PML opgavesæt 7

2023-05-10

Opgave 1

I denne opgave kigger vi på titanic datasættet. I første omgang vil vi kun kigge på *Alder* og *Billetpris*.

a) og b)

Jeg indlæser datasættet og de nødvendige pakker. Jeg standardiserer input-variablene *Alder* og *Billetpris*. Og splitter i et test-train split med 500 i train og 212 i test.

```
# Indlæs pakker
library(e1071)
library(stats)
library(class)
library(dann)
library(caret)
```

```
## Indlæser krævet pakke: ggplot2
```

```
## Indlæser krævet pakke: lattice
```

```
# Indlæs datasættet
titanic <- read.table("titanic.txt", header = TRUE, sep = "\t")

# Gør overlevelsesvariablen kategorisk og begrænse datasættet
titanic$Overlevelse <- as.factor(titanic$Overlevelse)
titanic <- titanic[, c(1, 4, 7)]

# Standardiser inputvariablerne
titanic$Alder <- scale(titanic$Alder)
titanic$Billetpris <- scale(titanic$Billetpris)

# Inddel datasættet i trænings- og testsæt
set.seed(123) # For at opnå de samme resultater hver gang
n <- nrow(titanic)
sample_size <- 500

train_index <- sample(seq_len(n), size = sample_size)
train_set <- titanic[train_index, ]
test_set <- titanic[-train_index, ]
```

c)

Jeg fitter en support vector classifier med `cost=2`og laver et plot af klassifikatioen på træningsdatasættet.

```
set.seed(2023)

# Fit en Support Vector Classifier model
svm_model <- svm(Overlevelse ~ ., data = train_set, cost = 2, kernel = "linear")




# Brug krydsvalidering til at finde den optimale cost-værdi
tune_result <- tune(svm, Overlevelse ~ ., data = train_set, ranges = list(cost = 2^(-2:7)))
best_cost <- tune_result$best.parameters$cost

# Fit en ny model med den optimale cost-værdi
svm_model_optimal_cost <- svm(Overlevelse ~ ., data = train_set, cost = best_cost, kernel = "linear")

# Plot for den første model (svm_model) med en ændret titel
plot(svm_model, train_set, main = "SVC Model med Cost = 2")

# Plot for den model med optimal cost (svm_model_optimal_cost) og en passende titel
plot(svm_model_optimal_cost, train_set, main = paste("SVC Model med Optimal Cost =", best_cost))
```

```r
# Prædikere testdata værdierne for den optimale model
predicted_values <- predict(svm_model_optimal_cost, newdata = test_set)

# Sammenlign de prædikerede værdier med de faktiske værdier for at finde fejlene
errors <- sum(predicted_values != test_set$Overlevelse)

# Beregn fejlraten
error_rate <- errors / length(predicted_values)

# Print fejlraten
cat("Fejlraten for den optimale model er:", round(error_rate * 100, 2), "%")
```

