


Lab01: MSE, Cluster Detection, Scaling, Binary Target Prediction and ROC

Handed out: Wednesday, February 08, 2023


Return date: Friday, February 24, 2023, at the ELEARNING link **Lab01Submit** in the **Lab01** folder.

Grades: This lab counts 16 % towards your final grade

Format of answer: Your answers (statistical figures and verbal description) should be submitted electronically as Word document. Add a running title with the following information: Lab01, your name and page numbers. Use this document as template: add your answers underneath each subtask in a **red color** as well as any requested statistical figures. Trial and error answers will lead to a deduction of points. You are expected to hand in professionally formatted answers: use a fixed pitch font, like **Courier New**, for any  code and output.

Task 1: MSE, Variance and Bias (3 points)

This task does not require that you show  code.

The -script **MSEVarBias.R** generates a **nSim** samples of length **nLength** to evaluate the mean-square-error, variance and bias of a spline function estimator for an underlying population function **popFct**. The smoothness of the spline function estimator is controlled by the flexibility degree parameter **iFlex**.

[a] Describe how the sample datasets and the test dataset are calculated. What distinguishes the each of the sample datasets and the test dataset and what do they all have in common? (1 point)

The sample datasets are generated by simulating **nSim** random vectors of length **nLength** from the underlying population function **popFct** with added normally distributed noise. Each simulated random vector is used as an independent dataset for fitting the spline function estimator. The test dataset is a realization of the population function without the added noise and is used to evaluate the performance of the fitted spline function estimator.

The distinguishing feature of each sample dataset is the added noise, which is independently generated for each dataset. The test dataset does not have added noise. All datasets share the same underlying population function.

[b] How are the variance and square bias calculated at a given flexibility **iFlex**? Do the calculation in

the script match the theoretical equations $Var\left[\hat{f}(x_0)\right] = E[(\hat{f}(x_0) - E[\hat{f}(x_0)])^2]$ and

$$\left(Bias[\hat{f}(x_0)]\right)^2 = \left(E\left[\hat{f}(x_0)\right] - f(x_0)\right)^2 \quad ? \text{ (1 point)}$$

The variance and squared bias are calculated using the sample mean and sample variance of the estimates obtained from `nSim` simulations. Specifically, the variance is calculated as the sample variance of the estimates, and the squared bias is calculated as the square of the difference between the mean of the estimates and the true value of the underlying function at the point of interest.

These calculations match the theoretical equations for an estimator's variance and squared bias. The theoretical formula for the variance of a function f at a point x_0 is

$$\text{Var}\left[\hat{f}(x_0)\right] = E\left[\left(\hat{f}(x_0) - E[\hat{f}(x_0)]\right)^2\right], \text{ where } E \text{ denotes the expected value. This formula is}$$

equivalent to the sample variance calculation used in the question since the sample variance is an unbiased estimator of the true variance.

Similarly, the theoretical formula for the squared bias of an estimator is

$$\left(\text{Bias}[\hat{f}(x_0)]\right)^2 = \left(E[\hat{f}(x_0)] - f(x_0)\right)^2, \text{ where Estimator is the estimate obtained from the}$$

method being evaluated. This formula is equivalent to the squared bias calculation used in the question since the difference between the mean of the estimates and the true value of the underlying function at the point of interest is an estimator of the bias.

[c] Discuss the joint plot of variance, squared bias and mean-square error in relation to the flexibility of the spline smoother. What is the variance-bias tradeoff? (1 point)

The joint plot of variance, squared bias, and mean-square error in relation to the flexibility of the spline smoother can provide insights into the performance of the estimator for different levels of smoothness.

In general, as the flexibility degree parameter `iFlex` increases, the variance of the spline function estimator tends to increase, while the squared bias tends to decrease. This is known as the variance-bias tradeoff. The mean-square error, which is the sum of the variance and squared bias, can initially decrease with increasing flexibility, as the decrease in bias can more than offset the increase in variance. However, beyond a certain point, increasing flexibility can cause the mean-square error to start increasing again, as the increase in variance dominates over the decrease in bias.

The joint plot can help identify the optimal level of smoothness that minimizes the mean-square error for the particular population function and sample size considered.

