HW6

2025-03-25

Setup 1

Consider a hypothetical situation where a financial company wants to model credit card default based on a customer's age, balance, and income. Suppose X_1 denotes a random vector containing the age, balance, and income of a default customer. Similarly, X_2 is defined for a non-default customer. From their previous data, the company finds the following distributions:

$$X_1 \sim N_3(\mu_1, \Sigma_1), \text{ and } X_2 \sim N_3(\mu_2, \Sigma_2),$$

where

$$\mu_1 = \begin{pmatrix} 38 \\ 1.75 \\ 32 \end{pmatrix}, \quad \mu_2 = \begin{pmatrix} 39 \\ 0.80 \\ 34 \end{pmatrix},$$

$$\Sigma_1 = \begin{pmatrix} 5.5 & 0.50 & 1 \\ 0.50 & 0.12 & -0.75 \\ 1 & -0.75 & 200 \end{pmatrix}, \quad \Sigma_2 = \begin{pmatrix} 4 & 0.05 & 0.1 \\ 0.05 & 0.21 & -0.98 \\ 0.1 & -0.98 & 170 \end{pmatrix}.$$

Here balance and income are measured in \$1,000 scale. The company estimated that 5% of their customers are default.

Questions

Generate a data set of size 10,000 (default and non-default combined) from the above setup. Apply different classification algorithms (Logistic Regression, LDA, QDA, KNN-5, KNN-10, and Naive Bayes) and compare their performance both in training and test data. For the test data, generate 100,000 samples.

Answers

Set up parameters:

```
mu_1 = t(c(38, 1.75, 32))
mu_2 = t(c(39, 0.80, 34))

sigma_1 = matrix(NA, 3, 3)
sigma_1[1,1] = 5.5; sigma_1[2,2] = 0.12; sigma_1[3,3] = 200
sigma_1[1,2] = sigma_1[2,1] = 0.50
sigma_1[1,3] = sigma_1[3,1] = 1
sigma_1[2,3] = sigma_1[3,2] = -0.75
sigma_1
```

```
##      [,1]  [,2]   [,3]
## [1,]  5.5  0.50   1.00
## [2,]  0.5  0.12  -0.75
## [3,]  1.0 -0.75 200.00
```

```
sigma_2 = matrix(NA, 3, 3)
sigma_2[1,1] = 4; sigma_2[2,2] = 0.21; sigma_2[3,3] = 170
sigma_2[1,2] = sigma_2[2,1] = 0.05
sigma_2[1,3] = sigma_2[3,1] = 0.1
sigma_2[2,3] = sigma_2[3,2] = -0.98
sigma_2
```

```
##      [,1]  [,2]   [,3]
## [1,] 4.00  0.05   0.10
## [2,] 0.05  0.21  -0.98
## [3,] 0.10 -0.98 170.00
```

Create training and test data

```
library(mvtnorm)
X1 = rmvnorm(n = 500, mean = mu_1, sigma = sigma_1)
X2 = rmvnorm(n = 9500, mean = mu_2, sigma = sigma_2)

X = rbind(X1,X2)
Y = rep(c("Yes", "No"), c(500, 9500))

training_data = data.frame(default = Y, age = X[,1], balance = X[,2], income = X[,3])
training_data$default = as.factor(training_data$default)

X1 = rmvnorm(n = 5000, mean = mu_1, sigma = sigma_1)
X2 = rmvnorm(n = 95000, mean = mu_2, sigma = sigma_2)

X = rbind(X1,X2)
Y = rep(c("Yes", "No"), c(5000, 95000))

test_data = data.frame(default = Y, age = X[,1], balance = X[,2], income = X[,3])
test_data$default = as.factor(test_data$default)
```

Create placeholder for results

```
classification_report = matrix(NA, 2, 6)
rownames(classification_report) = c("Training", "Test")
colnames(classification_report) = c("log_reg", "LDA", "QDA", "KNN(k=5)", "KNN(k=10)", "Naive Bayes")
```

