Homework 1: Classification

2025-02-06

Question 2.1

Describe a situation or problem from your job, everyday life, current events, etc., for which a classification model would be appropriate. List some (up to 5) predictors that you might use.

An example from my everyday life where a classification model would be useful is determining whether a workout is effective based on personal fitness data. Key predictors for this model could include: (1) heart rate, focusing on the average rate during the session and whether it reached the target zone, (2) calories burned, representing the estimated total energy expenditure during the workout, (3) duration, which captures the total time spent exercising; (4) workout intensity, categorized as light, moderate, or vigorous, and (5) post-workout fatigue, based on self-reported levels of energy or soreness. These factors combined could provide valuable insights into the effectiveness of each workout.

Question 2.2

1) Using the support vector machine function ksvm contained in the R package kernlab, find a good classifier for this data. Show the equation of your classifier, and how well it classifies the data points in the full data set. (Don't worry about test/validation data yet; we'll cover that topic soon.)

```
data1 <- read.table("credit_card_data-headers.txt", header = TRUE, sep = "\t")
library(kernlab)
# call ksvm.  Vanilladot is a simple linear kernel.
model <- ksvm(as.matrix(data1[,1:10]),as.factor(data1[,11]),type="C-svc",kernel="vanilladot",C=1000,sca
```

```
##  Setting default kernel parameters
```

```
# calculate a1...am
a <- colSums(model@xmatrix[[1]] * model@coef[[1]])
a
```

```
##            A1            A2            A3            A8            A9
##  1.280491e-04 -4.049492e-04  2.453581e-06  7.879067e-04  9.977432e-01
##           A10           A11           A12           A14           A15
## -8.338175e-05  6.767138e-04  5.172607e-04  6.494428e-04  1.129666e-03
```

```
# calculate a0
a0 <- model@b
a0
```

```
## [1] -0.07043544
```

```
# see what the model predicts
pred <- predict(model,data1[,1:10])
pred
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1
## [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1
## [260] 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
## [297] 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
## [334] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [371] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [408] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [445] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [482] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [519] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1
## [556] 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
## [593] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
## [630] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## Levels: 0 1
```

```
# see what fraction of the model's predictions match the actual classification
sum(pred == data1[,11]) / nrow(data1)
```

```
## [1] 0.8623853
```

```
for (n in c(0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000)){
  model_new <- ksvm(as.matrix(data1[,1:10]),as.factor(data1[,11]),type='C-svc',kernel='vanilladot',C=n,s
  a <- colSums(model_new@xmatrix[[1]] * model_new@coef[[1]])
  a0 <- model_new@b
  pred <- predict(model_new,data1[,1:10])
  print(paste('The accuracy number is', sum(pred == data1[,11]) / nrow(data1))) }
```

```
##  Setting default kernel parameters
## [1] "The accuracy number is 0.547400611620795"
##  Setting default kernel parameters
## [1] "The accuracy number is 0.837920489296636"
##  Setting default kernel parameters
## [1] "The accuracy number is 0.863914373088685"
##  Setting default kernel parameters
## [1] "The accuracy number is 0.863914373088685"
##  Setting default kernel parameters
## [1] "The accuracy number is 0.863914373088685"
##  Setting default kernel parameters
## [1] "The accuracy number is 0.863914373088685"
##  Setting default kernel parameters
## [1] "The accuracy number is 0.863914373088685"
##  Setting default kernel parameters
## [1] "The accuracy number is 0.862385321100917"
##  Setting default kernel parameters
## [1] "The accuracy number is 0.862385321100917"
```

Model Equation

The resulting classifier equation can be represented as: 1.280491e-04$x1$ *-4.049492e-04*x2 + 2.453581e-06$x3$ + *7.879067e-04*x4 + 9.977432e-01$x5$ *-8.338175e-05*x6 + 6.767138e-04$x7$ + *5.172607e-04*x8 + 6.494428e-04$x9$ + *1.129666e-03*x10 = 0 Here, x1, x2, x3, etc. represent the 10 predictors in the dataset.

Model Performance

Initially, with C=100 the model correctly classified about 86.39% of the data. This means the linear kernel did a good job finding the decision boundary to separate the classes.

Testing Difference Values of C

I tested how changing C, which controls the model's complexity, affected accuracy:

- **Small C (0.0001):** The model performed poorly, with accuracy around 54.74%, because it was too simple and didn't capture enough detail (underfitting).
- **Moderate C (100-1000):** Accuracy improved significantly, reaching 86.39%, which seems to be the sweet spot.
- **Large C (1000000):** Accuracy dropped again as the model became too focused on the training data, losing its ability to generalize (overfitting).