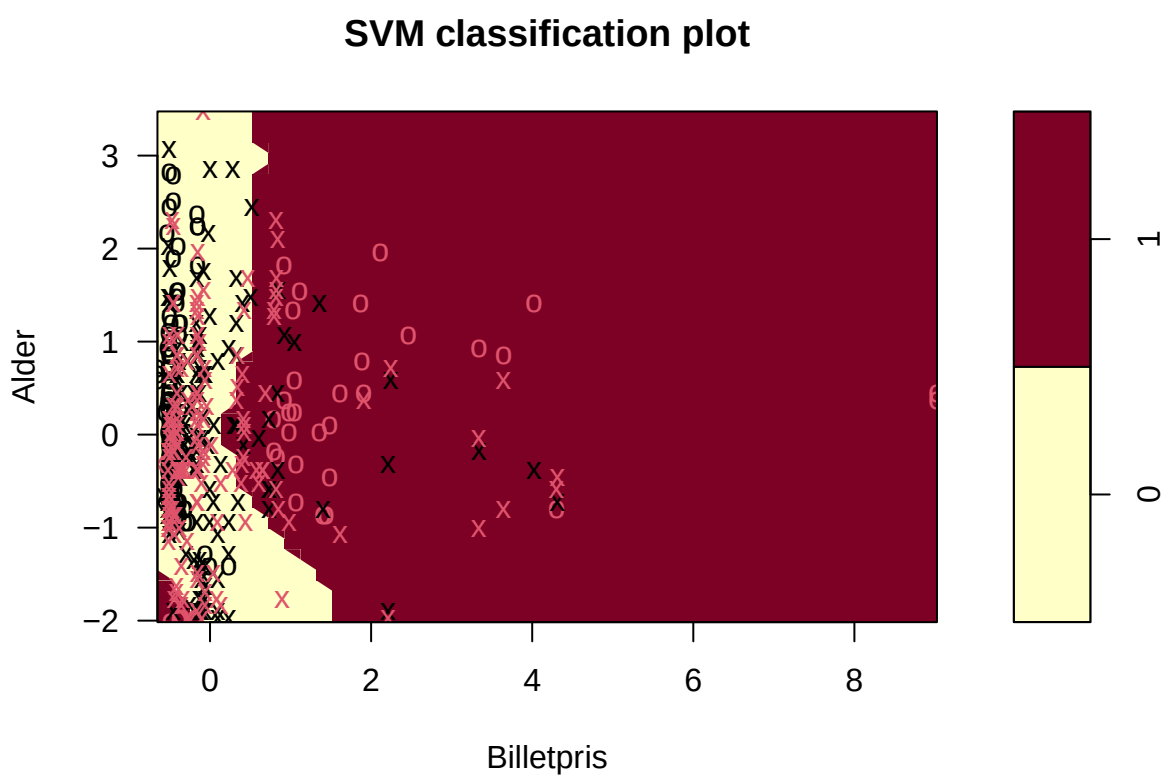
```
## Fejlraten for den optimale model er: 40.57 %
```

Det virker til at høje billetpriser har større chance for at overleve.

d)

Jeg gør det samme som i opgave c), men fitter en SVM med radial kernefunktion. Jeg benytter CV for at finde de optimale værdier for `cost` og `lambda`.

```r
# Brug krydsvalidering til at finde de optimale værdier for cost og gamma (inverse af lambda)
tune_result_radial <- tune(svm, Overlevelse ~ ., data = train_set, kernel = "radial", ranges = list(cos
best_cost_radial <- tune_result_radial$best.parameters$cost
best_gamma <- tune_result_radial$best.parameters$gamma

# Fit en ny model med de optimale cost og gamma værdier
svm_model_optimal_radial <- svm(Overlevelse ~ ., data = train_set, cost = best_cost_radial, gamma = bes

# Plot for den model med optimal cost og gamma (svm_model_optimal_radial) og en passende titel
plot(svm_model_optimal_radial, train_set, main = paste("Radial Kernel - Optimal Cost =", best_cost_radi
```




```
# Prædikere testdata værdierne for den optimale model
predicted_values_radial <- predict(svm_model_optimal_radial, newdata = test_set)

# Sammenlign de prædikerede værdier med de faktiske værdier for at finde fejlene
errors_radial <- sum(predicted_values_radial != test_set$Overlevelse)

# Beregn fejlraten
error_rate_radial <- errors_radial / length(predicted_values_radial)

# Print fejlraten
cat("Fejlraten for den optimale model med radial kernel er:", round(error_rate_radial * 100, 2), "%")
```

```
## Fejlraten for den optimale model med radial kernel er: 35.85 %
```

Det ses nu at der kommer en form for grupering ved børn, som viser en større chance for at overleve.

