Advanced Machine Learning Lab 1 (Graphical Models)

Omkar Bhutra (omkbh878)

21 September 2019

Task 1:

```r
data("asia")
asia <- as.data.frame(asia)
#default parameter run of HC
hillClimbingResults = hc(asia)
print(hillClimbingResults)
```

```
##
##   Bayesian network learned via Score-based methods
##
##   model:
##    [A][S][T][L|S][B|S][E|T:L][X|E][D|B:E]
##   nodes:                                 8
##   arcs:                                  7
##     undirected arcs:                     0
##     directed arcs:                       7
##   average markov blanket size:           2.25
##   average neighbourhood size:            1.75
##   average branching factor:              0.88
##
##   learning algorithm:                    Hill-Climbing
##   score:                                 BIC (disc.)
##   penalization coefficient:              4.258597
##   tests used in the learning procedure:  77
##   optimized:                             TRUE
```

```r
par(mfrow = c(2,3))

hillclimb <- function(x){
hc1 <- hc(x,restart = 15,score = "bde", iss = 3)
hc2 <- hc(x,restart = 10,score = "bde", iss = 5)
hc1dag <- cpdag(hc1)
plot(hc1dag, main = "plot of BN1")
hc1arcs <- vstructs(hc1dag, arcs = TRUE)
# print(hc1arcs)
arcs(hc1)
hc2dag <- cpdag(hc2)
plot(hc2dag, main = "plot of BN2")
hc2arcs <- vstructs(hc2dag, arcs = TRUE)
# print(hc2arcs)
arcs(hc2)
print(all.equal(hc1dag,hc2dag))
}
```
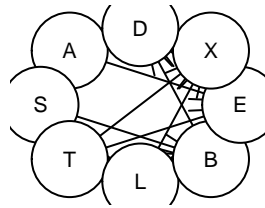
```
for(i in 1:3){
  hillclimb(asia)
}
```

[1] "Different number of directed/undirected arcs"

[1] "Different number of directed/undirected arcs"

plot of BN1
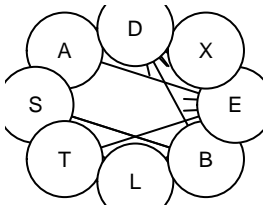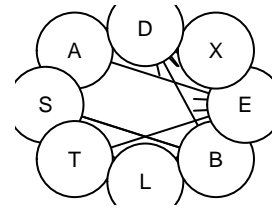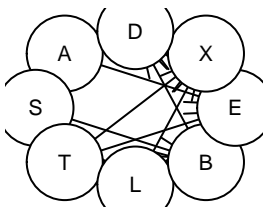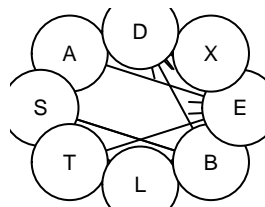
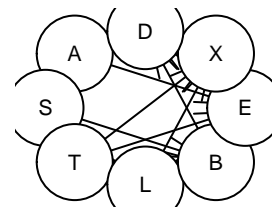

plot of BN2

plot of BN1

plot of BN2

plot of BN1

plot of BN2

[1] "Different number of directed/undirected arcs"

Non-equivalant solutions are produced using different starting parameters such as number of restarts, scoring algorithm, imaginary sample size etc. Hill climbing algorithm is a deterministic one, and hence generates different results on different input parameters. Hill climbing leads to a local maxima of the objective function which makes two different results possible with the same code. Adding imaginary sample size affects the relationships between the nodes such that we get more edges. Number of restarts increases the possibility of getting the same result.

Task 2:

Network structure is trained by the hill climbing algorithm using the BDE score. bn.fit is used to learn the paramets using maximum liklihood estimation. To predict S in the test data , evidence in the network is set to the values of the test data excluding S, using setEvidence. querygrain is used to find the probabilty, maximum of the values is taken to find the misclassification rate.

```r
#setting up training and testing datasets
id <- sample(x = seq(1, dim(asia)[1], 1),
                        size = dim(asia)[1]*0.8,
                        replace = FALSE)
asia.train <- asia[id,]
asia.test <- asia[-id,]

bnprediction = as.numeric()
#fitting a model using Hill Climbing
bnmodel <- hc(asia.train,score = "bde",restart = 50)
bnmodelfit <- bn.fit(bnmodel,asia.train,method = 'mle') #max liklihood estimation
bngrain <- as.grain(bnmodelfit)
comp <- compile(bngrain)

bnmodel_true <- model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
bnmodelfit_true <- bn.fit(bnmodel_true, asia.train)
bngrain_true <- as.grain(bnmodelfit_true)
comp_true <- compile(bngrain_true)

bnpredict <- function(bntree, data, obs_variables, pred_variable) {
  for (i in 1:dim(data)[1]) {
    x <- NULL
    for (j in obs_variables) {
      x[j] <- if(data[i, j] == "yes") "yes" else "no"
    }
    evidence <- setEvidence(object = bntree,nodes = obs_variables,states=x)
    prob_dist_fit  <- querygrain(object = evidence,nodes = pred_variable)$S
    bnprediction[i] <- if (prob_dist_fit["yes"] >= 0.5) "yes" else "no"
  }
  return(bnprediction)
}

# Predict S from Bayesian Network and test data observations
bnprediction <- bnpredict(bntree = comp,
                                    data = asia.test,
                                    obs_variables = c("A", "T", "L", "B", "E", "X", "D"),
                                    pred_variable = c("S"))
bnprediction_true <- bnpredict(bntree = comp_true,
                                    data = asia.test,
                                    obs_variables = c("A", "T", "L", "B", "E", "X", "D"),
                                    pred_variable = c("S"))

# Calculate confusion matricies
confusion_matrix_fit <- table(bnprediction, asia.test$S)
print(confusion_matrix_fit)
```

```
##
## bnprediction  no yes
##          no  329 136
##          yes 145 390
```

```r
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_fit))/sum(confusion_matrix_fit)))
```

```
## [1] "Misclassification rate: 0.281"
```

```r
confusion_matrix_true <- table(bnprediction_true, asia.test$S)
print(confusion_matrix_true)
```
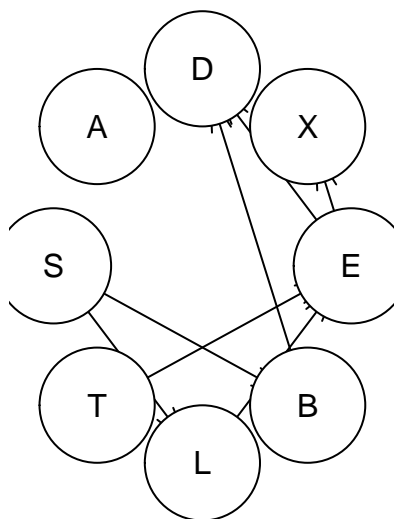
```
##
## bnprediction_true  no yes
##               no  329 136
##               yes 145 390
```

```r
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_true))/sum(confusion_matrix_true)))
```
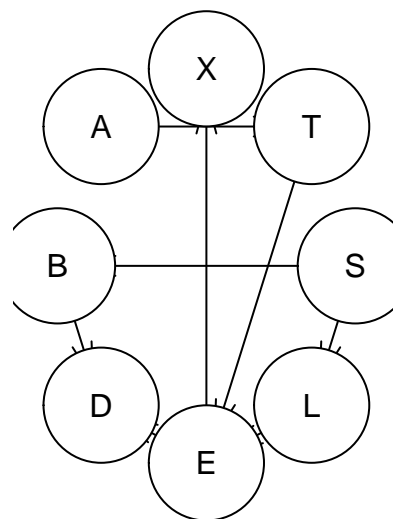
```
## [1] "Misclassification rate: 0.281"
```

```r
par(mfrow=c(1,2))
plot(bnmodel)
title("hill climb network")
plot(bnmodel_true)
title("True network")
```



```r
par(mfrow=c(1,1))
```

The Misclassification rate remain the same for both the hill climb network and the true network. However,
The state of "Different number of directed/undirected arcs" persists.

Task 3:

Using the markov blanket, we predict the results again, it is expected to output the same result.

```
markov_blanket = mb(bnmodel, c("S"))
markov_blanket_true = mb(bnmodel_true, c("S"))
bnpredict_mb <- bnpredict(comp, asia.test, markov_blanket, c("S"))
bnpredict_mb_true <- bnpredict(comp_true, asia.test, markov_blanket_true, c("S"))
confusion_matrix_fit <- table(bnpredict_mb, asia.test$S)
confusion_matrix_fit_true <- table(bnpredict_mb_true, asia.test$S)
print(confusion_matrix_fit)
```

```
##
## bnpredict_mb  no yes
##          no  329 136
##          yes 145 390
```

```
print(confusion_matrix_fit_true)
```

```
##
## bnpredict_mb_true  no yes
##               no  329 136
##               yes 145 390
```

```
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_fit))/sum(confusion_matrix_fit)))
```

```
## [1] "Misclassification rate: 0.281"
```

```
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_fit_true))/sum(confusion_matrix_fit_
```
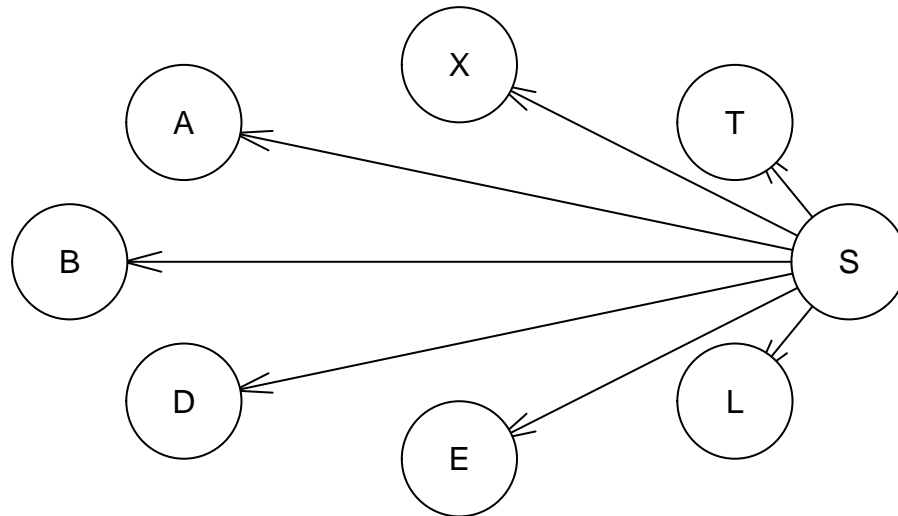
```
## [1] "Misclassification rate: 0.281"
```

