldots,\hat{y}_n]^{\mathrm{T}}=\sigma(X\beta))
\end{aligned}
$$

这样我们就证明了:

$$
\frac{\partial^2}{\partial\beta\partial\beta^\top}l(\beta)=\sum_{i=1}^{n}\hat{y}_i(1-\hat{y}_i)x^{(i)}(x^{(i)})^{\mathrm{T}}=\sum_{i=1}^{n}p(x^{(i)};\beta)(1-p(x^{(i)};\beta))x^{(i)}(x^{(i)})^{\mathrm{T}}
$$

Generate $X=(1,X_1,X_2)$, where $X_j\sim N(0,I_N)$ $(j=1,2)$.
Set true parameter $\beta=(0.5,1.2,-1)^\top$.
Set $N=200,500,800,1000$.
Estimate β using the Newton-Raphson (NR) algorithm for $R=200$ rounds of simulation.
For each round of simulation, terminate the iteration when $\max_j|\beta_j^{\mathrm{old}}-\beta_j^{\mathrm{new}}|<10^{-5}$
Denote $\hat{\beta}_j^{(r)}$ as the estimation of β_j in the r-th round of simulation.
Then please, for each j, draw $\hat{\beta}_j^{(r)}-\beta_j$ in boxplots for $N=200,500,800,1000$.

Solution:
生成样本的 python 代码:

```
# Function to generate data
def generate_data(N, true_beta):
    X1 = np.random.normal(size=N)
    X2 = np.random.normal(size=N)
    X = np.column_stack((np.ones(N), X1, X2))  # Adding intercept
    p = 1 / (1 + np.exp(-X @ true_beta))  # Compute probabilities
    y = np.random.binomial(1, p)  # Generate binary outcomes
    return X, y
```

纯 Newton 法的 python 代码:

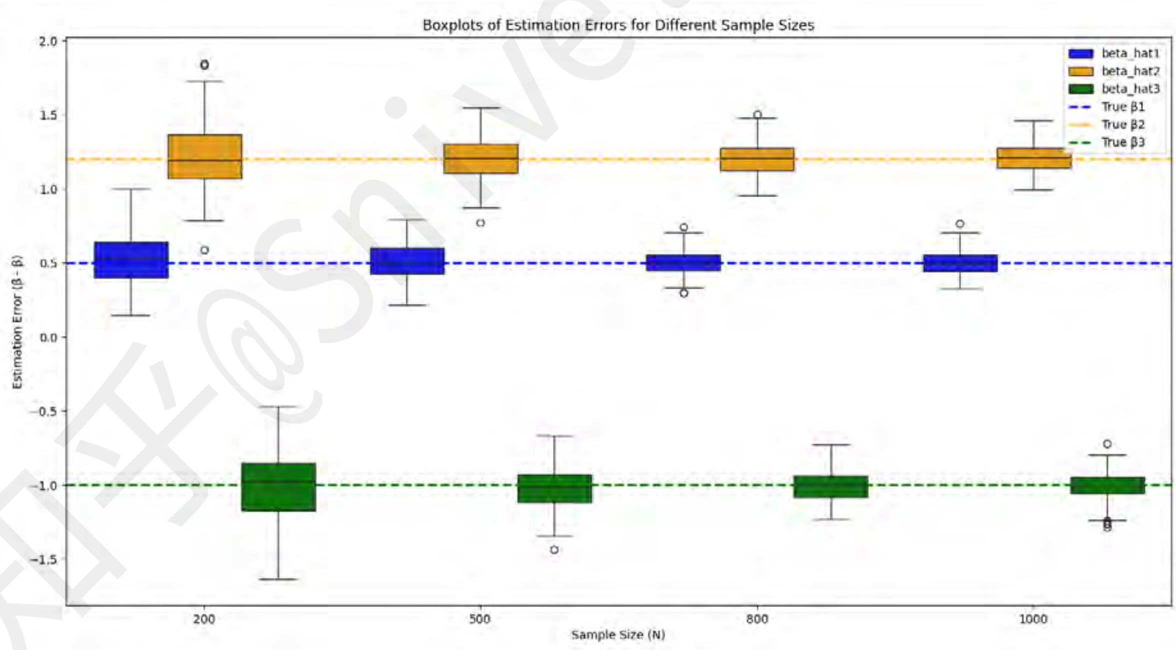
```
# Newton function returning the history of beta
def newton(X, y, max_iter=20, tolerance=1e-5):
    beta = np.zeros(X.shape[1])  # Initialize beta as a 1D array
    for i in range(max_iter):
        p_hat = 1 / (1 + np.exp(-X @ beta))  # Compute predicted probabilities
        gradient = X.T @ (y - p_hat)  # Compute the gradient
        hessian = -X.T @ np.diag(p_hat * (1 - p_hat)) @ X  # Compute the Hessian
        beta_new = beta - np.linalg.solve(hessian, gradient)  # Update beta

        if np.max(np.abs(beta_new - beta)) < tolerance:  # Check for convergence
            beta = beta_new
            break

        beta = beta_new  # Update beta for next iteration
    return beta
```

