Tenta Oct 2018

Oskar Hidén - oskhi827

10/24/2020

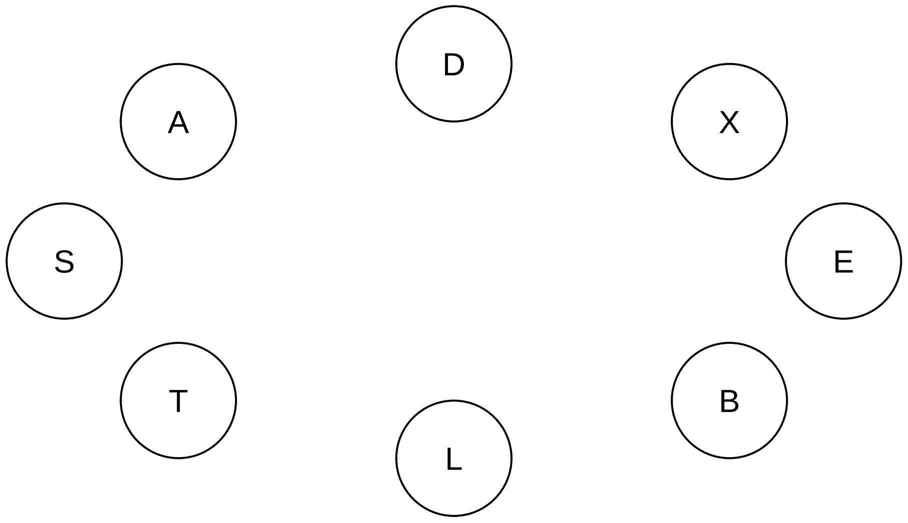
{ setup, include=FALSE} knitr::opts_chunk$set(echo = TRUE, message = FALSE, warning = FALSE, out.width='.49\\linewidth', fig.width=5, fig.height=5,fig.show='hold',fig.align='center', results='asis')

Graphical Models

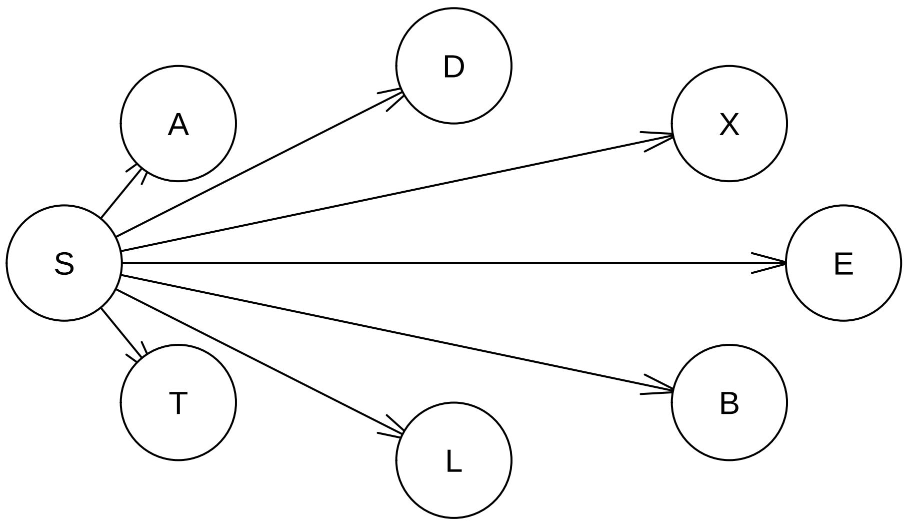
```
library(bnlearn)
library(gRain)
```

```
## Loading required package: gRbase

##
## Attaching package: 'gRbase'

## The following objects are masked from 'package:bnlearn':
##
##     ancestors, children, parents
```

```
data("asia")

set.seed(567)
data("asia")
ind <- sample(1:5000, 4000)
tr <- asia[ind,]
te <- asia[-ind,]

#Crating an empty network
b_net = empty.graph(names(asia))
plot(b_net)
```



```r
# Adjacncy matrix (0L ensures that the number is stored as an integer instead of a double)
adj = matrix(0L, ncol = 8, nrow = 8,
             dimnames = list(names(asia), names(asia)))

# Add edges in BN
for (i in names(asia)) {
  adj["S",i] = 1L
}
adj["S","S"] = 0L
amat(b_net) = adj
plot(b_net)
```