Task 4:

The naiveBayes structure implies that all variables are independant of S, this implies that there is a diffrent true structure used.

```
naiveBayes = model2network("[S][A|S][B|S][D|S][E|S][X|S][L|S][T|S]")
plot(naiveBayes, main = "Naive Bayes")
```

Naive Bayes


```
naiveBayes.fit <- bn.fit(naiveBayes, asia.train)
result_naive <- bnpredict(compile(as.grain(naiveBayes.fit)), asia.test, c("A", "T", "L", "B", "E", "X",
# Calculate confusion matricies
confusion_matrix_naive_bayes <- table(result_naive, asia.test$S)
print(confusion_matrix_naive_bayes)
```

```
##
## result_naive  no yes
##          no  349 208
##          yes 125 318
```

```
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_naive_bayes))/sum(confusion_matrix_na
```

```
## [1] "Misclassification rate: 0.333"
```

Task 5:

In Task 2, Same results are observed for the trained structure and the true structure. Same results are also obtained in Task 3 as before, this is expected due to the reduction of the node from the rest of the structure. The reduced number of predictors still provides the same misclassification rate although some information loss is expected. In Task 4, Naive bayes is not a predictor for the data as the structure used is different as the variables are inferred to be independant of S, but all are assumed to have an effect on S.

```r
knitr::opts_chunk$set(echo = TRUE)
library("dplyr")
library("ggplot2")
library("gRain")
library("bnlearn")
data("asia")
asia <- as.data.frame(asia)
#default parameter run of HC
hillClimbingResults = hc(asia)
print(hillClimbingResults)
par(mfrow = c(2,3))

hillclimb <- function(x){
hc1 <- hc(x,restart = 15,score = "bde", iss = 3)
hc2 <- hc(x,restart = 10,score = "bde", iss = 5)
hc1dag <- cpdag(hc1)
plot(hc1dag, main = "plot of BN1")
hc1arcs <- vstructs(hc1dag, arcs = TRUE)
# print(hc1arcs)
arcs(hc1)
hc2dag <- cpdag(hc2)
plot(hc2dag, main = "plot of BN2")
hc2arcs <- vstructs(hc2dag, arcs = TRUE)
# print(hc2arcs)
arcs(hc2)
print(all.equal(hc1dag,hc2dag))
}
for(i in 1:3){
  hillclimb(asia)
}
# Question 1: PGMs

# Learn a BN from the Asia dataset, find a separation statement (e.g. B _|_ E | S, T) and, then, check
# it corresponds to a statistical independence.
library(bnlearn)
library(gRain)
set.seed(123)
data("asia")
hc3<-hc(asia,restart=10,score="bde",iss=10)
plot(hc3)
hc4<-bn.fit(hc3,asia,method="bayes")
hc5<-as.grain(hc4)
hc6<-compile(hc5)
hc7<-setFinding(hc6,nodes=c("S","T","E"),states=c("yes","yes","yes"))
querygrain(hc7,c("B"))
hc7<-setFinding(hc6,nodes=c("S","T","E"),states=c("yes","yes","no"))
querygrain(hc7,c("B"))
hc7<-setFinding(hc6,nodes=c("S","T","E"),states=c("yes","no","yes"))
querygrain(hc7,c("B"))
hc7<-setFinding(hc6,nodes=c("S","T","E"),states=c("yes","no","no"))
querygrain(hc7,c("B"))
hc7<-setFinding(hc6,nodes=c("S","T","E"),states=c("no","yes","yes"))
querygrain(hc7,c("B"))
```

```r
hc7<-setFinding(hc6,nodes=c("S","T","E"),states=c("no","yes","no"))
querygrain(hc7,c("B"))
hc7<-setFinding(hc6,nodes=c("S","T","E"),states=c("no","no","yes"))
querygrain(hc7,c("B"))
hc7<-setFinding(hc6,nodes=c("S","T","E"),states=c("no","no","no"))
querygrain(hc7,c("B"))

# Sample DAGs at random and, then, check which of them coincide with their CPDAGs, i.e. the CPDAGs have
# undirected edge (recall that a CPDAG has an undirected edge if and only if there are two DAGs in the
# equivalence class that differ in the direction of that edge).

#  The exact ratio according to the literature is 11.2
library(bnlearn)
set.seed(123)
ss<-50000