函数调用及绘制箱线图的代码:

```
# Set parameters
np.random.seed(51)
true_beta = np.array([0.5, 1.2, -1])
N_values = [200, 500, 800, 1000]
Rounds = 200

# Run simulations
estimates = []
for N in N_values:
    rounds_estimates = []
    for r in range(Rounds):
        X, y = generate_data(N, true_beta)
        rounds_estimates.append(newton(X, y))  # Get the final estimate

        # Print progress every 10%
        if (r + 1) % (Rounds // 10) == 0:
            print(f"Progress for N={N}: Completed {((r + 1) / Rounds) *
100:.1f}%")

    estimates.append(np.array(rounds_estimates))

# Prepare data for boxplot
results = []
for i, N in enumerate(N_values):
    for r in range(Rounds):
        results.append({
            'N': N,
            'beta_hat1': estimates[i][r][0],  # Directly use estimates
            'beta_hat2': estimates[i][r][1],
            'beta_hat3': estimates[i][r][2]
        })

results_df = pd.DataFrame(results)

# Melt results for seaborn
results_melted = results_df.melt(id_vars='N', value_vars=['beta_hat1',
'beta_hat2', 'beta_hat3'],
```

```
                                     var_name='variable', value_name='value')

# Plot boxplots
plt.figure(figsize=(10, 6))
sns.boxplot(x='N', y='value', hue='variable', data=results_melted,
            palette={"beta_hat1": "blue", "beta_hat2": "orange", "beta_hat3":
"green"})
plt.xlabel("Sample Size (N)")
plt.ylabel("Estimation Error (β - β)")
plt.title("Boxplots of Estimation Errors for Different Sample Sizes")
handles = [
    mpatches.Patch(color='blue', label='β1'),
    mpatches.Patch(color='orange', label='β2'),
    mpatches.Patch(color='green', label='β3')
]
plt.legend(handles=handles, title='')

# Plot true values of β
for i, beta in enumerate(true_beta):
    plt.axhline(y=beta, color=['blue', 'orange', 'green'][i], linestyle='--',
linewidth=2, label=f'True β{i+1}')

# Show legend for true values
plt.legend(title='', loc='upper right')

plt.tight_layout()
plt.show()
```

运行结果:



Part (2)

假设有 N_+ 个正例和 N_- 个负例，令 D_+ 与 D_- 分别表示正例、负例集合.
定义排序 "损失" 如下:

$$\ell_{\text{rank}} = \frac{1}{N_+N_-} \sum_{x^+ \in D_+} \sum_{x^- \in D_-} \left(I(f(x^+) < f(x^-)) + \frac{1}{2} I(f(x^+) = f(x^-)) \right)$$

理解: 若正例的预测值小于负例，则记 1 个 "罚分"，若相等，则记 0.5 个罚分.
定义 $\text{AUC} := 1 - \ell_{\text{rank}}$
考虑一种简单的情况，即当数据中不存在 $f(x^+) = f(x^-)$ 时，定义排序 "损失" 如下:

$$\ell_{\text{rank}} = \frac{1}{N_+N_-}\sum_{x^+\in D_+}\sum_{x^-\in D_-} I(f(x^+) < f(x^-))$$

试证明以上定义的 AUC 即有限样本的 ROC 曲线下方的面积.