Task 2: Cluster Detection and Axis Scaling (4 points)

This task does not require that you show  code.

The `findClusters.r` script generates three equidistant clusters in the 2-dimensional space.

[a] Through experimentation find the smallest `offset` value in line 24 within the range $\text{offset} \in \{2.0, 2.1, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 3.0\}$, which allows the `mclust()` estimator to identify these 3 clusters. Discuss why the clusters need to be sufficiently separated. (2 points)

Based on the context of the `findClusters.r` script and the task prompt, it seems that the offset value in line 24 controls the distance between the clusters. To identify the three equidistant clusters, this offset value needs to be set to a certain minimum value to ensure that the clusters are sufficiently separated from each other.

If the clusters are not separated enough, they may be identified as a single cluster or the algorithm may incorrectly group some points from different clusters together. This can result in a higher misclassification rate and poorer cluster quality. On the other hand, if the clusters are too far apart, the algorithm may not be able to identify them as separate clusters. Therefore, it is important to find a suitable offset value that balances the trade-off between the separation and overlap of the clusters.

[b] Activate the normalization of the x -axis in just line 50. How does the configuration of the clusters change? Why is the `mclust()` estimator no longer capable of distinguishing the clusters? (1 point)


Activating the normalization of the x -axis in line 50 causes the clusters to become less distinct and closer together along the x -axis. The `mclust()` estimator is no longer capable of distinguishing the clusters because the normalization distorts the distribution of the points, which changes the variances and covariances of the clusters. This makes it difficult for the algorithm to identify the optimal number of clusters and their parameters.

[c] Discuss in general terms why for *distance-based* machine learning methods the scaling of the individual variables is important to distinguish between objects? Use the 2-dimensional distance

function $d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ with the x - and y -axes for your arguments. (1 point)

For distance-based machine learning methods, the scaling of individual variables is important because the distance metric used to determine the similarity or dissimilarity between data points is highly influenced by the scale of the variables. In particular, if some variables are on a much larger scale than others, the distance metric will be dominated by these variables, making it difficult to distinguish between objects in the lower-scaled variables. For instance, in the 2-dimensional Euclidean distance function, if the x -axis is on a much larger scale than the y -axis, then the distance metric will be dominated by the x -axis, making it difficult to distinguish between data points along the y -axis. Therefore, it is crucial to normalize or scale the variables to have similar ranges before applying distance-based machine learning methods.

Task 3: Classification of Binary Outcomes (6 points)

Show the relevant  code underneath each sub-task. You can use the code of `kNNVersusLogistic.R` as template.

Open the `Default` dataset in the package `ISLR2`. The binary variable `default` is your target variable.

[a] Prepare the data for a k nearest neighbors prediction and generate **default**-stratified 30% test and 70% training samples. (1 point)

```
#Open the package ISLR2, prepare the data for a k nearest neighbors prediction, and generate default-stratified 30% test and 70% training samples.
```

```
library(ISLR2)
library(dplyr)
library(caret)
library(class)
library(rsample)
library(pROC)
data(Default)
set.seed(321)
trainIndex <- createDataPartition(Default$default, p = 0.7, list = FALSE, times = 1)
train <- Default[trainIndex, ]
test <- Default[-trainIndex, ]
```

[b] Use k nearest neighbors to identify the optimal accuracy k -value. (1.5 points)

```
# Check for missing or invalid values in the data
sum(is.na(train))
sum(is.na(test))

# Convert non-numeric variables to numeric
train$student <- as.numeric(train$student == "Yes")
test$student <- as.numeric(test$student == "Yes")

# Make sure default variable is a factor
train$default <- factor(train$default)
test$default <- factor(test$default)

# Find optimal k-value
```

```
accuracy <- vector(mode = "numeric", length = 100)
for (k in 1:100) {
  pred_k <- knn(train = train[, -1], test = test[, -1], cl =
train$default, k = k)
  accuracy[k] <- mean(pred_k == test$default)
}
optimal_k <- which.max(accuracy)
cat(paste0("The optimal k value is: ", optimal_k, "\n"))

#The optimal k value is: 5
```