x<-random.graph(c("A","B","C","D","E"),num=ss,method="melancon",every=50,burn.in=30000)

y<-unique(x)
z<-lapply(y,cpdag)


r=0
for(i in 1:length(y)) {
  if(all.equal(y[[i]],z[[i]])==TRUE)
    r<-r+1
}
length(y)/r
#setting up training and testing datasets
id <- sample(x = seq(1, dim(asia)[1], 1),
                     size = dim(asia)[1]*0.8,
                     replace = FALSE)
asia.train <- asia[id,]
asia.test <- asia[-id,]

bnprediction = as.numeric()
#fitting a model using Hill Climbing
bnmodel <- hc(asia.train,score = "bde",restart = 50)
bnmodelfit <- bn.fit(bnmodel,asia.train,method = 'mle') #max liklihood estimation
bngrain <- as.grain(bnmodelfit)
comp <- compile(bngrain)

bnmodel_true <- model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
bnmodelfit_true <- bn.fit(bnmodel_true, asia.train)
bngrain_true <- as.grain(bnmodelfit_true)
comp_true <- compile(bngrain_true)

bnpredict <- function(bntree, data, obs_variables, pred_variable) {
  for (i in 1:dim(data)[1]) {
    x <- NULL
    for (j in obs_variables) {
      x[j] <- if(data[i, j] == "yes") "yes" else "no"
    }
    evidence <- setEvidence(object = bntree,nodes = obs_variables,states=x)
    prob_dist_fit  <- querygrain(object = evidence,nodes = pred_variable)$S
```

```r
    bnprediction[i] <- if (prob_dist_fit["yes"] >= 0.5) "yes" else "no"
  }
  return(bnprediction)
}

# Predict S from Bayesian Network and test data observations
bnprediction <- bnpredict(bntree = comp,
                                     data = asia.test,
                                     obs_variables = c("A", "T", "L", "B", "E", "X", "D"),
                                     pred_variable = c("S"))
bnprediction_true <- bnpredict(bntree = comp_true,
                                        data = asia.test,
                                        obs_variables = c("A", "T", "L", "B", "E", "X", "D"),
                                        pred_variable = c("S"))

# Calculate confusion matricies
confusion_matrix_fit <- table(bnprediction, asia.test$S)
print(confusion_matrix_fit)
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_fit))/sum(confusion_matrix_fit)))

confusion_matrix_true <- table(bnprediction_true, asia.test$S)
print(confusion_matrix_true)
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_true))/sum(confusion_matrix_true)))

par(mfrow=c(1,2))
plot(bnmodel)
title("hill climb network")
plot(bnmodel_true)
title("True network")
par(mfrow=c(1,1))
markov_blanket = mb(bnmodel, c("S"))
markov_blanket_true = mb(bnmodel_true, c("S"))
bnpredict_mb <- bnpredict(comp, asia.test, markov_blanket, c("S"))
bnpredict_mb_true <- bnpredict(comp_true, asia.test, markov_blanket_true, c("S"))
confusion_matrix_fit <- table(bnpredict_mb, asia.test$S)
confusion_matrix_fit_true <- table(bnpredict_mb_true, asia.test$S)
print(confusion_matrix_fit)
print(confusion_matrix_fit_true)
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_fit))/sum(confusion_matrix_fit)))
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_fit_true))/sum(confusion_matrix_fit_
naiveBayes = model2network("[S][A|S][B|S][D|S][E|S][X|S][L|S][T|S]")
plot(naiveBayes, main = "Naive Bayes")

naiveBayes.fit <- bn.fit(naiveBayes, asia.train)
result_naive <- bnpredict(compile(as.grain(naiveBayes.fit)), asia.test, c("A", "T", "L", "B", "E", "X",
# Calculate confusion matricies
confusion_matrix_naive_bayes <- table(result_naive, asia.test$S)
print(confusion_matrix_naive_bayes)
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_naive_bayes))/sum(confusion_matrix_na
```