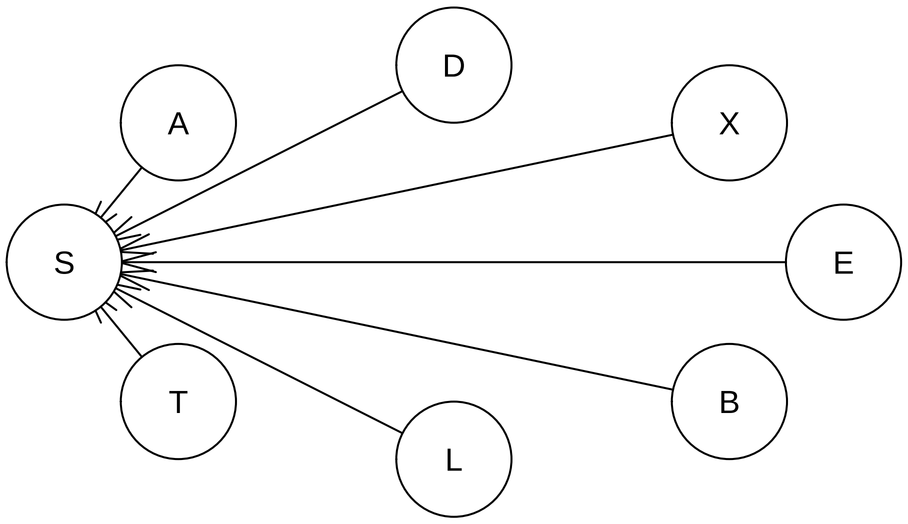
```r
get_error = function(b_net, tr, te, nr_data){
  # learn parameters(potentials) conditioning on the bn-structure.
  bn_pot = bn.fit(b_net, data=tr[1:nr_data,], method = "bayes")
  bn_grain = as.grain(bn_pot)

  # Calculate prob given evidense (observations)
  test_evid = te[,-2]
  test_ans = te[,2]
  nodes_ev = names(test_evid)

  pred_s =c()
  for (j in 1:dim(test_evid[1])) {
    # finding/evidance or potentials
    obs = c()
    for (i in 1:dim(test_evid)[2]) {
      obs = c(obs, as.character(test_evid[j,i]))
    }

    evid = setEvidence(bn_grain, nodes_ev, states = obs)

    # quergrain to get conditional distributon
    node = c("S")
    prob_s = querygrain(evid, nodes = node)

    if (prob_s$S[1]>prob_s$S[2]) {
      pred_s=c(pred_s,"no")
    }else{
      pred_s=c(pred_s,"yes")
    }
  }

  table = table(pred_s,test_ans)
  return (1 - sum(diag(table))/sum(table))
  }

  totalError<-array(dim=6)
  k<-1
  for (i in c(10, 20, 50, 100, 1000, 2000)) {
    totalError[k] = get_error(b_net, tr, te, nr_data = i)
    k = k+1
  }
```

```
## Warning in check.data(data, allow.missing = TRUE): variable A has levels that
## are not observed in the data.

## Warning in check.data(data, allow.missing = TRUE): variable T has levels that
## are not observed in the data.

## Warning in 1:dim(test_evid[1]): numerical expression has 2 elements: only the
## first used

## Warning in check.data(data, allow.missing = TRUE): variable A has levels that
## are not observed in the data.

## Warning in check.data(data, allow.missing = TRUE): variable T has levels that
## are not observed in the data.

## Warning in 1:dim(test_evid[1]): numerical expression has 2 elements: only the
## first used

## Warning in check.data(data, allow.missing = TRUE): variable T has levels that
## are not observed in the data.

## Warning in 1:dim(test_evid[1]): numerical expression has 2 elements: only the
## first used

## Warning in check.data(data, allow.missing = TRUE): variable T has levels that
## are not observed in the data.

## Warning in 1:dim(test_evid[1]): numerical expression has 2 elements: only the
## first used

## Warning in 1:dim(test_evid[1]): numerical expression has 2 elements: only the
## first used

## Warning in 1:dim(test_evid[1]): numerical expression has 2 elements: only the
## first used
```

```
  totalError
```

```
## [1] 0.328 0.328 0.335 0.335 0.335 0.335
```

```
# Reverse edges:
# Adjacncy matrix (0L ensures that the number is stored as an integer instead of a double)
adj = matrix(0L, ncol = 8, nrow = 8,
             dimnames = list(names(asia), names(asia)))

# Add edges in BN
for (i in names(asia)) {
  adj[i,"S"] = 1L
}
adj["S","S"] = 0L
amat(b_net) = adj
plot(b_net)
```



```
  totalError_rev<-array(dim=6)
  k<-1
  for (i in c(10, 20, 50, 100, 1000, 2000)) {
    totalError_rev[k] = get_error(b_net, tr, te, nr_data = i)
    k = k+1
  }
```

```
## Warning in check.data(data, allow.missing = TRUE): variable A has levels that
## are not observed in the data.

## Warning in check.data(data, allow.missing = TRUE): variable T has levels that
## are not observed in the data.

## Warning in 1:dim(test_evid[1]): numerical expression has 2 elements: only the
## first used

## Warning in check.data(data, allow.missing = TRUE): variable A has levels that
## are not observed in the data.

## Warning in check.data(data, allow.missing = TRUE): variable T has levels that
## are not observed in the data.

## Warning in 1:dim(test_evid[1]): numerical expression has 2 elements: only the
## first used

## Warning in check.data(data, allow.missing = TRUE): variable T has levels that
## are not observed in the data.

## Warning in 1:dim(test_evid[1]): numerical expression has 2 elements: only the
## first used

## Warning in check.data(data, allow.missing = TRUE): variable T has levels that
## are not observed in the data.

## Warning in 1:dim(test_evid[1]): numerical expression has 2 elements: only the
## first used

## Warning in 1:dim(test_evid[1]): numerical expression has 2 elements: only the
## first used

## Warning in 1:dim(test_evid[1]): numerical expression has 2 elements: only the
## first used
```

```
  totalError
```

```
## [1] 0.328 0.328 0.335 0.335 0.335 0.335
```

```
  totalError_rev
```

```
## [1] 0.488 0.488 0.307 0.320 0.302 0.305
```

When using Naive Bayes, where we have an arrow from “S” to all other nodes, we have created a bunch of forks. That means that we assume all other varibles to be independent, when conditioning on “S”. Which is used when calclating hte potentials. This makes the calculation quite eazy.