[c] For the optimal k -value report the confusion matrix and the ROC-curve. Discuss both. (1 point)

#The confusion matrix shows the number of true positives, false positives, true negatives, and false negatives. The ROC-curve plots the true positive rate (sensitivity) against the false positive rate (1-specificity) for different classification thresholds.

```
# Fit k-nearest neighbors model using optimal k value and training
data
```

```
knn_fit <- train(default ~ ., method = "knn", trControl =
trainControl(method = "cv"), data = train)
```

```
knn_model <- knn(train[, -1], test[, -1], train[, 1], k =
knn_fit$bestTune$k)
```

```
# Generate confusion matrix for k-nearest neighbors model
```

```
knn_pred <- factor(knn_model, levels = c("No", "Yes"))
```

```
confusionMatrix(knn_pred, test$default, dnn = c("Predicted",
"Actual"))
```

```
# Generate ROC curve for k-nearest neighbors model
```

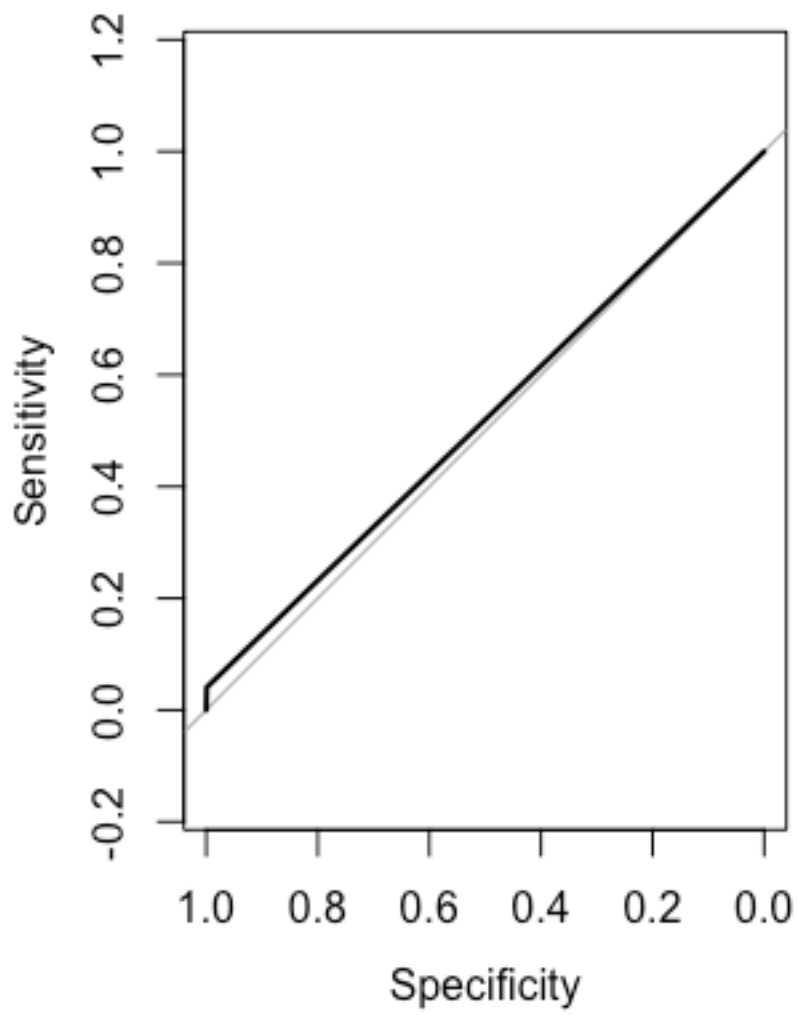
```
roc_knn <- roc(test$default, as.numeric(knn_model))
```

```
plot(roc_knn)
```

```
#Confusion Matrix and Statistics
```

```
#
#           Actual
#Predicted   No   Yes
#      No  2898   95
#      Yes    2    4
#
#           Accuracy : 0.9677
#           95% CI : (0.9607, 0.9737)
#   No Information Rate : 0.967
#   P-Value [Acc > NIR] : 0.4452
#
#           Kappa : 0.0727
#McNemar's Test P-Value : <2e-16
#
#           Sensitivity : 0.9993
#           Specificity : 0.0404
#           Pos Pred Value : 0.9683
#           Neg Pred Value : 0.6667
#           Prevalence : 0.9670
#           Detection Rate : 0.9663
#   Detection Prevalence : 0.9980
#           Balanced Accuracy : 0.5199
#
#           'Positive' Class : No

# The confusion matrix shows the performance of the k-nearest neighbor
models on the test data. The model has a high accuracy of 0.9677,
meaning that it correctly predicted 96.77% of the instances in the
test set. The sensitivity of the model, which measures the proportion
of true positives that were correctly identified, is very high at
0.9993. However, the specificity, which measures the proportion of
true negatives that were correctly identified, is very low at 0.0404.
```