Advance Machine Learning Lab 2

Omkar Bhutra (omkbh878)

26 September 2019

The purpose of the lab is to put in practice some of the concepts covered in the lectures. To do so, you are asked to model the behavior of a robot that walks around a ring. The ring is divided into 10 sectors. At any given time point, the robot is in one of the sectors and decides with equal probability to stay in that sector or move to the next sector. You do not have direct observation of the robot. However, the robot is equipped with a tracking device that you can access. The device is not very accurate though: If the robot is in the sector i, then the device will report that the robot is in the sectors [i - 2; i + 2] with equal probability.

Question 1: Build a hidden Markov model (HMM) for the scenario described above

A robot moves around a ring which divided into 10 sectors. The robot is in one sector at any given time step and it's equally probable for the robot to stay in the state as it is to move to the next state. The robot has a tracking device. If the robot is in sector i, the tracking device will report that the robot is in the sectors [i - 2, i + 2] with equal probability i.e P = 0.2 for being in each position of that range.

Create transition matrix where each row consists of: $P(Z^t|Z^t - 1), t = 1, ..., 10$

```r
states <- paste("state",1:10,sep="")
symbols <- paste("symbol",1:10,sep="")
transition_vector <- c(0.5, 0.5, 0, 0, 0, 0, 0, 0, 0, 0,
                       0, 0.5, 0.5, 0, 0, 0, 0, 0, 0, 0,
                       0, 0, 0.5, 0.5, 0, 0, 0, 0, 0, 0,
                       0, 0, 0, 0.5, 0.5, 0, 0, 0, 0, 0,
                       0, 0, 0, 0, 0.5, 0.5, 0, 0, 0, 0,
                       0, 0, 0, 0, 0, 0.5, 0.5, 0, 0, 0,
                       0, 0, 0, 0, 0, 0, 0.5, 0.5, 0, 0,
                       0, 0, 0, 0, 0, 0, 0, 0.5, 0.5, 0,
                       0, 0, 0, 0, 0, 0, 0, 0, 0.5, 0.5,
                       0.5, 0, 0, 0, 0, 0, 0, 0, 0, 0.5)

emission_vector <- c(0.2, 0.2, 0.2, 0, 0, 0, 0, 0, 0.2, 0.2,
                     0.2, 0.2, 0.2, 0.2, 0, 0, 0, 0, 0, 0.2,
                     0.2, 0.2, 0.2, 0.2, 0.2, 0, 0, 0, 0, 0,
                     0, 0.2, 0.2, 0.2, 0.2, 0.2, 0, 0, 0, 0,
                     0, 0, 0.2, 0.2, 0.2, 0.2, 0.2, 0, 0, 0,
                     0, 0, 0, 0.2, 0.2, 0.2, 0.2, 0.2, 0, 0,
                     0, 0, 0, 0, 0.2, 0.2, 0.2, 0.2, 0.2, 0,
                     0, 0, 0, 0, 0, 0.2, 0.2, 0.2, 0.2, 0.2,
                     0.2, 0, 0, 0, 0, 0, 0.2, 0.2, 0.2, 0.2,
                     0.2, 0.2, 0, 0, 0, 0, 0, 0.2, 0.2, 0.2)


transition_matrix <- matrix(data = transition_vector,nrow = 10,ncol = 10)

emission_matrix <- matrix(data = emission_vector,nrow = 10,ncol = 10)
# States are the hidden variables
# Symbols are the observable variables
hmm <- initHMM(States = states,
```

```
        Symbols = symbols,
        startProbs = rep(0.1,10),
        transProbs = transition_matrix,
        emissionProbs = emission_matrix)
hmm
```

```
## $States
##  [1] "state1"  "state2"  "state3"  "state4"  "state5"  "state6"  "state7"
##  [8] "state8"  "state9"  "state10"
##
## $Symbols
##  [1] "symbol1"  "symbol2"  "symbol3"  "symbol4"  "symbol5"  "symbol6"
##  [7] "symbol7"  "symbol8"  "symbol9"  "symbol10"
##
## $startProbs
##  state1  state2  state3  state4  state5  state6  state7  state8  state9
##     0.1     0.1     0.1     0.1     0.1     0.1     0.1     0.1     0.1
## state10
##     0.1
##
## $transProbs
##         to
## from      state1 state2 state3 state4 state5 state6 state7 state8 state9
##    state1    0.5    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
##    state2    0.5    0.5    0.0    0.0    0.0    0.0    0.0    0.0    0.0
##    state3    0.0    0.5    0.5    0.0    0.0    0.0    0.0    0.0    0.0
##    state4    0.0    0.0    0.5    0.5    0.0    0.0    0.0    0.0    0.0
##    state5    0.0    0.0    0.0    0.5    0.5    0.0    0.0    0.0    0.0
##    state6    0.0    0.0    0.0    0.0    0.5    0.5    0.0    0.0    0.0
##    state7    0.0    0.0    0.0    0.0    0.0    0.5    0.5    0.0    0.0
##    state8    0.0    0.0    0.0    0.0    0.0    0.0    0.5    0.5    0.0
##    state9    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.5    0.5
##    state10   0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.5
##         to
## from      state10
##    state1     0.5
##    state2     0.0
##    state3     0.0
##    state4     0.0
##    state5     0.0
##    state6     0.0
##    state7     0.0
##    state8     0.0
##    state9     0.0
##    state10    0.5
##
## $emissionProbs
##         symbols
## states    symbol1 symbol2 symbol3 symbol4 symbol5 symbol6 symbol7 symbol8
##    state1     0.2     0.2     0.2     0.0     0.0     0.0     0.0     0.0
##    state2     0.2     0.2     0.2     0.2     0.0     0.0     0.0     0.0
##    state3     0.2     0.2     0.2     0.2     0.2     0.0     0.0     0.0
##    state4     0.0     0.2     0.2     0.2     0.2     0.2     0.0     0.0
##    state5     0.0     0.0     0.2     0.2     0.2     0.2     0.2     0.0
```

```
##    state6       0.0      0.0      0.0      0.2      0.2      0.2      0.2      0.2
##    state7       0.0      0.0      0.0      0.0      0.2      0.2      0.2      0.2
##    state8       0.0      0.0      0.0      0.0      0.0      0.2      0.2      0.2
##    state9       0.2      0.0      0.0      0.0      0.0      0.0      0.2      0.2
##    state10      0.2      0.2      0.0      0.0      0.0      0.0      0.0      0.2
##           symbols
## states    symbol9 symbol10
##    state1       0.2      0.2
##    state2       0.0      0.2
##    state3       0.0      0.0
##    state4       0.0      0.0
##    state5       0.0      0.0
##    state6       0.0      0.0
##    state7       0.2      0.0
##    state8       0.2      0.2
##    state9       0.2      0.2
##    state10      0.2      0.2
```

