Classify Stroke Data Using Gaussian Classifier and Logistic Regression

Jaucelyn Canfield

Introduction

For my project I will use a data set that has 11 features used to predict whether a person will have a stroke or not. I shall consider a binary class problem where one class represents a stroke (Class 1) and the other class represents no stroke (Class 0). I have chosen two classification methods to classify the data: Gaussian Classifier and Logistic Regression. In the case for logistic regression, the beta coefficients are found by using iterative least squares and performing 100 bootstraps on the train data to and then taking the mean of each beta value.

Data Exploration

```
# Read in data
data <- read.csv("/Users/jaucelyncanfield/documents/stat 760/healthcare-dataset-stroke-data.csv",header
```

```
head(data)
```

```
##      id gender age hypertension heart_disease ever_married     work_type
## 1  9046   Male  67            0             1          Yes       Private
## 2 51676 Female  61            0             0          Yes Self-employed
## 3 31112   Male  80            0             1          Yes       Private
## 4 60182 Female  49            0             0          Yes       Private
## 5  1665 Female  79            1             0          Yes Self-employed
## 6 56669   Male  81            0             0          Yes       Private
##   Residence_type avg_glucose_level  bmi  smoking_status stroke
## 1          Urban            228.69 36.6 formerly smoked      1
## 2          Rural            202.21  N/A    never smoked      1
## 3          Rural            105.92 32.5    never smoked      1
## 4          Urban            171.23 34.4          smokes      1
## 5          Rural            174.12   24    never smoked      1
## 6          Urban            186.21   29 formerly smoked      1
```

```
#Clean Data

#check for NAs
#NA is inputted as N/A so I need to fix it
for(i in 1:nrow(data)){
  for(j in 1:ncol(data)){
    if(data[i,j]=="N/A"){
      data[i,j] <- NA
    }
  }
}

sum(is.na(data))
```

```
## [1] 201
```

```
#remove observations where bmi is NA
data <- na.omit(data)
```

```
#redefine variables
data$gender <- ifelse(data$gender=="Male",1,0)
data$ever_married <- ifelse(data$ever_married=="Yes",1,0)
#consider private vs other instead of all three cases
data$work_type <- ifelse(data$work_type=="Private",1,0)
data$Residence_type <- ifelse(data$Residence_type=="Urban",1,0)
data$bmi <- as.numeric(data$bmi)
```

There are too many observations where the `smoking_status` is unknown. So we will not consider this feature in our classification.

```
#remove first column with id and the smoking status feature
data <- data[,-c(1,11)]
```

Now, we will check for Multicollinearity.

```
var_names <- c("gender","age","hypertension","heart disease", "married","work", "residence",
               "glucose","bmi","stroke")
knitr::kable(cor(data),digits=2,col.names=var_names)
```

	gender	age	hypertension	heart disease	married	work	residence	glucose	bmi	stroke
gender	1.00	-0.03	0.02	0.08	-0.04	-0.04	0.00	0.05	-0.03	0.01
age	-0.03	1.00	0.27	0.26	0.68	0.12	0.01	0.24	0.33	0.23
hypertension	0.02	0.27	1.00	0.12	0.16	0.00	0.00	0.18	0.17	0.14
heart_disease	0.08	0.26	0.12	1.00	0.11	0.00	0.00	0.15	0.04	0.14
ever_married	-0.04	0.68	0.16	0.11	1.00	0.16	0.00	0.15	0.34	0.11
work_type	-0.04	0.12	0.00	0.00	0.16	1.00	-0.02	0.01	0.21	0.01
Residence_type	0.00	0.01	0.00	0.00	0.00	-0.02	1.00	-0.01	0.00	0.01
avg_glucose_level	0.05	0.24	0.18	0.15	0.15	0.01	-0.01	1.00	0.18	0.14
bmi	-0.03	0.33	0.17	0.04	0.34	0.21	0.00	0.18	1.00	0.04
stroke	0.01	0.23	0.14	0.14	0.11	0.01	0.01	0.14	0.04	1.00

`Age` and `ever_married` appear to be highly correlated. Since `ever_married` has a lower correlation with `stroke` than `age`, we will remove `ever_married`.

```r
#remove ever_married
data <- data[,-5]
```

```r
#split data into 80% train and 20% test
index <- sample(1:nrow(data),nrow(data)*.8,replace = FALSE)

train <- data[index,]
test <- data[-index,]
```

```r
#create a label for the two classes of stroke and no stroke
classes <- c(0:1)
```