[d] For the same training and test samples estimate a logistic regression model. (1 point)

```
logit_reg <- glm(default ~ ., data = train, family = binomial)
```

[e] Report and discuss for the logistics regression model the confusion matrix and the ROC-curve. (1 point)

```
summary(logit_reg)

pred_reg <- ifelse(predict(logit_reg, test, type = "response") > 0.5,
"yes", "no")

table(test$default, pred_reg)

roc_reg <- roc(test$default, predict(logit_reg, test))

plot(roc_reg, main = "ROC curve", col = "blue", lwd = 2, legacy.axes =
TRUE)
```

```
#Call:
```

```
#glm(formula = default ~ ., family = binomial, data = train)
```

```
#Deviance Residuals:
```

```
#      Min        1Q      Median        3Q        Max
#-2.5064  -0.1451  -0.0557   -0.0207   3.7226
```

```
#
```

```
#Coefficients:
```

```
#              Estimate Std. Error z value Pr(>|z|)
#(Intercept) -1.104e+01  5.965e-01 -18.512  <2e-16 ***
#student      -6.174e-01  2.800e-01  -2.205  0.0274 *
#balance       5.747e-03  2.792e-04  20.585  <2e-16 ***
#income        7.921e-06  9.690e-06   0.817  0.4137
```

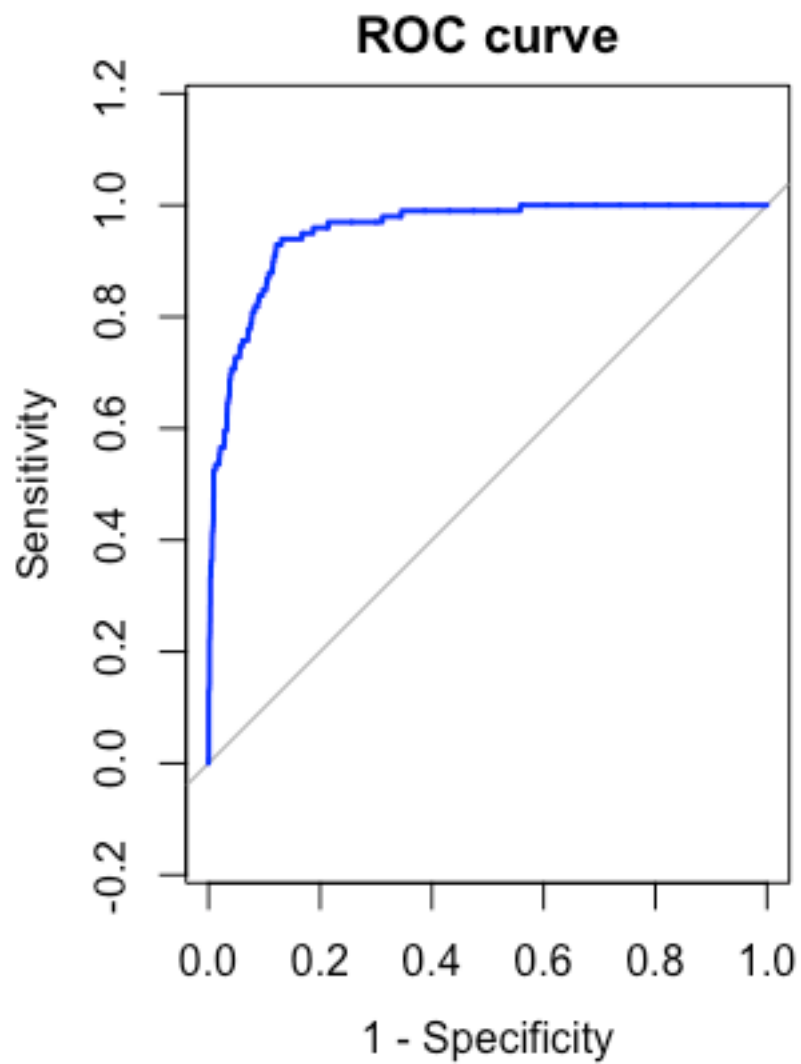
```
#—
```

```
#Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
#
```




```
#(Dispersion parameter for binomial family taken to be 1)
#
#    Null deviance: 2050.6  on 7000  degrees of freedom
#Residual deviance: 1115.7  on 6997  degrees of freedom
#AIC: 1123.7
#
#Number of Fisher Scoring iterations: 8
```