Question 2: Simulate the HMM for 100 time steps.

```r
hmm_sim <- simHMM(hmm = hmm,length = 100)
hmm_sim
```

```
## $states
##   [1] "state5"  "state5"  "state4"  "state3"  "state3"  "state3"  "state3"
##   [8] "state2"  "state2"  "state2"  "state1"  "state10" "state10" "state10"
##  [15] "state9"  "state8"  "state8"  "state7"  "state7"  "state7"  "state6"
##  [22] "state6"  "state6"  "state5"  "state4"  "state4"  "state4"  "state3"
##  [29] "state3"  "state3"  "state3"  "state3"  "state2"  "state1"  "state1"
##  [36] "state1"  "state1"  "state1"  "state10" "state9"  "state8"  "state8"
##  [43] "state8"  "state8"  "state8"  "state7"  "state6"  "state5"  "state5"
##  [50] "state4"  "state3"  "state2"  "state1"  "state10" "state10" "state9"
##  [57] "state8"  "state7"  "state7"  "state7"  "state6"  "state6"  "state6"
##  [64] "state5"  "state5"  "state4"  "state3"  "state2"  "state2"  "state1"
##  [71] "state10" "state10" "state9"  "state9"  "state9"  "state8"  "state8"
##  [78] "state8"  "state8"  "state8"  "state7"  "state7"  "state7"  "state6"
##  [85] "state6"  "state6"  "state6"  "state5"  "state5"  "state5"  "state4"
##  [92] "state3"  "state2"  "state2"  "state2"  "state2"  "state2"  "state2"
##  [99] "state2"  "state2"
##
## $observation
##   [1] "symbol7"  "symbol5"  "symbol2"  "symbol2"  "symbol1"  "symbol1"
##   [7] "symbol1"  "symbol2"  "symbol4"  "symbol2"  "symbol10" "symbol10"
##  [13] "symbol1"  "symbol9"  "symbol1"  "symbol10" "symbol7"  "symbol8"
##  [19] "symbol8"  "symbol5"  "symbol8"  "symbol7"  "symbol4"  "symbol4"
##  [25] "symbol2"  "symbol4"  "symbol2"  "symbol4"  "symbol1"  "symbol1"
##  [31] "symbol5"  "symbol1"  "symbol3"  "symbol3"  "symbol3"  "symbol9"
##  [37] "symbol10" "symbol2"  "symbol2"  "symbol1"  "symbol9"  "symbol7"
##  [43] "symbol10" "symbol8"  "symbol10" "symbol8"  "symbol7"  "symbol5"
##  [49] "symbol6"  "symbol4"  "symbol4"  "symbol2"  "symbol2"  "symbol8"
##  [55] "symbol1"  "symbol1"  "symbol7"  "symbol8"  "symbol6"  "symbol8"
##  [61] "symbol4"  "symbol6"  "symbol6"  "symbol5"  "symbol7"  "symbol6"
```

```
##  [67] "symbol1"  "symbol10" "symbol1"  "symbol3"  "symbol9"  "symbol9"
##  [73] "symbol1"  "symbol10" "symbol9"  "symbol10" "symbol6"  "symbol8"
##  [79] "symbol9"  "symbol10" "symbol8"  "symbol9"  "symbol8"  "symbol7"
##  [85] "symbol8"  "symbol4"  "symbol6"  "symbol3"  "symbol3"  "symbol4"
##  [91] "symbol5"  "symbol2"  "symbol10" "symbol4"  "symbol10" "symbol4"
##  [97] "symbol4"  "symbol2"  "symbol4"  "symbol2"
```

Question 3: Discard the hidden states from the sample obtained above. Use the remaining observationsto compute the filtered and smoothed probability distributions for each of the 100 time points. Compute also the most probable path. Question 4: Compute the accuracy of the filtered and smoothed probability distributions, and of themost probable path. That is, compute the percentage of the true hidden states that are guessed by each method

Hint: Note that the function forward in the HMM package returns probabilities in log scale. You may need to use the functions exp and prop.table in order to obtain a normalized probability distribution. You may also want to use the functions apply and which.max to find out the most probable states. Finally, recall that you can compare two vectors A and B elementwise as A==B, and that the function table will count the number of times that the different elements in a vector occur in the vector.

Forward probabilities can be found in a hidden markov model with hidden states X upto observation at time t defined as the probability of observing the sequence of observations $e_1, ..., e_k$ and that the state at time t is X. That is: $f[X, t] := P(X | E_1 = e_1, ..., E_k = e_k)$

```r
robot <- function(hiddenmarkovmodel,pars){
  hmm_sim <- simHMM(hmm = hmm,length = 100)
  hmm_obs <- hmm_sim$observation
  hmm_states <- hmm_sim$states
  #filter: forward function does the filtering and returns log probabilities
  log_filter = forward(hmm = hmm,observation = hmm_obs)
  filter = exp(log_filter)
  #normalised probability distribution
  filternormalised <- prop.table(filter,margin = 2)
  # find out the most probable states
  filternormalised_probable <- apply(filternormalised,MARGIN = 2, FUN = which.max)
  accuracy_filtering <- sum(paste("state", filternormalised_probable, sep ="")
                            ==hmm_states)/length(hmm_states)
  #filternormalised
  #accuracy_filtering
  #smoothing using function posterior
  smooth <- posterior(hmm,hmm_obs)
  smoothnormalised <- prop.table(smooth, margin = 2)
  smoothnormalised_probable <- apply(smoothnormalised,MARGIN = 2, FUN = which.max)
  accuracy_smooth <- sum(paste("state", smoothnormalised_probable, sep ="")
                          ==hmm_states)/length(hmm_states)
  #smoothnormalised
  #accuracy_smooth
  #Finding the most probable path using viterbi algorithm
  probable_path <- viterbi(hmm = hmm, observation = hmm_obs)
  accuracy_probable_path <- sum(probable_path == hmm_states)/ length(hmm_states)
  probable_path #most probable path
  #accuracy_probable_path
  if(pars == "filter"){
  return(filternormalised)
}
```

```
if(pars == "smooth"){
  return(smoothnormalised)
}
if(pars == "ProbablePath"){
  return(probable_path)
}
if(pars == "accuracy"){
  return(c(accuracy_filtering = accuracy_filtering,
          accuracy_smooth = accuracy_smooth,
          accuracy_probable_path = accuracy_probable_path))
}
}
```