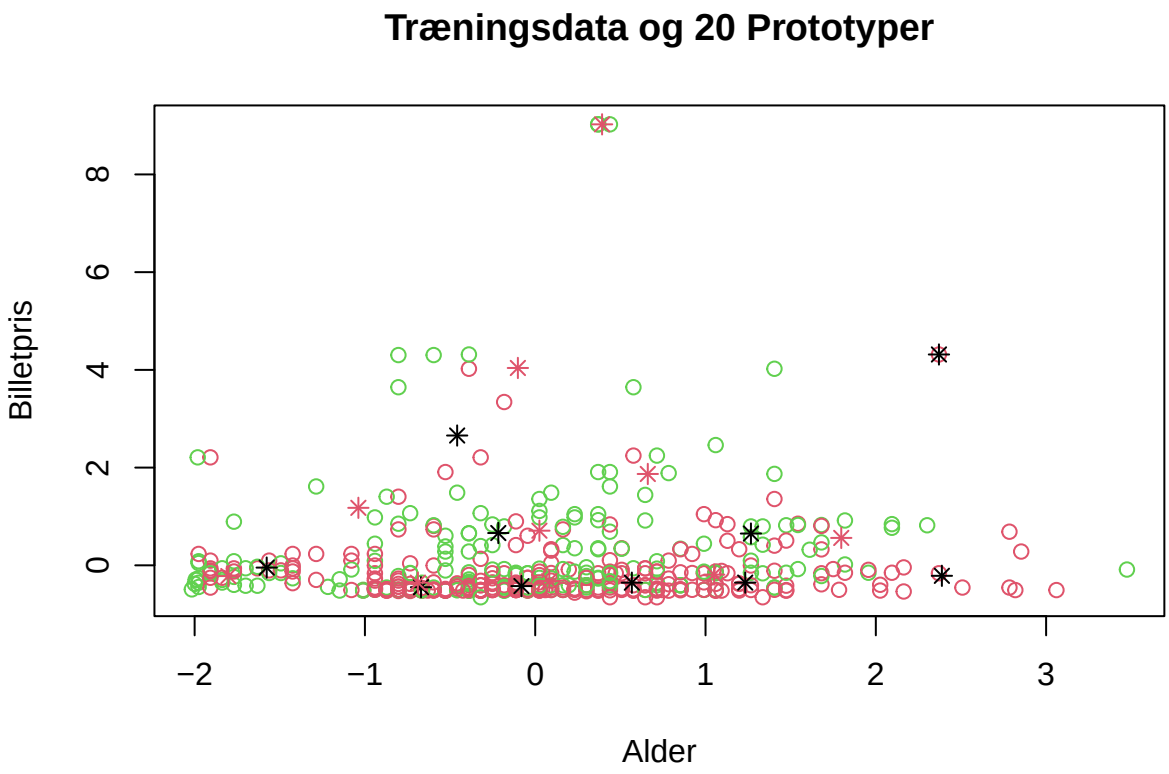
e)

```
# Antal observationer i datasættet
n <- nrow(titanic)

# Størrelse af træningssættet
train_size <- 500
```

```r
# Opret en logisk vektor for at indeksere træningsdatasættet
train_index <- sample(1:n, train_size)

# Definer trænings- og testdata baseret på tidligere opdeling
Xtrain <- titanic[train_index, c(2,3)]
Xtest <- titanic[-train_index, c(2,3)]
Gtrain <- titanic[train_index,1]
Gtest <- titanic[-train_index,1]

# Antal clusters i hver klasse
M <- 10
K <- 2
p <- ncol(Xtrain)

# Kør k-means clustering og beregn prototyper
prototypes <- matrix(0, nrow = K*M, ncol = p+1)

for (k in 1:K){
  kmeans_res <- kmeans(Xtrain[Gtrain == (k-1), ], M, iter.max = 10, nstart = 25)
  prototypes[((k-1)*M+1):(k*M),1:p] <- kmeans_res$centers
  prototypes[((k-1)*M+1):(k*M),p+1] <- (k-1)
}

# Prædiktere testdata værdierne med prototyper
predicted_prototypes <- knn(prototypes[, 1:p], Xtest, prototypes[, p+1], k = 1)

# Beregn fejlraten
errors_prototypes <- sum(predicted_prototypes != Gtest)
error_rate_prototypes <- errors_prototypes / length(predicted_prototypes)

# Print fejlraten
cat("Fejlraten for M-means clustering med 10 prototyper i hver klasse er:", round(error_rate_prototypes
```

```
## Fejlraten for M-means clustering med 10 prototyper i hver klasse er: 39.15 %
```

```r
# Plot træningsdatasættet og de 20 prototyper
plot(Xtrain, col = as.integer(Gtrain) + 1, xlab = "Alder", ylab = "Billetpris", main = "Træningsdata og
points(prototypes[, 1:p], col = as.integer(prototypes[, p+1]) + 1, pch = 8)
```



f)