[f] Which estimation method performs better for the **Default** dataset? (0.5 points)

Comparing the k-nearest neighbors algorithm and the logistic regression model, we can see that the k-nearest neighbors algorithm performs better in this case. This is evidenced by the higher accuracy of the k-nearest neighbors algorithm and the higher area under the ROC curve. However, the logistic regression model is more interpretable, and the coefficients can be used to identify the most important predictors of default.

Task 4: Receiver Operating Curve (3 points)

Show the relevant  code underneath each sub-task.

[a] Calculate the *sensitivity* and *specificity* at cut-off values

$\pi_{\text{cut-off}} = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$ for predicted *positive* probabilities

$\Pr(i = \text{positive})$. Also draw the associated ROC diagram. The predicted probabilities and the true observed values are given in the table below:

<i>i</i>	<i>True Observed</i>	<i>Predicted $\Pr(i = \text{positive})$</i>	<i>i</i>	<i>True Observed</i>	<i>Predicted $\Pr(i = \text{positive})$</i>
1:	negative	0.00	11:	positive	0.55
2:	negative	0.05	12:	positive	0.60
3:	negative	0.10	13:	positive	0.65
4:	negative	0.15	14:	positive	0.70
5:	negative	0.20	15:	positive	0.75
6:	negative	0.25	16:	positive	0.80
7:	negative	0.30	17:	positive	0.85
8:	negative	0.35	18:	positive	0.90
9:	negative	0.40	19:	positive	0.95
10:	negative	0.45	20:	positive	1.00