Question 5: Repeat the previous exercise with different simulated samples. In general, the smoothed distributions should be more accurate than the filtered distributions. Why ? In general, the smoothed distributions should be more accurate than the most probable paths, too. Why ?

```
robot(hmm,"ProbablePath")
```

```
##   [1] "state1"  "state10" "state9"  "state8"  "state8"  "state8"  "state8"
##   [8] "state8"  "state8"  "state8"  "state8"  "state7"  "state6"  "state6"
##  [15] "state6"  "state6"  "state5"  "state5"  "state5"  "state4"  "state3"
##  [22] "state3"  "state3"  "state2"  "state1"  "state10" "state9"  "state9"
##  [29] "state8"  "state8"  "state7"  "state6"  "state6"  "state5"  "state5"
##  [36] "state5"  "state4"  "state3"  "state3"  "state3"  "state3"  "state3"
##  [43] "state2"  "state1"  "state1"  "state1"  "state1"  "state1"  "state10"
##  [50] "state9"  "state8"  "state7"  "state6"  "state5"  "state4"  "state4"
##  [57] "state4"  "state3"  "state2"  "state1"  "state1"  "state10" "state9"
##  [64] "state8"  "state8"  "state8"  "state8"  "state8"  "state7"  "state7"
##  [71] "state6"  "state6"  "state5"  "state5"  "state4"  "state3"  "state2"
##  [78] "state1"  "state10" "state9"  "state9"  "state8"  "state7"  "state6"
##  [85] "state5"  "state5"  "state4"  "state4"  "state4"  "state4"  "state3"
##  [92] "state2"  "state1"  "state10" "state9"  "state9"  "state9"  "state8"
##  [99] "state8"  "state8"
```
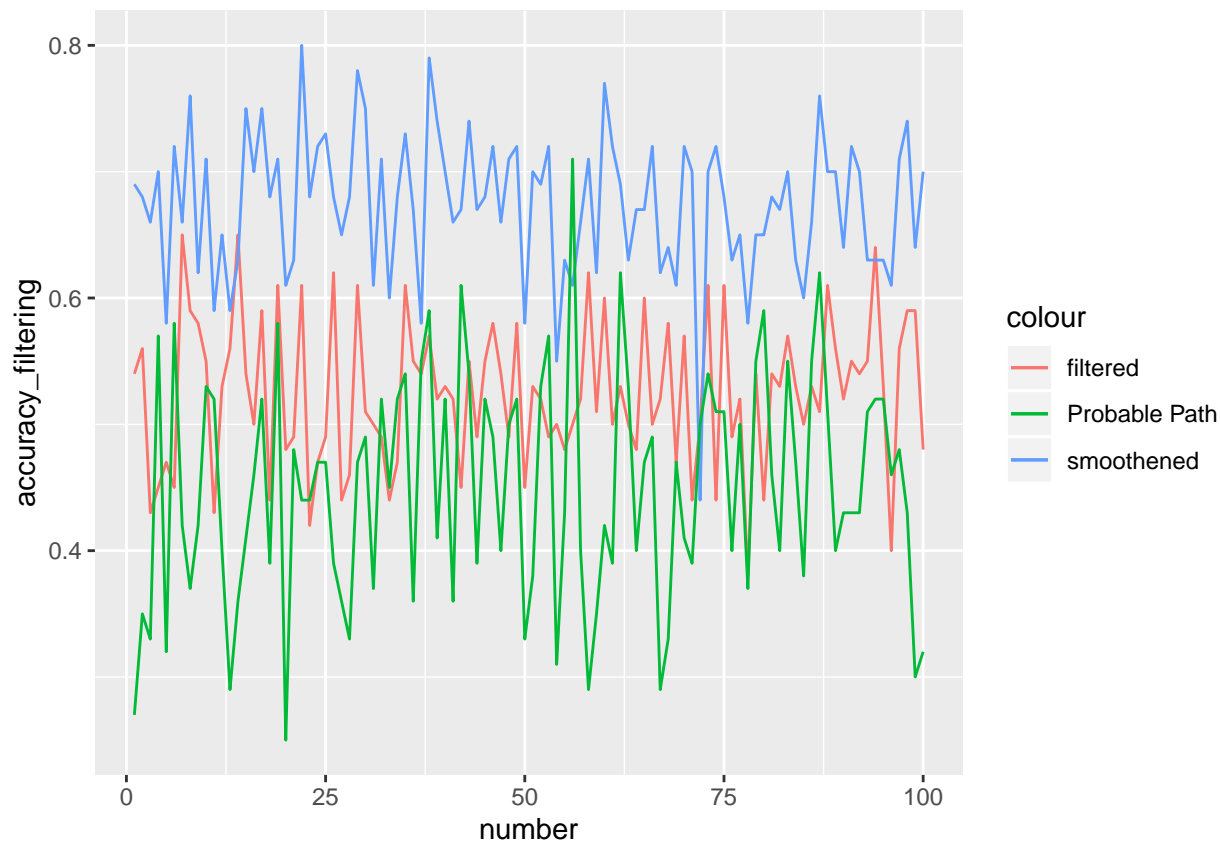
```
acc <- sapply(1:100,FUN = function(x){robot(hmm,"accuracy")})
acc <- data.frame(t(acc))
acc$number <- 1:100

ggplot(data = acc) + geom_line(aes(x=number,y=accuracy_filtering,col="filtered"))+
                    geom_line(aes(x=number,y=accuracy_smooth,col="smoothened"))+
                    geom_line(aes(x=number,y=accuracy_probable_path,col="Probable Path"))
```

```r
colMeans(acc[,-4])
```

```
accuracy_filtering   accuracy_smooth accuracy_probable_path
0.5246   0.6736   0.4499
```