Proof:
[(ROC 曲线和 AUC)]
ROC 曲线的纵轴为 $\text{TPR} = \text{Recall} = \frac{\text{TP}}{\text{TP}+\text{FN}}$，横轴为 $\text{FPR} := \frac{\text{FP}}{\text{TN}+\text{FP}}$
而 AUC 是 **ROC 曲线下的面积** (Area Under Curve)，其取值范围为 $[0,1]$

有限样本集上的 ROC 曲线:

- ① 设分类阈值为 1，则 $(\text{FPR}, \text{TPR}) = (0,0)$
- ② 将预测值从高到低排序，将阈值依次设为预测值，分别计算 (FPR, TPR) (可以利用已经计算的结果快速更新)
  若当前样本是真正例，则向纵轴方向走 (FPR 部分不变，TPR 部分增长一个单位 $\frac{1}{N_+}$)
  若当前样本是假正例，则向横轴方向走 (FPR 部分增长一个单位 $\frac{1}{N_-}$，TPR 部分不变)

记 $N := N_+ + N_-$
不失一般性，假设预测值是从高到低排序的 (即有 $f(x_1) \geq \cdots \geq f(x_N)$)
其中负例的指标为 $i_1, \ldots, i_{N_-}$
则我们有:

$$\begin{aligned}
\text{AUC} &= 1 - l_{\text{rank}} \\
&= 1 - \frac{1}{N_-N_+}\sum_{x_-\in D_-}\sum_{x^+\in D_+}[1 - I(f(x_+) < f(x_-))] \\
&= \frac{1}{N_-N_+}\sum_{x_-\in D_-}\sum_{x^+\in D_+} I(f(x_+) \geq f(x_-)) \\
&= \frac{1}{N_-N_+}\sum_{x_-\in D_-} \text{Number of positive samples } x_+ \text{ that satisfies } f(x_+) \geq f(x_-) \\
&= \frac{1}{N_-N_+}\sum_{j=1}^{N_-} \text{Number of positive samples in } \{x_1, \ldots, x_{i_j}\} \\
&= \sum_{j=1}^{N_-}\frac{1}{N_-}\frac{i_j - j}{N_+} \\
&= S(\text{Area Under ROC Curve})
\end{aligned}$$

因此 $\text{AUC} := 1 - l_{\text{rank}}$ 即为 ROC 曲线下方的面积.

Problem 2

客户流失预警数据:

- 训练数据集: `sampledata.csv`
- 测试数据集: `preddata.csv`

数据文件来自国内某运营商，数据已经进行了清理，数据集共 8 个变量:

	变量名		详细说明	备注
因变量（下月）	churn	是否流失	1=流失 0=不流失	流失率 1.25%
自变量（当月）	tenure	在网时长	连续变量 单位：天	客户从入网到截止数据提取日期时在网时间
	expense	当月花费	连续变量 单位：元	客户在提取月份时的花费总额
	degree	个体的度	连续变量 单位：人数	和客户通话的总人数，去重之后的呼入与呼出加总
	tightness	联系强度	连续变量 分钟/人	通话总时间除以总人数
	entropy	个体信息熵	连续变量	$E_i = -\sum_{a_{ij}=1} p_{ij} * \log(p_{ij})$，其中$E_i$为个体 i 的信息熵，$a_{ij} = 1$代表个体 i 和 j 通过电话，$p_{ij}$代表 j 和 i 通话的分钟数据占 i 总通话分钟的比例
	chgdegree	个体度的变化	连续变量 单位：%	（本月个体的度-上月个体的度）/上月个体的度
	chgexpense	花费的变化	连续变量 单位：%	（本月花费-上月花费）/上月花费

Part (1)

读入数据:

```
library(ROSE)

# 读取训练数据集并过采样
train_data <- read.csv("sampledata.csv")
train_data <- ovun.sample(churn ~ ., data = train_data, method = "over", N = 2 *
nrow(train_data))$data

# 读取测试数据集并过采样
test_data <- read.csv("preddata.csv")
test_data <- ovun.sample(churn ~ ., data = test_data, method = "over", N = 2 *
nrow(test_data))$data
```

其中过采样是因为我们发现在训练集中，正例只占 1.24%

```
> train_data <- read.csv("sampledata.csv")
> sum(train_data$churn == 1)
[1] 602
> nrow(train_data)
[1] 48285
> 602 / 48285 * 100
[1] 1.246764
```

Part (2)