When all arrows point torwards “S”, there could be a dependece between all other nodes when conditioning on “S”. Therfore this makes computation harder. Because we also need to find these other dependencies we need more training data. But wit enough data, since it do take those other dependencies into acount, we get a lower misclassification rate. We can also see in this experiment that the naive approach performs better on low ammounts of data, while the later model works better for larger ammounts of traning data.

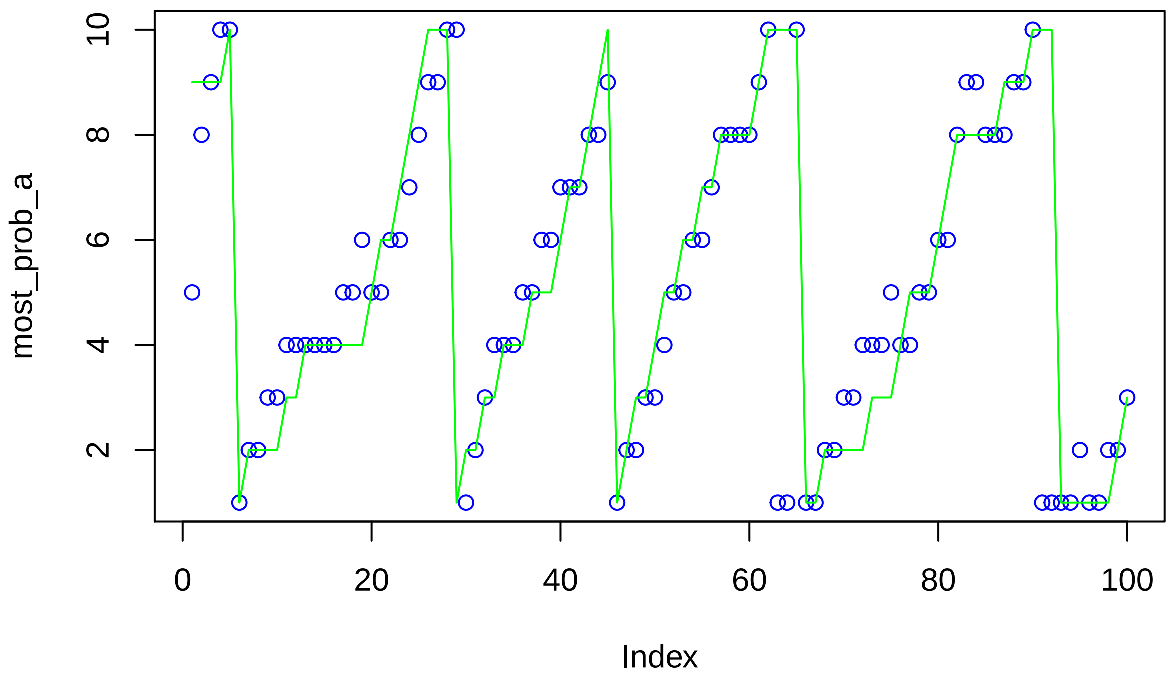
In other words,note that p(C|A_1,. . . ,A_n) is proportional to P(A_1,. . . ,A_n|C) p(C) by Bayes theorem. NB assumesthat P(A_1,. . . ,A_n|C) factorizes into a product of factors p(A_i|C) whereas the alternative model assumesnothing. The NB's assumption may hurt performance. This can be observed in the experiments.4

(Discussion from answers) The NB classifier only needs to estimate the parameters for distributions of the form p(C) and P(A_i|C) where C is the class variable and A_i is a predictive attribute. The alternative model needs to estimate p(C) and P(C|A_1,. . . ,A_n). Therefore, it requires more data to obtain reliable estimates (e.g. many of the parental combinations may not appear in the training data and this is actually why you should use method="bayes", to avoid zero relative frequency counters).This may hurt performance when little learning data is available. This is actually observed in the experiments above. However, when the size of the learning data increases, the alternative model should outperform NB, because the latter assumes that the attributes are independent given the class whereas the former does not. In other words, note that p(C|A_1,. . . ,A_n) is proportional to P(A_1,. . . ,A_n|C) p(C) by Bayes theorem. NB assumes that P(A_1,. . . ,A_n|C) factorizes into a product of factors p(A_i|C) whereas the alternative model assumes nothing. The NB's assumption may hurt performance. This can be observed in the experiments.

2 - Hidden Markow Models

```r
# Setting upp the HMM
library(HMM)

states = c("1", "2", "3", "4", "5", "6", "7", "8", "9", "10")
symbols = c("1", "2", "3", "4", "5", "6", "7", "8", "9", "10")
#start_prob = rep(0, 10)
#start_prob[1] = 1
start_prob = rep(0.1, 10)
sur_state = function(x){
  state = x%%10
  if (state ==0) {
    state=10
  }
  return(state)
}
trans_prob = matrix(data=0, nrow = 10, ncol=10)
for (i in 1:10) {
  trans_prob[i,i] = 0.5
  trans_prob[i,sur_state(i+1)] = 0.5
}
emmis_prob = matrix(data=0, nrow = 10, ncol=10)
for (i in 1:10) {
  for (j in -2:2) {
    emmis_prob[i,sur_state(i+j)] = 0.2
  }
}
HMM = initHMM(states, symbols, start_prob, trans_prob, emmis_prob)

# sim data from HMM
set.seed(12345)
N = 100
```