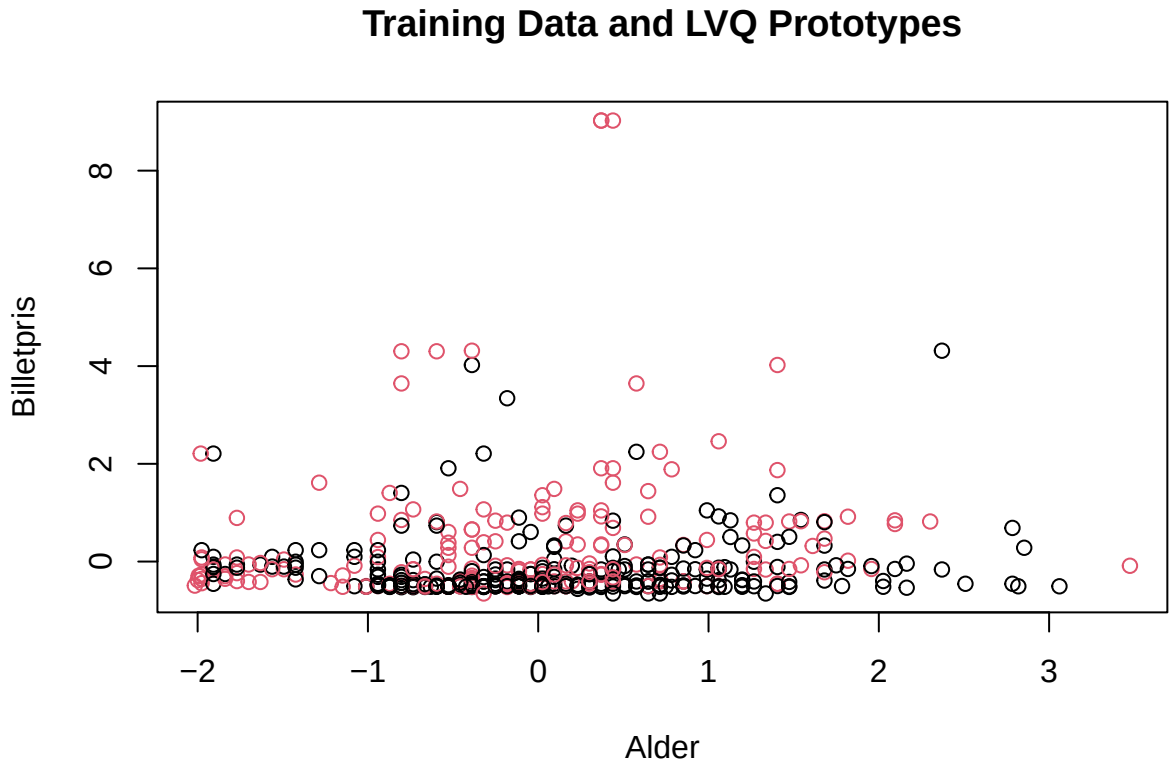
Jeg bruger LVQ-metoden igen med 20 prototyper (10 i hver klasse).

```
# Define the number of prototypes per class
M <- 10

# Use LVQ to get the prototypes
start_prototypes <- lvqinit(Xtrain, Gtrain, size=M)
lvq_prototypes <- lvq1(Xtrain, Gtrain, start_prototypes) #adjust this line as per the actual lvq1 funct

# Predict the test data values with LVQ prototypes
lvq_predictions <- lvqtest(lvq_prototypes, Xtest) #adjust this line as per the actual lvqtest function

# Calculate the error rate
lvq_errors <- sum(lvq_predictions != Gtest)
lvq_error_rate <- lvq_errors / length(Gtest)

# Print the error rate
print(paste("The error rate for LVQ with", M, "prototypes per class is:", round(lvq_error_rate * 100, 2
```

```
## [1] "The error rate for LVQ with 10 prototypes per class is: 35.38 %"
```

```r
# Convert the factor variables to numeric
Gtrain_numeric <- as.numeric(as.character(Gtrain))
Gtest_numeric <- as.numeric(as.character(Gtest))

# Plot the training data and the prototypes
plot(Xtrain, col=Gtrain_numeric+1, main="Training Data and LVQ Prototypes")
points(lvq_prototypes$x, col=lvq_prototypes$cl+1, pch=8) #adjust this line as per the actual structure
```

```
## Warning in Ops.factor(lvq_prototypes$cl, 1): '+' er ikke meningsfuld for
## faktorer
```