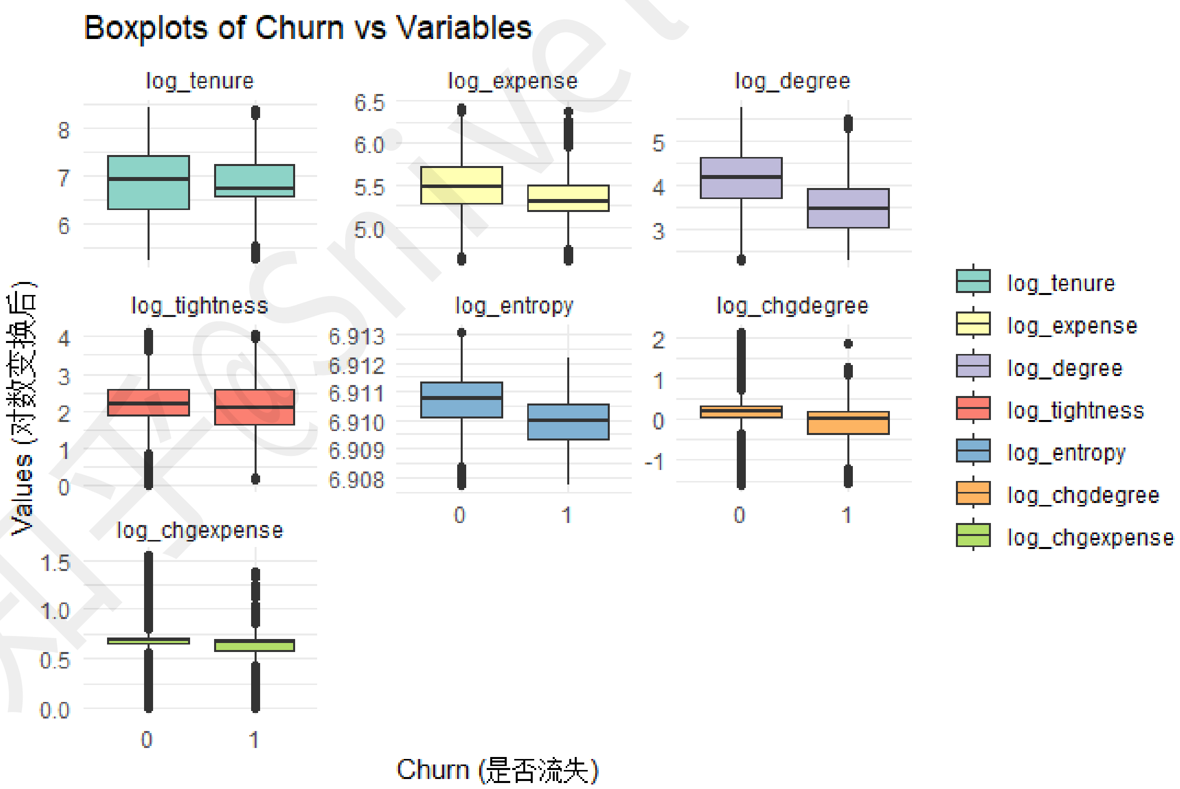
绘制因变量和各个自变量的箱线图 (提示: 可以对右偏分布的数据取对数)

Solution:

```
library(ggplot2)
library(reshape2)
```

```r
# 对右偏分布的数据进行对数变换（以防止负值）
alpha = c(0, 1e2, 1e1, 1, 1e3, 1.2, 2)
train_data$log_tenure <- log(train_data$tenure + alpha[1])
train_data$log_expense <- log(train_data$expense + alpha[2])
train_data$log_degree <- log(train_data$degree + alpha[3])
train_data$log_tightness <- log(train_data$tightness + alpha[4])
train_data$log_entropy <- log(train_data$entropy + alpha[5])
train_data$log_chgdegree <- log(train_data$chgdegree + alpha[6])
train_data$log_chgexpense <- log(train_data$chgexpense + alpha[7])

# 创建一个包含所有自变量和因变量的数据框
melted_data <- melt(train_data, id.vars = "churn",
                    measure.vars = c("log_tenure", "log_expense", "log_degree",
                                     "log_tightness", "log_entropy",
                                     "log_chgdegree", "log_chgexpense"))

# 绘制箱线图
ggplot(melted_data, aes(x = factor(churn), y = value, fill = variable)) +
  geom_boxplot() +
  labs(x = "Churn (是否流失)", y = "Values (对数变换后)",
       title = "Boxplots of Churn vs Variables") +
  theme_minimal() +
  scale_fill_brewer(palette = "Set3") +
  theme(legend.title = element_blank()) +
  facet_wrap(~ variable, scales = "free_y")
```

运行结果:



Part (3)

以是否流失为因变量，使用 `scale()` 函数对自变量进行标准化 (使其均值为 0，方差为 1)
使用 `glm()` 函数建立逻辑回归模型，给出系数估计结果，并对结果进行解读.