Gaussian Classifier

```r
# Create data frame to store the means of each class
means <- data.frame(matrix(NA,
                              nrow=(ncol(data)-1),
                              ncol=length(classes)))


rownames(means) <- colnames(data[,-9])
colnames(means) <- classes

#Store mean of each class for each feature
for(i in classes){
    means[,i+1] <- colMeans(train[train$stroke==i,-9])
}
```

```r
#create a list to store the two covariance matrices for each class
Cov <- list()

#create covariance matrices
for(i in classes) {
  temp_cov <- matrix(0,
                     nrow = ncol(data)-1,
                     ncol = ncol(data)-1)
  for(j in 1:sum(train$stroke==i)) {
    mat <- matrix(as.numeric(train[train$stroke==i, -9][j,] - means[,i+1]),
                  nrow = 8, ncol = 1, byrow=TRUE)
    temp_cov <- temp_cov + (mat %*% t(mat))
  }
  Cov[[i+1]] <- (1/sum(train$stroke==i)) * temp_cov
}
```

```r
# Store train data distances to mean point for each class
# Using Mahalanobis distance

mahalanobis_dist <- data.frame(matrix(NA,
                               nrow=nrow(train),
                               ncol=length(classes)))
colnames(mahalanobis_dist) <- classes
```

```r
for(i in classes) {
  for(j in 1:nrow(train)) {
    mat <- matrix(as.numeric(train[j, -9] - means[,i+1]),
                  nrow = 8, ncol = 1, byrow=TRUE)
    mahalanobis_dist[j,i+1] <- t(mat) %*% solve(Cov[[i+1]]) %*% mat
  }
}
```

```r
train_pred <- data.frame(matrix(NA,
                                nrow = nrow(train),
                                ncol = length(classes)))

# Store probabilities from multivariate normal probability function
for(i in classes) {
  train_pred[,i+1] <- ((1/sqrt(det(2*pi*Cov[[i+1]])))*exp(-0.5*mahalanobis_dist[,i+1]))
}


pred_class_train <- c()

# Store predicted class of train data based on class with highest probability
for(i in 1:nrow(train)) {
  pred_class_train <- c(pred_class_train, as.numeric(which.max(train_pred[i,])))
}

train$class <- ifelse(train$stroke == 0,1,2)
cat(c("The misclassification error for the train data is: ",
      round(mean((train$class != pred_class_train)^2), 7)))
```

```
## The misclassification error for the train data is:  0.2294372
```

```r
# Store test data distances to mean point for each class
# Using Mahalanobis distance
mahalanobis_dist_test <- data.frame(matrix(NA,
                                nrow=nrow(test),
                                ncol=length(classes)))
colnames(mahalanobis_dist_test) <- classes

for(i in classes) {
  for(j in 1:nrow(test)) {
    #using means and covariance matrices from training data
    mat <- matrix(as.numeric(test[j, -9] - means[,i+1]),
                  nrow = 8, ncol = 1, byrow=TRUE)
    mahalanobis_dist_test[j,i+1] <- t(mat) %*% solve(Cov[[i+1]]) %*% mat
  }
}

test_pred <- data.frame(matrix(NA,
                                nrow = nrow(test),
                                ncol = length(classes)))

# Store probabilities from multivariate normal probability function
```

```
for(i in classes) {
  test_pred[,i+1] <- ((1/sqrt(det(2*pi*Cov[[i+1]])))*exp(-0.5*mahalanobis_dist_test[,i+1]))
}

pred_class_test <- c()


# Store predicted class of test data based on class with highest probability
for(i in 1:nrow(test)) {
  pred_class_test <- c(pred_class_test, which.max(test_pred[i,]))
}

test$class <- ifelse(test$stroke == 0,1,2)
cat(c("The misclassification error for the test data is: ",
      round(mean((test$class != pred_class_test)^2), 7)))
```

```
## The misclassification error for the test data is:  0.2179226
```

The class distribution is highly unbalanced; there are for more many observations belonging to the Class 0 (no stroke) than to the Class 1 (stroke). I am curious if we can get better results by addressing this. We will re-balance the classes by under-sampling the observations that belong to Class 0.

```
data_stroke <- data[data$stroke==1,]
data_nostroke <- data[data$stroke==0,]
ind <- sample(1:nrow(data_nostroke),sum(data$stroke),replace = FALSE)
data_nostroke_reduced <- data_nostroke[ind,]
data_balanced <- rbind(data_stroke,data_nostroke_reduced)
```

```
#split data into 80% train and 20% test

index2 <- sample(1:nrow(data_balanced),nrow(data_balanced)*.8,replace = FALSE)

train <- data_balanced[index2,]
test <- data_balanced[-index2,]
```

The rest of the Gaussian Classifier is done similarly to before, so I shall omit the code for the sake of brevity.

```
test$class <- ifelse(test$stroke == 0,1,2)
cat(c("The misclassification error for the test data is: ",
      round(mean((test$class != pred_class_test)^2), 7)))
```

```
## The misclassification error for the test data is:  0.2738095
```

This does not appear to do any better. Let us go back to the previously defined `test` and `train` data.

```
#reset the test and train data to what they were before
train <- data[index,]
test <- data[-index,]
```