Logistic Regression

```
log_reg = glm(default ~ ., data = training_data, family = "binomial")
log_reg_prob_train = predict(log_reg, type = "response")
log_reg_pred_train = ifelse(log_reg_prob_train > 0.5, "Yes", "No")
table(log_reg_pred_train, training_data$default)
```

```
##
## log_reg_pred_train   No  Yes
##                No  9377  292
##                Yes  123  208
```

```
train_accuracy = mean(log_reg_pred_train == training_data$default)*100
classification_report[1,1] = train_accuracy

log_reg_prob_test = predict(log_reg, data = test_data, type = "response")
log_reg_pred_test = ifelse(log_reg_prob_test > 0.5, "Yes", "No")
test_accuracy = mean(log_reg_pred_test == test_data$default)*100
classification_report[2,1] = test_accuracy
```

LDA

```
library(MASS)
lda_fit = lda(default ~ ., data = training_data)
lda_pred_train = predict(lda_fit)
classification_report["Training", "LDA"] = mean(lda_pred_train$class == training_data$default)*100

lda_pred_test = predict(lda_fit, newdata = test_data)
classification_report["Test", "LDA"] = mean(lda_pred_test$class == test_data$default)*100
```

QDA

```
qda_fit = qda(default ~ ., data = training_data)
qda_pred_train = predict(qda_fit)
classification_report["Training", "QDA"] = mean(qda_pred_train$class == training_data$default)*100

qda_pred_test = predict(qda_fit, newdata = test_data)
classification_report["Test", "QDA"] = mean(qda_pred_test$class == test_data$default)*100
```

KNN with k=5

```
library(class)
knn_5_fit = knn(train = training_data[,-1], test = training_data[,-1], cl = training_data$default, k = 5
classification_report["Training", "KNN(k=5)"] = mean(knn_5_fit == training_data$default)*100

knn_5_fit = knn(train = training_data[,-1], test = test_data[,-1], cl = training_data$default, k = 5)
classification_report["Test", "KNN(k=5)"] = mean(knn_5_fit == test_data$default)*100
```

KNN with k=10

```
library(class)
knn_10_fit = knn(train = training_data[,-1], test = training_data[,-1], cl = training_data$default, k =
```

```r
classification_report["Training", "KNN(k=10)"] = mean(knn_10_fit == training_data$default)*100

knn_10_fit = knn(train = training_data[,-1], test = test_data[,-1], cl = training_data$default, k = 10)
classification_report["Test", "KNN(k=10)"] = mean(knn_10_fit == test_data$default)*100
```

Naive Bayes

```r
library(e1071)
nb_fit = naiveBayes(default ~ ., data = training_data)
nb_pred_train = predict(nb_fit, newdata = training_data)
classification_report["Training", "Naive Bayes"] = mean(nb_pred_train == training_data$default)*100

nb_pred_test = predict(nb_fit, newdata = test_data)
classification_report["Test", "Naive Bayes"] = mean(nb_pred_test == test_data$default)*100
```

Print result

```r
classification_report
```

```
##          log_reg   LDA    QDA KNN(k=5) KNN(k=10) Naive Bayes
## Training  95.850 95.94 96.000   96.120    95.470      95.990
## Test      92.228 95.91 96.158   95.085    95.062      95.935
```

Conclusion

The QDA performs a little better than others. As the covariance matrices are different in two classes, it violated the assumption of LDA, logistic regression and Naive Bayes.

Setup 2

Suppose $X = (X_1, X_2, X_3, X_4)^T$ denotes a random vector containing four input variables. Assume $X \sim N_4(\mu, \Sigma)$, where

$$\mu = \begin{pmatrix} 35 \\ 1.75 \\ 32 \\ -5 \end{pmatrix}, \quad \Sigma = \begin{pmatrix} 5.5 & 0.50 & 1 & 0 \\ 0.50 & 0.12 & -0.75 & 0.1 \\ 1 & -0.75 & 100 & 0 \\ 0 & 0.1 & 0 & 50 \end{pmatrix}.$$

Generate a sample of size 1,000 from $X. Then, calculate the probability vector $p(x)$ using the following formula:

$$p(x) = \frac{\exp(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4)}{1 + \exp(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4)},$$

where $\beta = (\beta_0, \beta_1, \beta_2, \beta_3, \beta_4)^T = (3, 0.5, 0, -0.6, 1)^T$. Now, generate a set of 1,000 Bernoulli random variable using the success probability as $p(x)$. This is the response variable Y for the classification problem. So, you have a training data of size $n = 1000$ from (Y, X).