g)

Jeg gør nu det samme for m-nearest-neighbors og *dann* metoden. Jeg benytter CV for m-NN:

```r
k_values <- 1:20

# Initialize a vector to store the error rates for each k
error_rates <- numeric(length(k_values))

# Loop over k values
for (k in k_values) {
  # Use knn function to predict the test data values
  knn_predictions <- knn(train = Xtrain, test = Xtest, cl = Gtrain, k = k)
```

```
  # Calculate the error rate
  knn_errors <- sum(knn_predictions != Gtest)
  knn_error_rate <- knn_errors / length(Gtest)

  # Store the error rate
  error_rates[k] <- knn_error_rate
}

# Get the optimal k value
optimal_k <- which.min(error_rates)

# Print the optimal k and corresponding error rate
print(paste("The optimal k is", optimal_k, "with an error rate of", round(error_rates[optimal_k] * 100,
```

```
## [1] "The optimal k is 19 with an error rate of 28.77 %"
```

```
# Ensure the classes are numeric
Gtrain <- as.numeric(Gtrain)

dann.pred<-dann(Xtrain, Gtrain, Xtest, k = 15, neighborhood_size = 50, epsilon=1)
dann_error_rate <- mean(dann.pred!=Gtest)

# Print the error rate
print(paste("The error rate for Dann is:", round(dann_error_rate * 100, 2), "%"))
```

```
## [1] "The error rate for Dann is: 75 %"
```

DANN performer bedst.