Logistic Regression

```r
#remove class column that was created with the Gaussian classifier
train <- train[,-10]
test <- test[,-10]
```

```r
# Create vectors to store betas from each bootstrap
beta <- data.frame(matrix(NA, nrow=100, ncol=9))
colnames(beta) <- c("intercept", colnames(train[,-9]))
# 100 bootstrap iterations
for(i in 1:100) {
  dat_index <- sample(1:nrow(train), nrow(train), replace = TRUE)
  dat <- data[dat_index, ]
  # Separate into X matrix (with intercept column) and Y matrix
  X <- cbind(intercept = rep(1,nrow(dat)), dat[,1:8])
  Y <- dat[,9]
  # Initial Assignments
  beta_old <- rep(0, ncol(X))
  W <- diag(nrow = nrow(X))
  mat_x <- as.matrix(X, nrow = ncol(X),ncol = nrow(X),byrow = TRUE)
  p_x <- c()
  for(j in 1:nrow(X)) {
    p_x[j] <- exp(t(beta_old) %*% mat_x[j,])/(1 + exp(t(beta_old) %*% mat_x[j,]))
  }
  z <- mat_x %*% beta_old + solve(W) %*% (Y-p_x)
  beta_new <- solve(t(mat_x) %*% W %*% mat_x) %*% t(mat_x) %*% W %*% z
  # While any of the betas are not within 0.000000001 of the previous beta
  while(any(abs(beta_new-beta_old) >= 0.000000001)) {
  # new beta from last iteration becomes old bets
  beta_old <- beta_new
  # Find probabilities
  p_x <- c()
  for(j in 1:nrow(X)) {
    p_x[j] <- exp(t(beta_old) %*% mat_x[j,])/(1 + exp(t(beta_old) %*% mat_x[j,]))
  }
  # Find weights matrix
  W <- diag(p_x * (1-p_x))
  #Solve for z: adjusted response
  z <- mat_x %*% beta_old + solve(W) %*% (Y-p_x)
  # Solve for new beta using iteratively reweighted least squares
  beta_new <- solve(t(mat_x) %*% W %*% mat_x) %*% t(mat_x) %*% W %*% z
  }
  beta[i,] <- beta_new
}
mean_beta <- apply(beta, 2, mean)
var_beta <- apply(beta, 2, var)
bootstrap <- data.frame("Mean" = round(mean_beta,8),
"Variance" = round(var_beta,8),
check.names = FALSE)
```

```r
train_pred2 <- c()

# Store probabilities of having a stroke with logistic regression equation
for(j in 1:nrow(train)){
  train_pred2[j] <- exp(bootstrap$Mean[1]+ bootstrap$Mean[2]*train[j,1]+bootstrap$Mean[3]*train[j,2]+
                        bootstrap$Mean[4]*train[j,3]+bootstrap$Mean[5]*train[j,4]+
                          bootstrap$Mean[6]*train[j,5]+
                        bootstrap$Mean[7]*train[j,6]+bootstrap$Mean[8]*train[j,7]+
                          bootstrap$Mean[9]*train[j,8])/
                      (1 + exp(bootstrap$Mean[1]+ bootstrap$Mean[2]*train[j,1]+
                                 bootstrap$Mean[3]*train[j,2]+
                        bootstrap$Mean[4]*train[j,3]+bootstrap$Mean[5]*train[j,4]+
                          bootstrap$Mean[6]*train[j,5]+
                        bootstrap$Mean[7]*train[j,6]+bootstrap$Mean[8]*train[j,7]+
                          bootstrap$Mean[9]*train[j,8]))
}

# Create vector to store predicted class for train data based on probabilities
pred_class_train2 <- c()


# Store predicted class of test data based on class with highest probability
pred_class_train2 <- ifelse(train_pred2 > 0.5,1,0)


cat(c("The misclassification error for the train data is: ",
      round(mean((train$stroke != pred_class_train2)^2), 7)))
```

```
## The misclassification error for the train data is:  0.0448179
```

```r
test_pred2 <- c()

# Store probabilities of having a stroke with logistic regression equation
for(j in 1:nrow(test)){
  test_pred2[j] <- exp(bootstrap$Mean[1]+bootstrap$Mean[2]*test[j,1]+bootstrap$Mean[3]*test[j,2]+
                        bootstrap$Mean[4]*test[j,3]+bootstrap$Mean[5]*test[j,4]+
                         bootstrap$Mean[6]*test[j,5]+
                         bootstrap$Mean[7]*test[j,6]+bootstrap$Mean[8]*test[j,7]+
                         bootstrap$Mean[9]*test[j,8])/
                    (1 + exp(bootstrap$Mean[1]+ bootstrap$Mean[2]*test[j,1]+
                    bootstrap$Mean[3]*test[j,2]+ bootstrap$Mean[4]*test[j,3]+
                      bootstrap$Mean[5]*test[j,4]+
                      bootstrap$Mean[6]*test[j,5]+bootstrap$Mean[7]*test[j,6]+
                      bootstrap$Mean[8]*test[j,7]+
                      bootstrap$Mean[9]*test[j,8]))
}
```

```r
# Create vector to store predicted class for test data based on probabilities
pred_class_test2 <- c()


# Store predicted class of test data based on class with highest probability
pred_class_test2 <- ifelse(test_pred2 > 0.5,1,0)


cat(c("The misclassification error for the test data is: ",
      round(mean((test$stroke != pred_class_test2)^2), 7)))
```

```
## The misclassification error for the test data is:  0.0336049
```

Conclusion

While the Gaussian Classifier wasn't bad, Logistic Regression performed very well and produced very low misclassification error rates.