Questions

Apply different classification algorithms (Logistic Regression, LDA, QDA, KNN-5, KNN-10, and Naive Bayes) and compare their performance both in training and test data. For the test data, generate 10,000 samples.

Note: For the logistics regression, you may get a warning message as "Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred". This is because some fitted probability values are very close to 0 or 1. So, it is not necessarily a convergence issue.

Answers

Setting up parameters:

```
n <- 1000
p <- 4
beta <- c(3, 0.5, 0, -0.6, 1)

library(mvtnorm)
mu_vec <- c(35, 1.75, 32, -5)


Sigma <- matrix(0, 4, 4)
diag(Sigma) <- c(5.5, 0.12, 100, 50)
Sigma[1, 2] <- Sigma[2, 1] <- 0.5
Sigma[1, 3] <- Sigma[3, 1] <- 1
Sigma[2, 3] <- Sigma[3, 2] <- -0.75
Sigma[4, 2] <- Sigma[2, 4] <- 0.1
Sigma
```

```
##      [,1]  [,2]   [,3] [,4]
## [1,]  5.5  0.50   1.00  0.0
## [2,]  0.5  0.12  -0.75  0.1
## [3,]  1.0 -0.75 100.00  0.0
## [4,]  0.0  0.10   0.00 50.0
```

Create training and test data

```
X <- rmvnorm(n = n, mean = mu_vec, sigma = Sigma)
p_x <- 1/(1 + exp(-(beta[1] + X %*% beta[-1])))
Y <- rbinom(n = n, size = 1, prob = p_x)

data_training <- data.frame(Y, X)
colnames(data_training) <- c("Y", paste0("X", 1:p))

# Test data
n_test <- 10000
X_test <- rmvnorm(n = n_test, mean = mu_vec, sigma = Sigma)
p_x_test <- 1/(1 + exp(-(beta[1] + X_test %*% beta[-1])))
Y_test <- rbinom(n = n_test, size = 1, prob = p_x_test)

data_test <- data.frame(Y_test, X_test)
colnames(data_test) <- c("Y", paste0("X", 1:p))
```

Create placeholder for results

```
Classification_Error <- matrix(NA, 2, 6)
rownames(Classification_Error) <- c("Training", "Test")
colnames(Classification_Error) <- c("Logistic", "LDA", "QDA", "KNN-5", "KNN-10",
  "Naive Bayes")
```

Logistic regression

```
logistic.fit <- glm(Y ~ ., data = data_training, family = "binomial")
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
logistic.predict <- predict(logistic.fit, type = "response")
logistic.predict.y <- ifelse(logistic.predict > 0.5, 1, 0)

Classification_Error["Training", "Logistic"] <- mean(logistic.predict.y != data_training$Y)
logistic.predict <- predict(logistic.fit, type = "response", newdata = data_test)
logistic.predict.y <- ifelse(logistic.predict > 0.5, 1, 0)
Classification_Error["Test", "Logistic"] <- mean(logistic.predict.y != data_test$Y)
```

LDA

```
library(MASS)
lda.fit <- lda(Y ~ ., data = data_training)

lda.predict <- predict(lda.fit)
Classification_Error["Training", "LDA"] <- mean(lda.predict$class != data_training$Y)

lda.predict <- predict(lda.fit, newdata = data_test)
Classification_Error["Test", "LDA"] <- mean(lda.predict$class != data_test$Y)
```

QDA

```
qda.fit <- qda(Y ~ ., data = data_training)
qda.predict <- predict(qda.fit)
Classification_Error["Training", "QDA"] <- mean(qda.predict$class != data_training$Y)

qda.predict <- predict(qda.fit, newdata = data_test)
table(qda.predict$class, data_test$Y)
```

```
##
##        0    1
##   0 6247  310
##   1  343 3100
```

```
Classification_Error["Test", "QDA"] <- mean(qda.predict$class != data_test$Y)
```

KNN with k=5

```
knn5.fit <- knn(train = data_training[, -1], test = data_training[, -1], cl = Y,
                k = 5)
Classification_Error["Training", "KNN-5"] <- mean(knn5.fit != data_training$Y)

knn5.fit <- knn(train = data_training[, -1], test = data_test[, -1], cl = Y, k = 5)
Classification_Error["Test", "KNN-5"] <- mean(knn5.fit != data_test$Y)
```