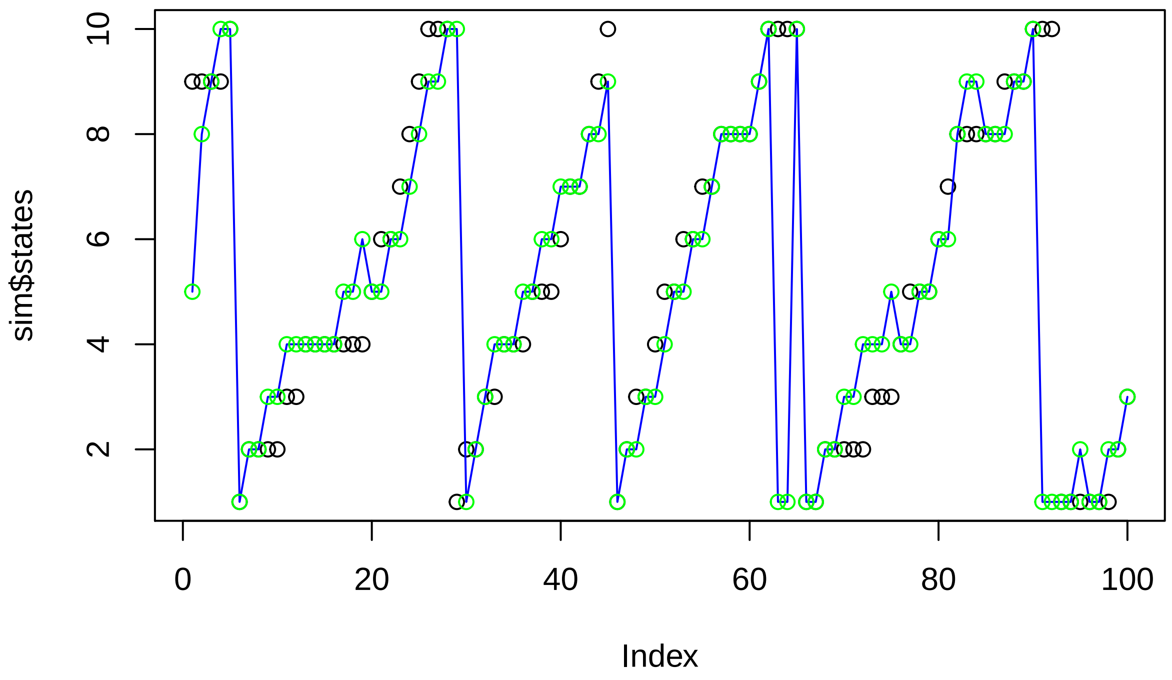
```
sim = simHMM(HMM, N)
observed = sim$observation

# filtered distributon from HMM package. Filterd alpha --Alpha uses all observations up to point t to e
alpha_log = forward(HMM, observed)
alpha = exp(alpha_log)
most_prob_a = apply(alpha, MARGIN = 2, which.max)
plot(most_prob_a, col="blue")
lines(sim$states, col="green")
```



```
# Implementing filtered distribution
a = matrix(NA, nrow = N, ncol = length(states))

# a(z_0)
for (i in 1:length(states)) {
  a[1,i] = emmis_prob[i,as.integer(observed[1])]*start_prob[i]
}

# recursivly calculate a(z_t)
for (t in 2:N) {
  for (i in 1:length(states)) {
    a[t,i] = emmis_prob[i,as.integer(observed[t])] * sum(a[t-1,]*trans_prob[,i])
    # For each state z in time t, calculate the "prob"
             # ^ Prob to observe symbol x given z
                                                       # ^ sum upp "prob" to transission to z, given all z
  }
}

# Calculate filter (nomalize)
for (t in 1:N) {
  a[t,] = a[t,]/sum(a[t, ])
}
# now it sums to 1:
sum(a[3,])
```

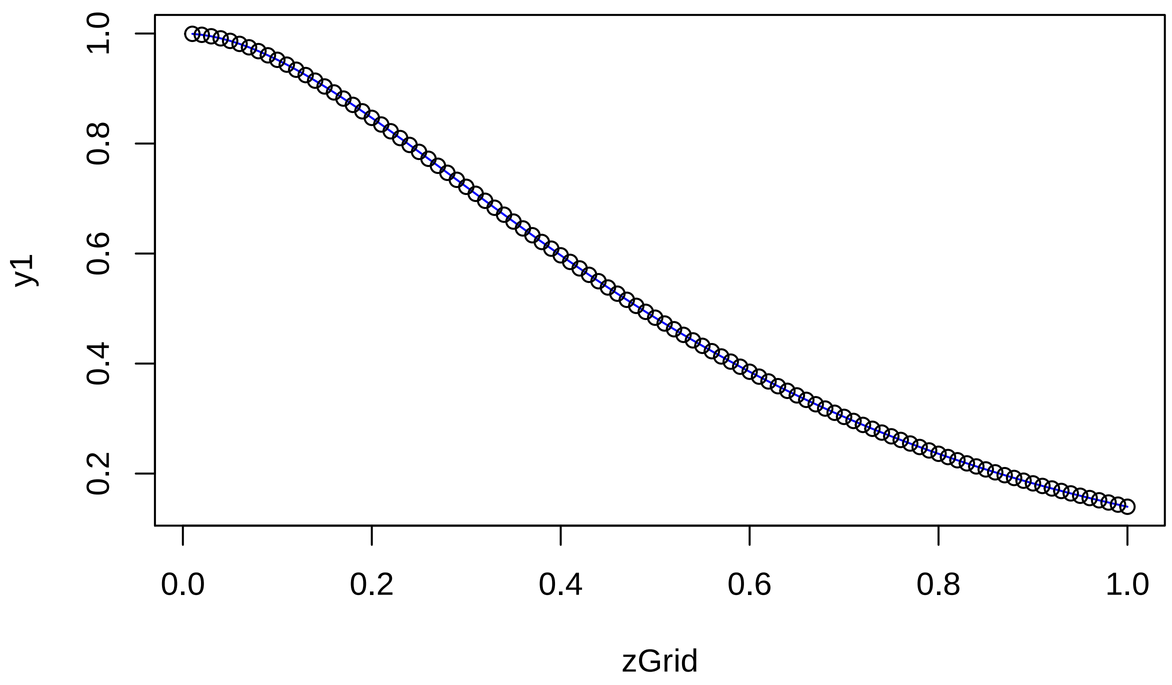
```
## [1] 1
```

```
# make prediction from max prob.
filter_pred = apply(a, MARGIN = 1, FUN=which.max) # random when equal prob: which.max(rank(a[t,], ties.

plot(sim$states)
lines(most_prob_a, col="blue")
lines(filter_pred, col="green", type = "p")
```