Conclusion

The best accuracy (86.39%) was achieved with C=1000, suggesting this value balances simplicity and accuracy. Increasing C beyond 1000 didn't improve performance, but extremely high values like 1000000 caused accuracy to drop. In summary, the linear SVM with C=1000 works well for this data, correctly classifying 86.39% of the points. With further tuning and testing, the model's performance could potentially be improved.

2) You are welcome, but not required, to try other (nonlinear) kernels as well; we're not covering them in this course, but they can sometimes be useful and might provide better predictions than vanilladot.

```
##  Setting default kernel parameters
```

```
##            A1            A2            A3            A8            A9
##  1.804289e-04  1.909777e-03  1.342818e-04  4.602983e-04  9.985869e-01
##           A10           A11           A12           A14           A15
## -1.555920e-04  1.195944e-03  5.215811e-05  1.774333e-05  7.601946e-04
```

```
## [1] -0.07016765
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1
## [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1
## [260] 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
## [297] 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
## [334] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [371] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [408] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [445] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [482] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [519] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1
## [556] 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
## [593] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
## [630] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## Levels: 0 1
```

```
## [1] 0.8623853
```

```
##  Setting default kernel parameters
## [1] "The accuracy number for polydot is 0.547400611620795"
##  Setting default kernel parameters
## [1] "The accuracy number for polydot is 0.837920489296636"
##  Setting default kernel parameters
## [1] "The accuracy number for polydot is 0.863914373088685"
##  Setting default kernel parameters
## [1] "The accuracy number for polydot is 0.863914373088685"
##  Setting default kernel parameters
## [1] "The accuracy number for polydot is 0.863914373088685"
##  Setting default kernel parameters
## [1] "The accuracy number for polydot is 0.863914373088685"
##  Setting default kernel parameters
## [1] "The accuracy number for polydot is 0.863914373088685"
##  Setting default kernel parameters
## [1] "The accuracy number for polydot is 0.862385321100917"
##  Setting default kernel parameters
## [1] "The accuracy number for polydot is 0.862385321100917"
```

```
##  Setting default kernel parameters
```

```
##          A1          A2          A3          A8          A9         A10
##  -4288.2852 -19042.3446 -16604.3401   3915.4566    593.2796   2266.8149
##         A11         A12         A14         A15
##  -1692.2386  -3912.3250   -964.7174   2822.0637
```

```
## [1] -191.758
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
##  [75] 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 1 1 1 0 0 0 1 1 0 0 1 1
## [112] 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [186] 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 0
## [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1
## [260] 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
## [297] 0 0 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [334] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
## [371] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1
## [408] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [445] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [482] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
## [519] 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
## [556] 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
## [593] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [630] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## Levels: 0 1
```

```
## [1] 0.9204893
```

```
##  Setting default kernel parameters
## [1] "The accuracy number for besseldot is 0.547400611620795"
##  Setting default kernel parameters
## [1] "The accuracy number for besseldot is 0.547400611620795"
##  Setting default kernel parameters
## [1] "The accuracy number for besseldot is 0.678899082568807"
##  Setting default kernel parameters
```

```
## [1] "The accuracy number for besseldot is 0.856269113149847"
##  Setting default kernel parameters
## [1] "The accuracy number for besseldot is 0.868501529051988"
##  Setting default kernel parameters
## [1] "The accuracy number for besseldot is 0.896024464831804"
##  Setting default kernel parameters
## [1] "The accuracy number for besseldot is 0.925076452599388"
##  Setting default kernel parameters
## [1] "The accuracy number for besseldot is 0.920489296636086"
##  Setting default kernel parameters
## [1] "The accuracy number for besseldot is 0.920489296636086"
```

```
##        A1        A2        A3        A8        A9       A10       A11       A12
## -50.59975 -21.97167 -60.08432 121.34947  85.90509 -93.02275  94.87993 -63.80575
##       A14       A15
## -69.81127 102.50618
```

```
## [1] -0.6636819
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
##  [75] 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
## [112] 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1
## [260] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [297] 0 0 0 0 0 0 1 0 1 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [334] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [371] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [408] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [445] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [482] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [519] 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [556] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
## [593] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [630] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## Levels: 0 1
```

```
## [1] 0.9847095
```

```
## [1] "The accuracy number for rbfdot is 0.547400611620795"
## [1] "The accuracy number for rbfdot is 0.547400611620795"
## [1] "The accuracy number for rbfdot is 0.561162079510703"
## [1] "The accuracy number for rbfdot is 0.859327217125382"
## [1] "The accuracy number for rbfdot is 0.871559633027523"
## [1] "The accuracy number for rbfdot is 0.906727828746177"
## [1] "The accuracy number for rbfdot is 0.957186544342508"
## [1] "The accuracy number for rbfdot is 0.981651376146789"
## [1] "The accuracy number for rbfdot is 0.995412844036697"
```

Kernels Tested and Accuracy:

Vanilladot (Linear Kernel):

- Accuracy: 86.39%
- Works well for simple, straight-line relationships in data but may miss complex patterns.