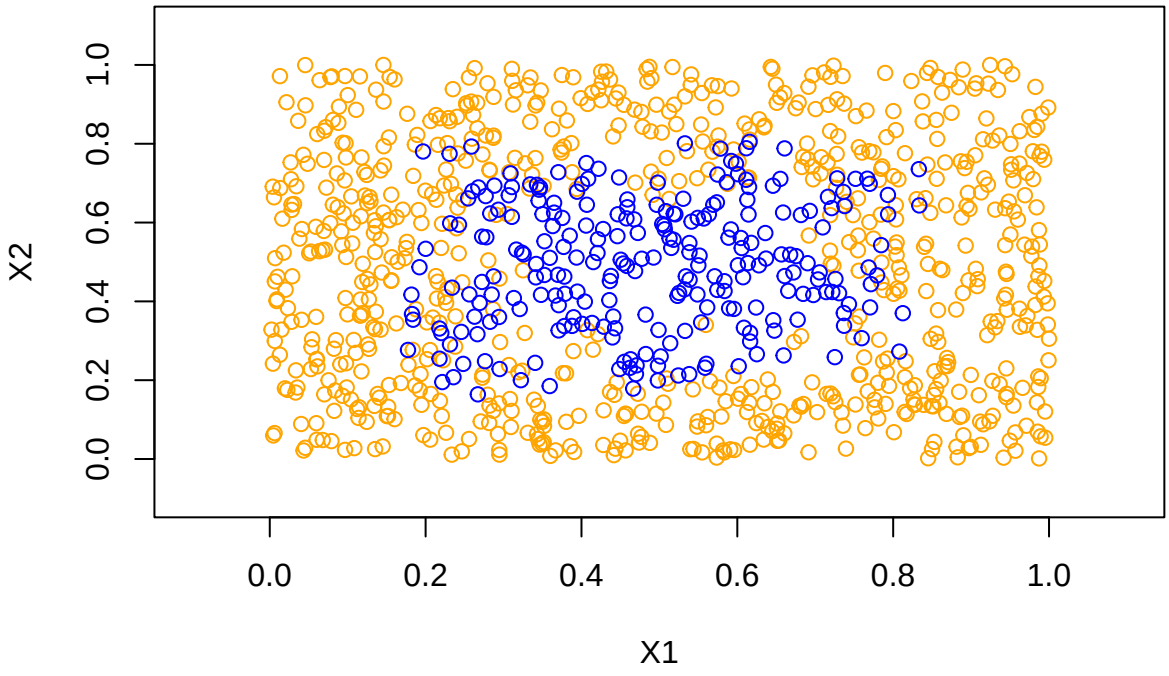
Opgave 2

Der henvises til opgave 2 i opgavesættet for en forklaring.

Afsnit 1

```
##########################################
############# Afsnit 1 ###################
##########################################
library(class)

N<-1000
m<-10

X1<-runif(N)
X2<-runif(N)
U1<-runif(N,-0.1,0.1)
U2<-runif(N,-0.1,0.1)
```

```
G<-((X1+U1)>0.25)*((X1+U1)<0.75)*((X2+U2)>0.25)*((X2+U2)<0.75)

X<-cbind(X1,X2)

plot(X[G==0,],col='orange',xlab='X1',ylab='X2',xlim=c(-0.1,1.1),ylim=c(-0.1,1.1))
points(X[G==1,],col='blue')
```




```
Xtest<-matrix(c(0.7,0.7),ncol=2)

pred<-knn(X, Xtest, G, k = m,prob=TRUE)

pred
```

```
## [1] 0
## attr(,"prob")
## [1] 0.6
## Levels: 0 1
```

```
ssh<-(pred==1)*attributes(pred)$prob+(pred==0)*(1-attributes(pred)$prob)
ssh
```

```
## [1] 0.4
```

```
##########################################
############# Afsnit 2 ###################
##########################################

U1<-runif(100000,-0.1,0.1)
U2<-runif(100000,-0.1,0.1)

Gtruedist<-((0.7+U1)>0.25)*((0.7+U1)<0.75)*((0.7+U2)>0.25)*((0.7+U2)<0.75)
mean(Gtruedist)
```

```
## [1] 0.56159
```

```
##########################################
############# Afsnit 3 ###################
##########################################

m<-10
N<-1000
NN<-10000
ssh<-rep(NA,NN)

for (i in 1:NN){

  X1<-runif(N)
  X2<-runif(N)
  U1<-runif(N,-0.1,0.1)
  U2<-runif(N,-0.1,0.1)

  G<-((X1+U1)>0.25)*((X1+U1)<0.75)*((X2+U2)>0.25)*((X2+U2)<0.75)

  X<-cbind(X1,X2)

  #plot(X[G==0,],col='orange',xlab='X1',ylab='X2',xlim=c(-0.1,1.1),ylim=c(-0.1,1.1))
  #points(X[G==1,],col='blue')

  Xtest<-matrix(c(0.7,0.7),ncol=2)

  pred<-knn(X, Xtest, G, k = m,prob=TRUE)
  ssh[i]<-(pred==1)*attributes(pred)$prob+(pred==0)*(1-attributes(pred)$prob)
}


mean(ssh)
```

```
## [1] 0.5612345
```

```
var(ssh)
```

```
## [1] 0.02531167
```

a)

Afsnit 1:

Dette afsnit genererer et datasæt med 1000 observationer (N = 1000). Hver observation har to træk (X1 og X2), som er tilfældige værdier mellem 0 og 1. En tilsvarende værdi 'G' genereres for hver observation baseret på betingelserne på X1 og X2 (det vil sige, hvis begge ligger mellem 0.25 og 0.75, er G = 1, ellers er G = 0). Det genererede datasæt visualiseres derefter med en plot-funktion, hvor orange punkter repræsenterer observationer med G = 0 og blå punkter repræsenterer observationer med G = 1.

Derefter bruges k-Nearest Neighbors (kNN) klassificeringsalgoritmen (med k = 10) til at prædiktere værdien af G for en testobservation med X1 = X2 = 0.7. Koden udskriver også den estimerede sandsynlighed for, at G = 1 for denne testobservation.