KNN with k=10

```
knn10.fit <- knn(train = data_training[, -1], test = data_training[, -1], cl = Y,
                 k = 10)
Classification_Error["Training", "KNN-10"] <- mean(knn10.fit != data_training$Y)

knn10.fit <- knn(train = data_training[, -1], test = data_test[, -1], cl = Y, k = 10)
Classification_Error["Test", "KNN-10"] <- mean(knn10.fit != data_test$Y)
```

Naive Bayes

```
NBayes.fit <- naiveBayes(Y ~ ., data = data_training)
NBayes.predict <- predict(NBayes.fit, newdata = data_training)
Classification_Error["Training", "Naive Bayes"] <- mean(NBayes.predict != data_training$Y)

NBayes.predict <- predict(NBayes.fit, newdata = data_test)
Classification_Error["Test", "Naive Bayes"] <- mean(NBayes.predict != data_test$Y)
```

Print result

```
Classification_Error
```

```
##          Logistic    LDA    QDA KNN-5 KNN-10 Naive Bayes
## Training   0.0430 0.0480 0.0470 0.042 0.0500      0.0780
## Test       0.0596 0.0624 0.0653 0.074 0.0704      0.0804
```

Conclusion

Based on all outputs, the LDA and the logistic regression perform a little better than others. The setup is as per the assumptions of the logistic regression and the LDA.

Setup 3

Load the iris data from the R datasets package. This famous (Fisher's or Anderson's) iris data set gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are Iris setosa, versicolor, and virginica.

Questions

Use a pair plot of the numeric data and label observations by species. Fit a multinomial logistic regression and LDA using the full data and compare their performance.

Answer

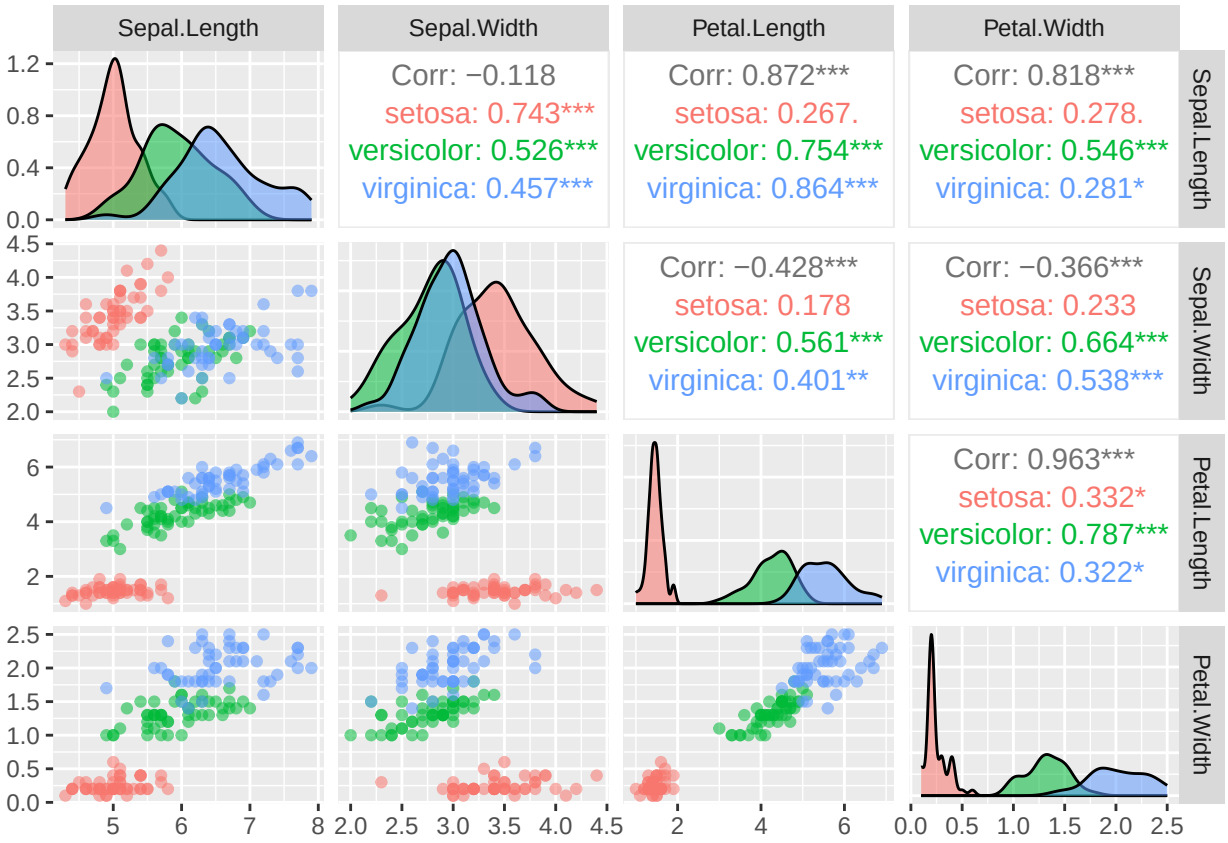
```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
library(ggplot2)
library(GGally)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
ggpairs(iris, columns = 1:4, aes(colour=Species, alpha = 0.5))
```