Polydot (Polynomial Kernel):

- Accuracy: 86.23% _ Similar to vanilladot. It can capture non-linear relationships but didn't show much improvement here.

Besseldot (Bessel Kernel):

- Accuracy: 92.04%
- Showed noticeable improvement, meaning it's better at handling more complex patterns.

rbfdot (Radial Basis Function Kernel):

- Accuracy: 98.47%
- The highest accuracy! This kernel seems to be great at handling complex, non-linear data and clearly outperformed the others.

Conclusion:

The RBF kernel is the best option, with an accuracy of almost 98.5%. It's much better than the simpler linear and polynomial kernels, making it the ideal choice for this dataset.

3) Using the k-nearest-neighbors classification function kknn contained in the R kknn package, suggest a good value of k, and show how well it classifies that data points in the full data set. Don't forget to scale the data (scale=TRUE in kknn).

```
library(kknn)

knn_accuracy <- function(k_num) {
  knn_pred <- rep(0, nrow(data1))

  for (i in 1:nrow(data1)) {
      # Train the k-NN model with leave-one-out
    knn_model <- kknn(R1 ~ ., data1[-i,], data1[i, , drop = FALSE], k = k_num, scale = TRUE)
    knn_pred[i] <- as.integer(fitted(knn_model) + 0.5)  # Round prediction to the nearest integer (0 or
  }

    # Correct accuracy calculation
  accuracy <- sum(knn_pred == data1[, 11]) / nrow(data1)
  return(accuracy)
}

  # Store the results in a character vector for better display
knn_acc_vec <- character(30)

for (a in 1:30) {
  accuracy_result <- knn_accuracy(a)
  knn_acc_vec[a] <- paste('k is', a, 'and the accuracy is', round(accuracy_result, 4))
}

  # Print the accuracy results
print(knn_acc_vec)
```

```
##  [1] "k is 1 and the accuracy is 0.815"   "k is 2 and the accuracy is 0.815"
##  [3] "k is 3 and the accuracy is 0.815"   "k is 4 and the accuracy is 0.815"
##  [5] "k is 5 and the accuracy is 0.8517"  "k is 6 and the accuracy is 0.8456"
##  [7] "k is 7 and the accuracy is 0.8471"  "k is 8 and the accuracy is 0.8486"
##  [9] "k is 9 and the accuracy is 0.8471"  "k is 10 and the accuracy is 0.8502"
```

```
## [11] "k is 11 and the accuracy is 0.8517" "k is 12 and the accuracy is 0.8532"
## [13] "k is 13 and the accuracy is 0.8517" "k is 14 and the accuracy is 0.8517"
## [15] "k is 15 and the accuracy is 0.8532" "k is 16 and the accuracy is 0.8517"
## [17] "k is 17 and the accuracy is 0.8517" "k is 18 and the accuracy is 0.8517"
## [19] "k is 19 and the accuracy is 0.8502" "k is 20 and the accuracy is 0.8502"
## [21] "k is 21 and the accuracy is 0.8486" "k is 22 and the accuracy is 0.8471"
## [23] "k is 23 and the accuracy is 0.844"  "k is 24 and the accuracy is 0.8456"
## [25] "k is 25 and the accuracy is 0.8456" "k is 26 and the accuracy is 0.844"
## [27] "k is 27 and the accuracy is 0.841"  "k is 28 and the accuracy is 0.8379"
## [29] "k is 29 and the accuracy is 0.8394" "k is 30 and the accuracy is 0.841"
```

Conclusion:

Before training the model, all the features (the data points' details) are scaled so they all have the same importance. This is necessary for k-NN because it calculates distances between points, and we don't want features with larger values to have more influence than others. The script tries different values of "k" from 1 to 30 to see how the choice of neighbors affects the accuracy. It calculates how many predictions match the actual labels for each "k" value. The accuracy stays steady around 81.5% when using smaller values of "k" (like 1 to 4). This suggests the model might not work as well with very small values of "k". As "k" increases, the accuracy improves and reaches its highest at 85.32% for "k" values of 12 and 15. After "k = 15", the accuracy stays the same or changes very little, indicating that larger "k" values do not improve the model much. The best results come from using "k = 12" or "k = 15", both of which give an accuracy of about 85.32%. These values are considered the most balanced, providing the best prediction accuracy without the model being too sensitive to noise or too simple.