4 - Gaussian Processes

```r
library(kernlab)
sigma2_f = 1
ell = 0.5

set_mattern_kernel = function(sigma2_f, ell){
  mat_kern = function(x, x_star){
    r = sqrt(sum((x - x_star)^2)) # Euclidian distance
    p1 = sigma2_f*(1+sqrt(3)*r/ell)
    p2 = exp(-sqrt(3)*r/ell)
    return(p1*p2)
  }
  class(mat_kern) <- "kernel"
  return(mat_kern)
}

Matern32 <- function(sigmaf = 1, ell = 1)
{
  rval <- function(x, y = NULL) {
      r = sqrt(crossprod(x-y));
      return(sigmaf^2*(1+sqrt(3)*r/ell)*exp(-sqrt(3)*r/ell))
    }
  class(rval) <- "kernel"
  return(rval)
}

mat1 = set_mattern_kernel(sigma2_f, ell)
mat2 = Matern32(sigma2_f, ell)

zGrid = seq(0.01,1,by=0.01)
x = 0

y1 = kernelMatrix(mat1, x, zGrid)
y2 = kernelMatrix(mat2, x, zGrid)

plot(zGrid, y1, col="blue", type = "l")
points(zGrid, y2)
```

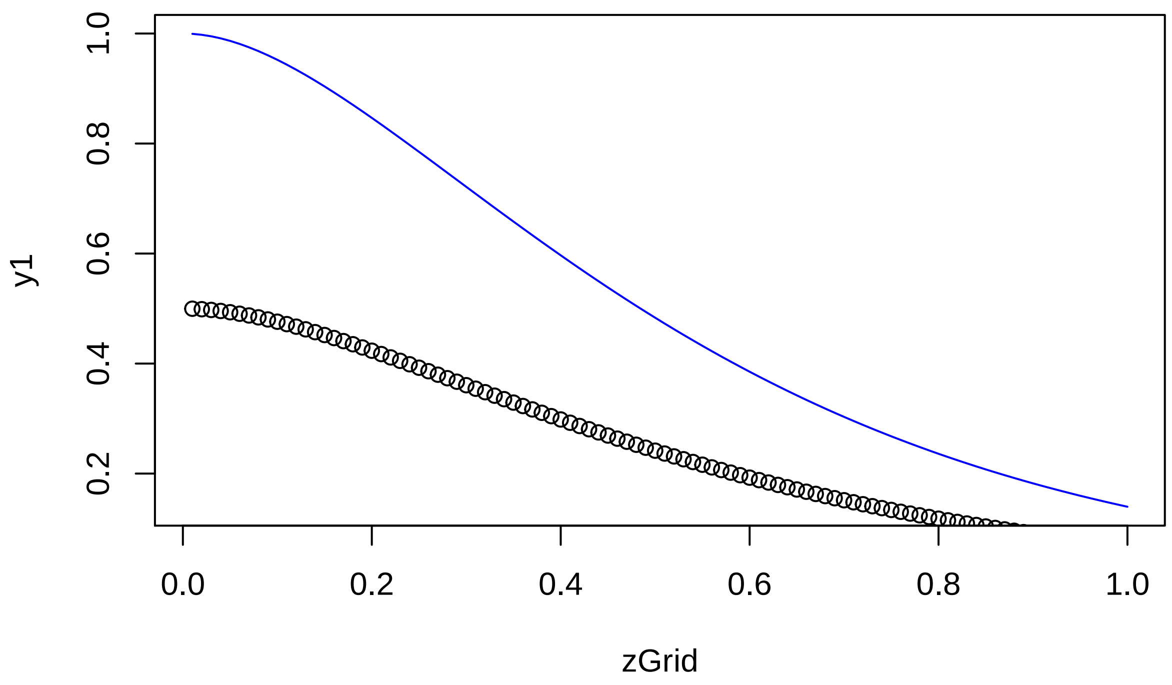


Interpret the plot. Connect your discussion to the smoothness of f.

The plot shows how the correlation(y-axis) between two point is reduced by the increase in distance(x-axis) between them. The smoothnes of f is grater if the correlation is keept larger for points that are futhrer from eachoter, the graph would in that case have a larger derivitive and decrease slower thorwards 0.