Multinomial Logistics Regression

```
library(nnet)
iris$Species <- relevel(iris$Species, ref = "setosa")
mlr = multinom(Species ~ ., data = iris)
```

```
## # weights:  18 (10 variable)
## initial  value 164.791843
## iter  10 value 16.177348
## iter  20 value 7.111438
## iter  30 value 6.182999
## iter  40 value 5.984028
## iter  50 value 5.961278
## iter  60 value 5.954900
## iter  70 value 5.951851
## iter  80 value 5.950343
## iter  90 value 5.949904
## iter 100 value 5.949867
## final  value 5.949867
## stopped after 100 iterations
```

```
logistic.predict <- predict(mlr)

table(logistic.predict, iris$Species)
```

```
##
## logistic.predict setosa versicolor virginica
##       setosa         50          0         0
##       versicolor      0         49         1
##       virginica       0          1        49
```

```
(accuracy_mlr = mean(logistic.predict == iris$Species))
```

```
## [1] 0.9866667
```

LDA

```
iris.lda <- lda(Species ~ ., data = iris)
lda.predict <- predict(iris.lda)
mean(lda.predict$class == iris$Species)
```

```
## [1] 0.98
```

Setup 4

The following questions are based on the credit card default data mentioned in the textbook (see the attached file).

Questions

Use the full data to fit a logistic model. Take different threshold values to predict the default status from the fitted probability. Then, calculate the Cohen's Kappa statistic (use the following definition, not a built-in function like confusionMatrix), and plot them against the threshold values. Also, draw the ROC curve preferably using your own function.

The definition of Cohen's Kappa statistic:

$$\kappa = \frac{2 \times (TP \times TN - FN \times FP)}{(TP + FP) \times (FP + TN) + (TP + FN) \times (FN + TN)},$$

where TP are the true positives, FP are the false positives, TN are the true negatives, and FN are the false negatives.

Answers

Load data

```
default = read.csv("/Users/atanugiri/OneDrive - University of Texas at El Paso/Class Documents/Data Mini
head(default)
```

```
##   default student   balance    income
## 1      No      No  729.5265 44361.625
## 2      No     Yes  817.1804 12106.135
## 3      No      No 1073.5492 31767.139
## 4      No      No  529.2506 35704.494
## 5      No      No  785.6559 38463.496
## 6      No     Yes  919.5885  7491.559
```

```
default$default = as.factor(default$default)
default$student = as.factor(default$student)
summary(default)
```

```
##  default    student       balance           income
##  No :9667   No :7056   Min.   :   0.0   Min.   :  772
##  Yes: 333   Yes:2944   1st Qu.: 481.7   1st Qu.:21340
##                        Median : 823.6   Median :34553
##                        Mean   : 835.4   Mean   :33517
##                        3rd Qu.:1166.3   3rd Qu.:43808
##                        Max.   :2654.3   Max.   :73554
```

Logistic regression

```
log_fit = glm(default ~., data = default, family = "binomial")
log_fit_prob = predict(log_fit, type = "response")
```