Afsnit 2:

Dette afsnit genererer 100000 tilfældige G-værdier ved hjælp af samme metode som i afsnit 1, men denne gang for en fast X-værdi (X1 = X2 = 0.7). Den gennemsnitlige værdi af disse G-værdier beregnes derefter, som giver en estimering af den sande betingede sandsynlighed P(G = 1 | X = (0.7, 0.7)).

Afsnit 3:

Dette afsnit gentager processen i afsnit 1 i en løkke 10000 gange. I hver iteration genereres et nyt datasæt, og kNN-algoritmen bruges til at prædiktere G-værdien for X1 = X2 = 0.7. De estimerede sandsynligheder for G = 1 akkumuleres i en vektor (ssh), og derefter beregnes gennemsnitsværdien og variansen for disse sandsynligheder.

b)

Resultatet fra Afsnit 3, som er middelværdien og variansen af de estimerede sandsynligheder, kan bruges til at evaluere præcisionen og stabiliteten af k-Nearest Neighbors (kNN) algoritmen.

Middelværdien af de estimerede sandsynligheder (fra Afsnit 3) skulle gerne være tæt på den sande betingede sandsynlighed (fra Afsnit 2), hvis kNN-algoritmen er nøjagtig. Afvigelser mellem disse to værdier angiver bias i kNN-estimaterne.

Variansen af de estimerede sandsynligheder (fra Afsnit 3) angiver, hvor meget disse estimater varierer fra et datasæt til et andet. En høj varians indikerer, at kNN-estimaterne er følsomme over for ændringer i datasættet, hvilket kan resultere i overfitting. En lav varians indikerer, at kNN-estimaterne er stabile over forskellige datasæt.

Det ser ud til at afsnit i gennemsnit rammer tæt på ssh. i afsnit 2 fra ca. 0.562 til 0.56175. Mens variansen er givet ved 0.024. Ved støre *m=20* estimeres ssh. i gennemsnit til 0.548 dog reduceres variansen til 0.0125 (højere bias lavere varians).

c)

Den værdi af m vi vælger til m-nearest-neighbors algoritmen har en direkte indflydelse på bias og varians for modellen.

Når m er stor (fx 20), vil modellen have lav varians og høj bias. Denne model vil være mindre følsom over for udsving i træningssættet, men kan være for simplistisk, hvilket kan føre til undervurdering. Med andre ord, vil den ikke fange kompleksiteten i data særlig godt, hvilket fører til en højere bias. Men da den bruger flere punkter (20 nærmeste naboer) til at træffe sin beslutning, vil dens beslutninger være mere stabile og dermed have en lavere varians.

Når m er lav (fx 1 eller 2), vil modellen have høj varians og lav bias. I denne situation vil modellen potentielt kunne fange en høj grad af kompleksitet i data, hvilket fører til en lavere bias. Men da den bruger meget få punkter (1 eller 2 nærmeste naboer) til at træffe sin beslutning, kan dens beslutninger variere meget, hvis vi ændrer træningssættet en lille smule, hvilket betyder, at den har høj varians.

d)

`N` i denne kode repræsenterer antallet af observationer i træningsdatasættet. Ændringer i N kan påvirke resultatet på følgende måder:

Når `N` er stor:

Hvis vi har en stor mængde træningsdata, vil vores model generelt blive bedre til at generalisere til nye data, fordi den har haft mere information at lære af. Dog kan det tage længere tid at træne modellen, da der er flere data at behandle. Desuden kan det, afhængig af dataens art, potentielt introducere mere støj i dine forudsigelser, især hvis meget af data er outliers eller støj.

Når N er lille:

Hvis vi har en lille mængde træningsdata, vil modellen måske ikke være i stand til at lære de underliggende mønstre i dataene meget godt, hvilket kan føre til overfitting (hvor modellen lærer træningsdataene for godt og har dårlig ydeevne på nye data) eller underfitting (hvor modellen er for simpel til at fange de underliggende mønstre i dataene). Men det vil tage mindre tid at træne modellen, da der er færre data at behandle.