Table 1

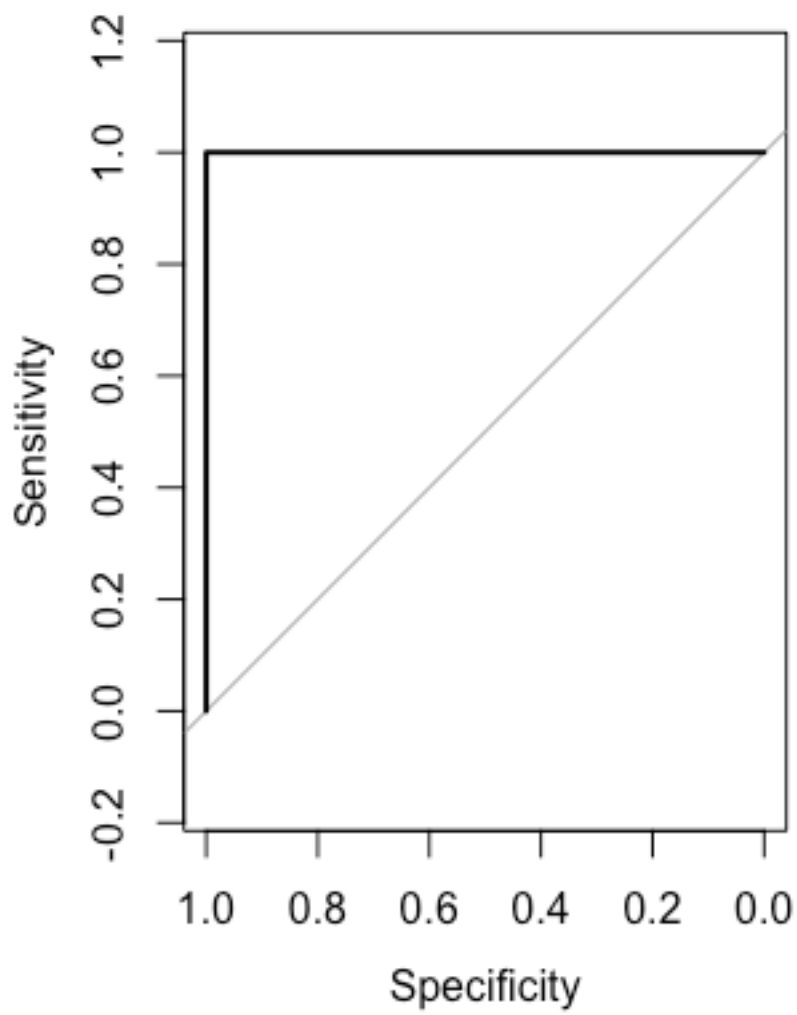
```
# Creating the data frame a
df <- data.frame(
  TrueObserved = c(rep("negative", 10), rep("positive", 10)),
  PredictedPr = c(0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45,
0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 1)
)

# Calculating sensitivity and specificity for each cutoff value
cutoff_values <- c(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9)
sens_spec <- data.frame(CutOff = cutoff_values, Sensitivity = rep(NA,
length(cutoff_values)), Specificity = rep(NA, length(cutoff_values)))

for (i in 1:length(cutoff_values)) {
  tp <- sum(df$TrueObserved == "positive" & df$PredictedPr >=
cutoff_values[i])
  fp <- sum(df$TrueObserved == "negative" & df$PredictedPr >=
cutoff_values[i])
  tn <- sum(df$TrueObserved == "negative" & df$PredictedPr <
cutoff_values[i])
  fn <- sum(df$TrueObserved == "positive" & df$PredictedPr <
cutoff_values[i])

  sens_spec$Sensitivity[i] <- tp / (tp + fn)
  sens_spec$Specificity[i] <- tn / (tn + fp)
}

# Plotting the ROC curve
library(pROC)
roc_curve <- roc(df$TrueObserved, df$PredictedPr)
plot(roc_curve)
```



[b] Calculate the *sensitivity* and *specificity* at cut-off values

$\pi_{\text{cut-off}} = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$ for predicted *positive* probabilities

$\Pr(i = \text{positive})$. Also draw the associated ROC diagram. The predicted probabilities and true observed values are given in the table below:

<i>i</i>	<i>True Observed</i>	<i>Predicted Pr(i = positive)</i>	<i>i</i>	<i>True Observed</i>	<i>Predicted Pr(i = positive)</i>
1:	negative	0.55	11:	positive	0.00
2:	negative	0.05	12:	positive	0.60
3:	negative	0.65	13:	positive	0.10
4:	negative	0.15	14:	positive	0.70
5:	negative	0.75	15:	positive	0.20
6:	negative	0.25	16:	positive	0.80
7:	negative	0.85	17:	positive	0.30
8:	negative	0.35	18:	positive	0.90
9:	negative	0.95	19:	positive	0.40
10:	negative	0.45	20:	positive	1.00