Answer The graph plots Cov(f(0),f(z)), the correlation between two FUNCTION VALUES, as a function of the distance between two inputs (0 and z). As expected the correlation between two points on f decreases as the distance increases. The fact that points of f are dependent, as given by the covariance in the plot, makes the curves smooth. Nearby inputs will have nearby outputs when the correlation is large.

```
mat3 = set_mattern_kernel(sigma2_f = 0.5, ell = 0.5 )

y3 = kernelMatrix(mat3, x, zGrid)

plot(zGrid, y1, col="blue", type = "l")
points(zGrid, y3)
```



.. discuss the effect this change has on the distribution of f.

σ_f^2 is the variance for x = x (where the graph crosses the y-axix), and decreasing σ_f^2 decreases the overall size of correlation. Allowing the overall variation form funciton mean to be lower. Changing σ_f^2 will not effective the relative covariance between the points on the curve (smoothness).

Answer Changing sigma2f will have not effect on the relative covariance between points on the curve, i.e. will not affect the smoothness. But lowering sigma2f makes the whole covariance curve lower. This means that the variance k(0,0) is lower and has the effect of giving a probability distribution over curves which is tighter (lower variance) around the mean function of the GP. This means that simulated curves from the GP will be less variable.

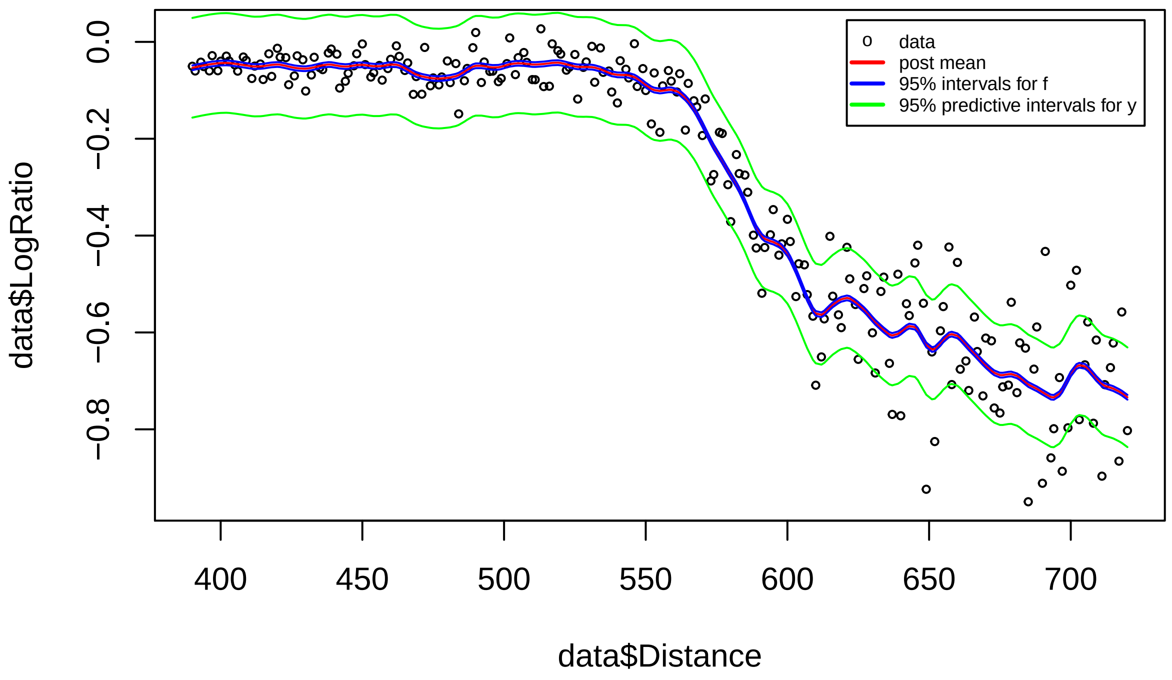
```
library(kernlab)
# load("/Users/oskarhiden/Git/TDDD15/Exams/lidar.rData") # loading the data
data = read.table("/Users/oskarhiden/Git/TDDD15/Exams/LidarData.txt", header = TRUE, dec = ".")
data$LogRatio
```

```
##   [1] -0.050355730 -0.060097060 -0.041900910 -0.050984700 -0.059913450
##   [6] -0.028423920 -0.059584210 -0.039888810 -0.029395820 -0.039494450
##  [11] -0.047647490 -0.060380000 -0.031230340 -0.038165840 -0.075622690
##  [16] -0.050017510 -0.045729500 -0.077669660 -0.024606410 -0.071331840
##  [21] -0.013207460 -0.031626150 -0.032474780 -0.088407970 -0.070241660
##  [26] -0.028772630 -0.036967020 -0.101562500 -0.068313730 -0.031757510
##  [31] -0.053368190 -0.057263810 -0.022955150 -0.014791660 -0.025318840
##  [36] -0.095389440 -0.081261080 -0.064738010 -0.049400040 -0.024539830
##  [41] -0.004223316 -0.046929080 -0.072642600 -0.063754970 -0.048675710
##  [46] -0.079021940 -0.055083920 -0.036009150 -0.008198360 -0.029916380
##  [51] -0.059044170 -0.043490760 -0.108366700 -0.071644950 -0.108043100
##  [56] -0.011447560 -0.090664970 -0.074388490 -0.088807160 -0.072401300
##  [61] -0.039412080 -0.084136980 -0.044777780 -0.148665100 -0.080272660
##  [66] -0.054824790 -0.012024890  0.019348600 -0.083894270 -0.041574770
##  [71] -0.061091210 -0.060443250 -0.082201860 -0.075303490 -0.044809910
##  [76]  0.008222156 -0.067588090 -0.032499460 -0.021981460 -0.042326210
##  [81] -0.077852130 -0.078061950  0.026907170 -0.092260960 -0.091590450
##  [86] -0.004001756 -0.018189330 -0.025276170 -0.058424990 -0.052573050
##  [91] -0.026062480 -0.118087700 -0.052631100 -0.041351480 -0.009199134
##  [96] -0.083366440 -0.012533340 -0.062903320 -0.060180180 -0.103568600
## [101] -0.126116600 -0.038870120 -0.056549840 -0.074448420 -0.003788664
## [106] -0.092039710 -0.055173560 -0.100433700 -0.169438000 -0.064141840
## [111] -0.186734800 -0.090860060 -0.059187140 -0.081035100 -0.103477600
## [116] -0.065673940 -0.182124500 -0.085712700 -0.121604100 -0.134269200
## [121] -0.193390800 -0.117863900 -0.287109000 -0.273944700 -0.186719400
## [126] -0.189195300 -0.294862500 -0.371198400 -0.232675300 -0.271973600
## [131] -0.275020400 -0.310527100 -0.399157200 -0.425672200 -0.519027200
## [136] -0.424584400 -0.398271100 -0.346376700 -0.440465100 -0.416947400
## [141] -0.366367800 -0.412190200 -0.525884900 -0.458107900 -0.460408100
## [146] -0.521879200 -0.566426900 -0.708995600 -0.650540200 -0.571982900
## [151] -0.401473600 -0.525136200 -0.563555800 -0.590039200 -0.424141300
## [156] -0.489361900 -0.542304700 -0.655504900 -0.509341000 -0.483083200
## [161] -0.600755200 -0.683696500 -0.515332200 -0.485786300 -0.663813500
## [166] -0.769263200 -0.479689100 -0.772027200 -0.540661400 -0.565100900
## [171] -0.456521700 -0.419902300 -0.539716900 -0.923823200 -0.640224000
## [176] -0.825267700 -0.596618300 -0.546587100 -0.423674600 -0.707649500
## [181] -0.455401300 -0.675938200 -0.658868800 -0.719863500 -0.568205500
## [186] -0.639420400 -0.731023700 -0.611648100 -0.617259600 -0.755963700
## [191] -0.766384800 -0.712309300 -0.708644300 -0.537711200 -0.724290000
## [196] -0.621456400 -0.632412100 -0.949553500 -0.675763100 -0.588709700
## [201] -0.911435300 -0.432679400 -0.859001700 -0.798765300 -0.693147200
## [206] -0.886574000 -0.796826100 -0.502526800 -0.471670200 -0.780108800
## [211] -0.666843100 -0.578347900 -0.787452200 -0.615695600 -0.896760200
## [216] -0.707737900 -0.672567000 -0.621841300 -0.865761100 -0.557754000
## [221] -0.802668400
```