Function to extract parameters

```
stats_summary = function(thr) {
  log_fit_prob = ifelse(log_fit_prob > thr, "Yes", "No")
  log_fit_prob = as.factor(log_fit_prob)

  con_matrix = table(log_fit_prob, default$default)

  if (nrow(con_matrix) == 1) {
    if (rownames(con_matrix) == "Yes") {
      con_matrix = rbind(con_matrix, c(0,0))
      rownames(con_matrix) = c("Yes", "No")
    } else {
      con_matrix = rbind(c(0,0), con_matrix)
      rownames(con_matrix) = c("Yes", "No")
    }
  }

  TP = con_matrix[1,1]; FN = con_matrix[2,1]; FP = con_matrix[1,2]; TN = con_matrix[2,2]

  kappa_num = 2*(TP*TN - FN*FP)
  kappa_den = (TP+FP)*(FP+TN) + (TP+FN)*(FN+TN)

  kappa = kappa_num/kappa_den

  TPR = TP/(TP+FN); FPR = FP/(FP+TN)
```

```
  return(list(kappa = kappa, TPR = TPR, FPR = FPR))
}
```

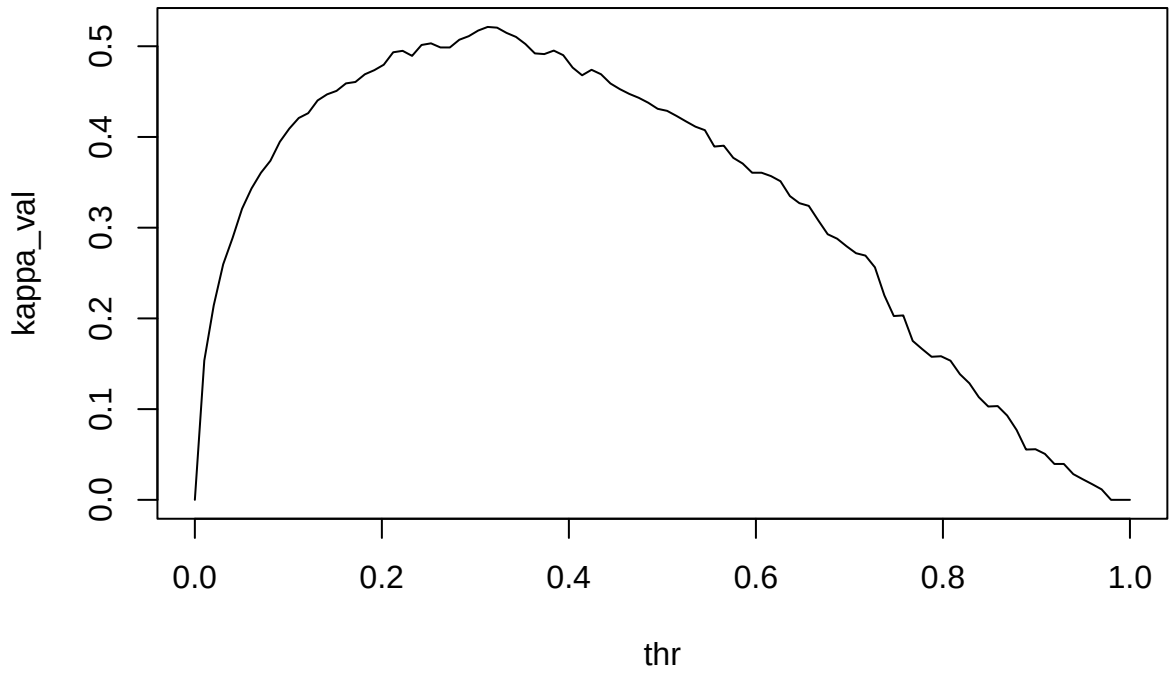
Placeholder for result

```
thr = seq(0,1,length.out=100)
kappa_val = TPR_val = FPR_val = numeric(0)
```

```
for (i in 1:length(thr)) {
  result = stats_summary(thr[i])
  kappa_val[i] = result$kappa
  TPR_val[i] = result$TPR
  FPR_val[i] = result$FPR
}
```

Kappa statistic plot

```
plot(thr, kappa_val, type = "l")
```




```
(Threshold_max = thr[which.max(kappa_val)])
```

```
## [1] 0.3131313
```

ROC curve

```
plot(FPR_val, TPR_val)
abline(a = 0, b = 1, lty = 2)
```