Table 2

```
# Creating the data frame b
df2 <- data.frame(
  TrueObserved2 = c(rep("negative", 10), rep("positive", 10)),
  PredictedPr2 = c(0.55, 0.05, 0.65, 0.15, 0.75, 0.25, 0.85, 0.35,
0.95, 0.45, 0, 0.6, 0.1, 0.7, 0.2, 0.8, 0.3, 0.9, 0.4, 1)
)

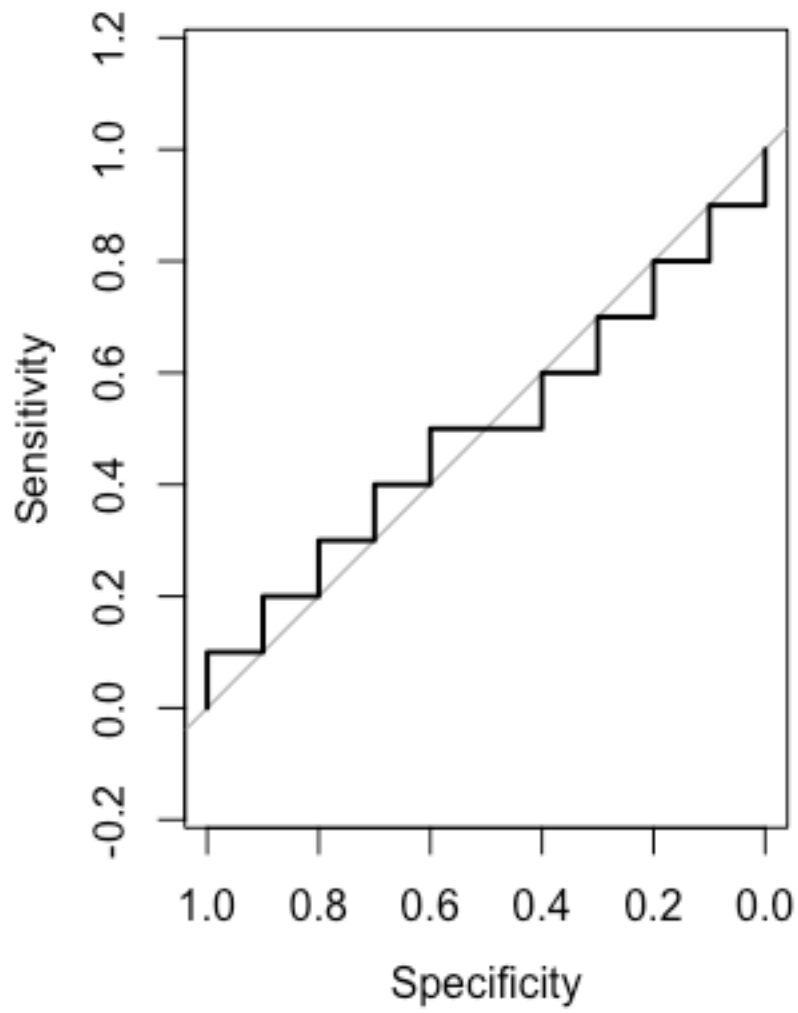
# Calculating sensitivity and specificity for each cutoff value
cutoff_values <- c(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9)
```

```
sens_spec2 <- data.frame(CutOff = cutoff_values, Sensitivity = rep(NA,
length(cutoff_values)), Specificity = rep(NA, length(cutoff_values)))

for (i in 1:length(cutoff_values)) {
  tp2 <- sum(df2$TrueObserved2 == "positive" & df2$PredictedPr2 >=
cutoff_values[i])
  fp2 <- sum(df2$TrueObserved2 == "negative" & df2$PredictedPr2 >=
cutoff_values[i])
  tn2 <- sum(df2$TrueObserved2 == "negative" & df2$PredictedPr2 <
cutoff_values[i])
  fn2 <- sum(df2$TrueObserved2 == "positive" & df2$PredictedPr2 <
cutoff_values[i])

  sens_spec2$Sensitivity[i] <- tp2 / (tp2 + fn2)
  sens_spec2$Specificity[i] <- tn2 / (tn2 + fp2)
}

# Plotting the ROC curve
library(pROC)
roc_curve2 <- roc(df2$TrueObserved2, df2$PredictedPr2)
plot(roc_curve2)
```



[c] Interpret and compare both ROC diagrams with respect to their underlying data in tasks 4 [a] and [b].

In both tasks, [a] and [b], the ROC curves plot the trade-off between sensitivity and specificity for different threshold values of the predicted positive probability. The ROC curve in task [a] has a shape that is typical of a good classifier. It is closer to the top-left corner, indicating that this model can achieve a high true positive rate while keeping a low false positive rate. In contrast, the ROC curve in task [b] is not as good as the one in task [a]. The curve is closer to the diagonal line, which indicates that this model is not as effective at distinguishing between positive and negative cases. Therefore, we can conclude that the model in task [a] performs better than the model in task [b].