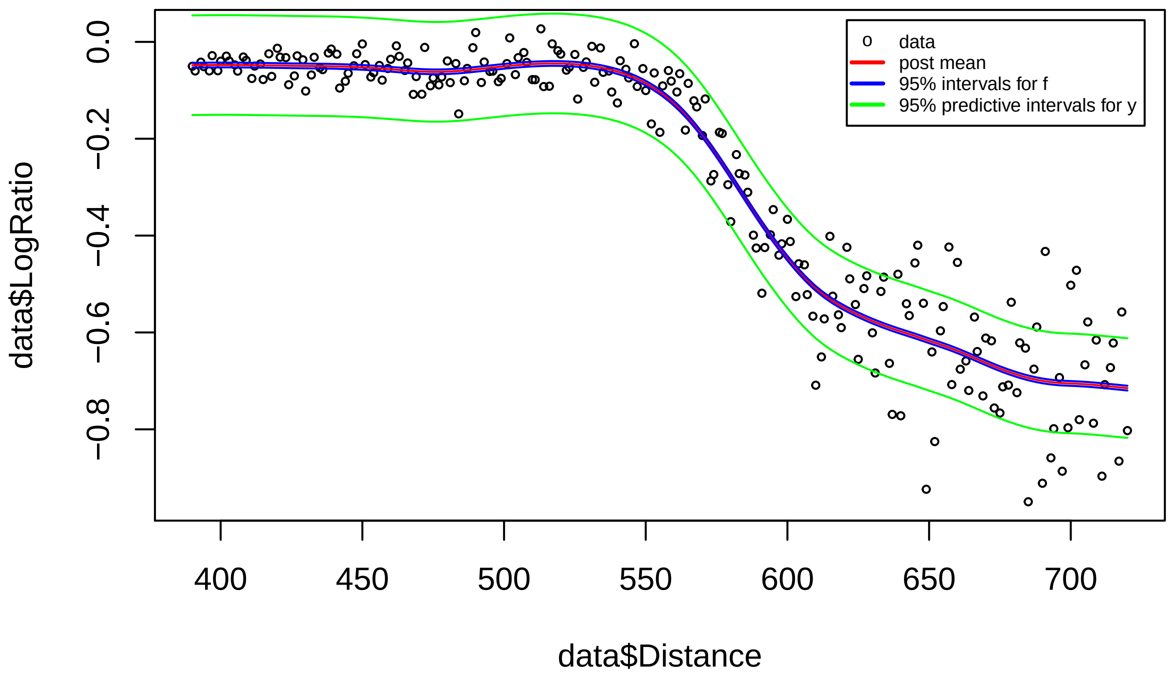
```
data$Distance
```

```
##   [1] 390 391 393 394 396 397 399 400 402 403 405 406 408 409 411 412 414 415
##  [19] 417 418 420 421 423 424 426 427 429 430 432 433 435 436 438 439 441 442
##  [37] 444 445 447 448 450 451 453 454 456 457 459 460 462 463 465 466 468 469
##  [55] 471 472 474 475 477 478 480 481 483 484 486 487 489 490 492 493 495 496
##  [73] 498 499 501 502 504 505 507 508 510 511 513 514 516 517 519 520 522 523
##  [91] 525 526 528 529 531 532 534 535 537 538 540 541 543 544 546 547 549 550
## [109] 552 553 555 556 558 559 561 562 564 565 567 568 570 571 573 574 576 577
## [127] 579 580 582 583 585 586 588 589 591 592 594 595 597 598 600 601 603 604
## [145] 606 607 609 610 612 613 615 616 618 619 621 622 624 625 627 628 630 631
## [163] 633 634 636 637 639 640 642 643 645 646 648 649 651 652 654 655 657 658
## [181] 660 661 663 664 666 667 669 670 672 673 675 676 678 679 681 682 684 685
## [199] 687 688 690 691 693 694 696 697 699 700 702 703 705 706 708 709 711 712
## [217] 714 715 717 718 720
```

```r
# Algorithm 2.1
posteriorGP = function(X_input, y_targets, k_cov_function, sigmaNoise=1, XStar){
  A = k_cov_function(X_input, X_input)
  A = A + diag(length(X_input))*sigmaNoise^2
  L = t(chol(A)) # chol Returns t(L)
  L_y = solve(L, y_targets)
  alpha = solve(t(L), L_y)

  k_star = k_cov_function(X_input, XStar)
  f_star = t(k_star)%*%alpha

  v = solve(L,k_star)
  V_f_star = k_cov_function(XStar, XStar) - t(v)%*%v
  n = length(y_targets)
  log_mar = -0.5 * (t(y_targets)%*%alpha) - sum(diag(L))-(n/2)*log(2*pi)
  return(list("mean"=f_star, "cov" = V_f_star, "log" = log_mar))
}
# END alg

k = function(x, x_star){
  return(kernelMatrix(kernel, x, x_star))
}

sigma_f = 1
ell = 1
sigma_n = 0.05

kernel = set_mattern_kernel(sigma2_f = sigma_f^2, ell = ell)

gp_model = gausspr( x = data$Distance, y = data$LogRatio, type = "regression", kernel=kernel, var = sig

mean_pred = predict(gp_model, data$Distance)

# find conf intervall usign alg 2.1
postGP = posteriorGP(X_input = data$Distance, y_targets = data$LogRatio, k_cov_function = k, sigmaNoise
sd_conf = sqrt(diag(postGP$cov))

# find pred intervall
sd_pred = sd_conf+sigma_n

plot(data$Distance, data$LogRatio, main = "", cex = 0.5)
lines(data$Distance, mean_pred, col = "red")
lines(data$Distance, mean_pred+1.96*sd_conf, col = "blue")
lines(data$Distance, mean_pred-1.96*sd_conf, col = "blue")
lines(data$Distance, mean_pred+1.96*sd_pred, col = "green")
lines(data$Distance, mean_pred-1.96*sd_pred, col = "green")
legend("topright", inset = 0.02, legend = c("data","post mean","95% intervals for f", "95% predictive i
       col = c("black", "red", "blue", "green"),
       pch = c('o',NA,NA,NA), lty = c(NA,1,1,1), lwd = 2, cex = 0.55)
```



```
# with a larger ell
ell=5

kernel = set_mattern_kernel(sigma2_f = sigma_f^2, ell = ell)

gp_model = gausspr( x = data$Distance, y = data$LogRatio, type = "regression", kernel=kernel, var = sig

mean_pred = predict(gp_model, data$Distance)

# find conf intervall usign alg 2.1
postGP = posteriorGP(X_input = data$Distance, y_targets = data$LogRatio, k_cov_function = k, sigmaNoise
sd_conf = sqrt(diag(postGP$cov))

# find pred intervall
sd_pred = sd_conf+sigma_n

plot(data$Distance, data$LogRatio, main = "", cex = 0.5)
lines(data$Distance, mean_pred, col = "red")
lines(data$Distance, mean_pred+1.96*sd_conf, col = "blue")
lines(data$Distance, mean_pred-1.96*sd_conf, col = "blue")
lines(data$Distance, mean_pred+1.96*sd_pred, col = "green")
lines(data$Distance, mean_pred-1.96*sd_pred, col = "green")
legend("topright", inset = 0.02, legend = c("data","post mean","95% intervals for f", "95% predictive i
       col = c("black", "red", "blue", "green"),
       pch = c('o',NA,NA,NA), lty = c(NA,1,1,1), lwd = 2, cex = 0.55)
```



Discuss the differences in results from using the two length scales. The two lenght scales creates a prior regarding the smoothnes of the function. A higher ell forces the posterior to be more smooth, as we can se in the two plots.

Answer The larger length scale gives smoother fits. The smaller length scale seems to generate too jagged fits.