Solution:

```r
# 标准化自变量
train_data$standardized_tenure <- scale(train_data$tenure)
train_data$standardized_expense <- scale(train_data$expense)
train_data$standardized_degree <- scale(train_data$degree)
train_data$standardized_tightness <- scale(train_data$tightness)
train_data$standardized_entropy <- scale(train_data$entropy)
train_data$standardized_chgdegree <- scale(train_data$chgdegree)
train_data$standardized_chgexpense <- scale(train_data$chgexpense)

# 建立逻辑回归模型
model <- glm(churn ~ standardized_tenure + standardized_expense +
               standardized_degree + standardized_tightness +
               standardized_entropy + standardized_chgdegree +
               standardized_chgexpense,
             data = train_data,
             family = binomial)

# 输出系数估计结果
summary(model)
```

输出结果:

```
Coefficients:
                         Estimate Std. Error z value Pr(>|z|)
(Intercept)              0.007507   0.007542   0.995     0.32
standardized_tenure     -0.179136   0.008318 -21.535   <2e-16 ***
standardized_expense    -0.242420   0.008585 -28.236   <2e-16 ***
standardized_degree     -0.467112   0.016561 -28.205   <2e-16 ***
standardized_tightness  -0.224234   0.007843 -28.590   <2e-16 ***
standardized_entropy    -0.498783   0.015555 -32.066   <2e-16 ***
standardized_chgdegree  -0.259828   0.008355 -31.099   <2e-16 ***
standardized_chgexpense -0.122920   0.007941 -15.478   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 133859  on 96569  degrees of freedom
Residual deviance: 111191  on 96562  degrees of freedom
AIC: 111207

Number of Fisher Scoring iterations: 4
```

结果解读:

- 模型残差偏差 (Residual deviance) 为 `111191` (因变量数据的固有偏差为 `133859`)，AIC 值为 `111207`
  表明该模型具有较好的拟合效果且复杂度不高.

- `***` 表示 p 值小于 0.001，`**` 表示 p 值小于 0.01，`*` 表示 p 值小于 0.05
  上述结果显示截距项和 7 个自变量均极为显著.

Part (4)

使用建立好的逻辑回归模型，分别使用 `predict()` 函数对训练集和测试集进行预测，得到每个用户的预测流失概率值.

Solution:

```
# 训练集预测
train_data$predicted_probabilities <- predict(model, newdata = train_data, type =
"response")
head(train_data[, c("churn", "predicted_probabilities")])  # 查看训练集的预测结果

# 测试集预测
test_data$standardized_tenure <- scale(test_data$tenure)
test_data$standardized_expense <- scale(test_data$expense)
test_data$standardized_degree <- scale(test_data$degree)
test_data$standardized_tightness <- scale(test_data$tightness)
test_data$standardized_entropy <- scale(test_data$entropy)
test_data$standardized_chgdegree <- scale(test_data$chgdegree)
test_data$standardized_chgexpense <- scale(test_data$chgexpense)

test_data$predicted_probabilities <- predict(model, newdata = test_data, type =
"response")
head(test_data[, c("predicted_probabilities")])  # 查看测试集的预测结果
```

Part (5)

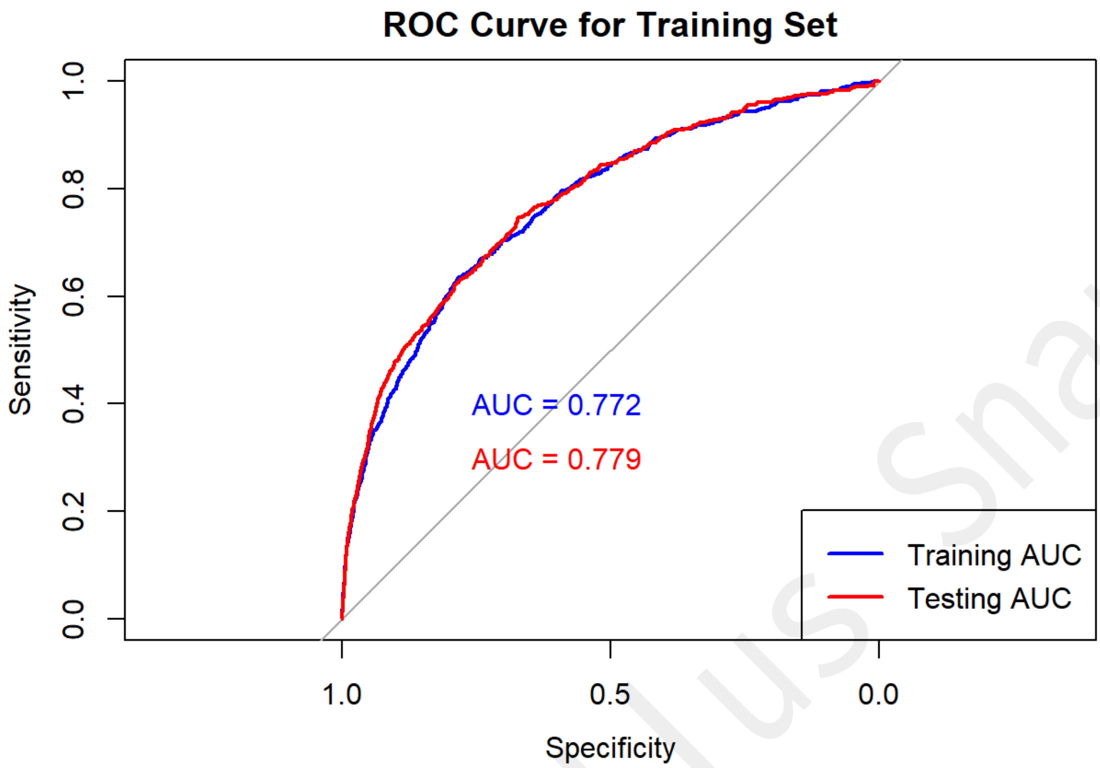
基于 Part(4) 中预测的结果，分别使用 R 包 `pROC` 中的 `plot.roc()` 绘制训练集和测试集上预测结果的 ROC 曲线，
计算相应的 AUC 值，并根据 ROC 曲线和 AUC 值对模型进行评价.

Solution:

```
library(pROC)

# 绘制训练集的 ROC 曲线
roc_train <- roc(train_data$churn, train_data$predicted_probabilities)
plot(roc_train, main = "ROC Curve for Training Set", col = "blue")
auc_train <- auc(roc_train)
print(paste("AUC on train data:", round(auc_train, 3)))
text(0.6, 0.4, paste("AUC =", round(auc_train, 3)), col = "blue")

# 绘制测试集的 ROC 曲线
roc_test <- roc(test_data$churn, test_data$predicted_probabilities)
plot(roc_test, main = "ROC Curve for Testing Set", col = "red", add = TRUE)
auc_test <- auc(roc_test)
print(paste("AUC on test data:", round(auc_test, 3)))
text(0.6, 0.3, paste("AUC =", round(auc_test, 3)), col = "red")

# 添加图例
legend("bottomright", legend = c("Training AUC", "Testing AUC"),
       col = c("blue", "red"), lwd = 2)
```

运行结果:

```
"AUC on train data: 0.772"
"AUC on test data: 0.779"
```



模型无论是在测试集还是训练集上 AUC 值都远离 0.5，接近 1，说明其对顾客是否流失的预测能力较强. (AUC 值越接近于 1，当前的分类算法越有可能将正样本排在负样本前面，即能够更好的分类)

Part (6)

(自己补充的题目)
基于 Part (5) 绘制的 ROC 曲线，设定最优阈值用于对预测结果进行二分类.

Solution:

```
# 计算 Youden's J
coords <- coords(roc_train, "best", ret = c("threshold", "sensitivity",
"specificity"))

# 输出最佳阈值
best_threshold <- as.numeric(coords["threshold"])
print(paste("Best threshold:", round(best_threshold, 3)))

# 二分类：将预测概率转换为分类标签
test_data$predicted_class <- ifelse(test_data$predicted_probabilities >=
best_threshold, 1, 0)

# 计算分类准确率
train_accuracy <- mean(train_data$predicted_class == train_data$churn)
print(paste("训练集分类准确率:", round(train_accuracy * 100, 2), "%"))
test_accuracy <- mean(test_data$predicted_class == test_data$churn)
print(paste("测试集分类准确率:", round(test_accuracy * 100, 2), "%"))
```

输出结果:

```
"Best threshold: 0.575"
"训练集分类准确率: 70.89 %"
"测试集分类准确率: 58.93 %"
```

The End