The smoothed distribution is using more information i.e the whole set of observed emission x[0:T] whereas filtered only uses data emitted up to that point x[0:t] The most probable path generated by the viterbi algorithm is constrained by the transitions between states in the hidden variables. The smooth distribution approximates the most probable state and hence can make jumps from one state to another when predicting current state.

**Question 6: Is it true that the more observations you have the better you know where the robot is ?**

Hint: You may want to compute the entropy of the filtered distributions with the function entropy.empirical of the package entropy.

```r
hmm_filter <- robot(hmm,"filter")
hmm_filter_entropy <- data.frame(index = 1:100, Entropy = apply(hmm_filter, MARGIN = 2,
                                 FUN = entropy::entropy.empirical))
ggplot(hmm_filter_entropy,aes(x = index, y = Entropy))+
  geom_line()+ggtitle("Entropy of filtered distribution")
```

Entropy of filtered distribution
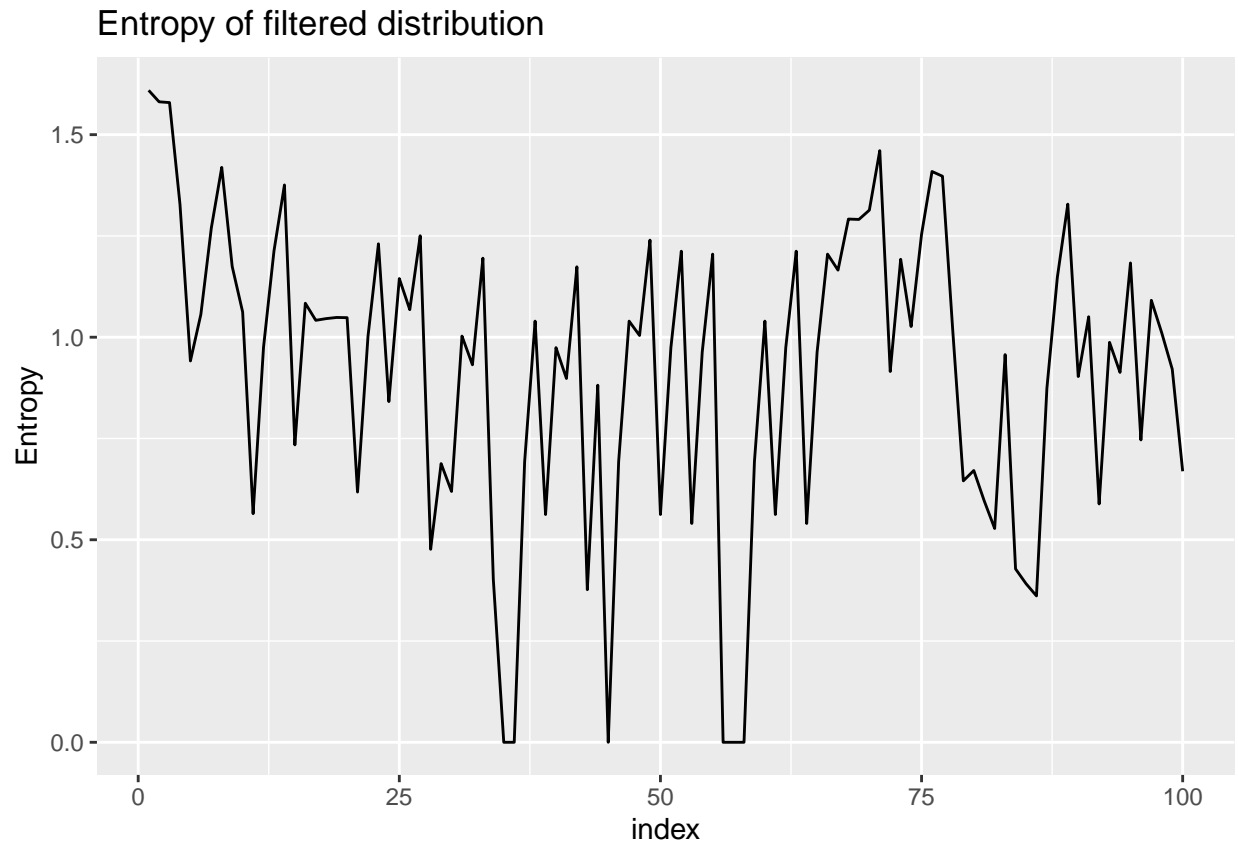
The entropy is random even when we increase the number of observations added to the hidden markov model. This is because the HMM is Markovian and only depends on the previous observation.

Question 7: Consider any of the samples above of length 100. Compute the probabilities of the hidden states for the time step 101.

```
posterior <- hmm_filter[,100]
# matrix multiplication
probability <- as.data.frame(hmm$transProbs %*% posterior)
ggplot(probability)+geom_line(aes(x=1:10,y=V1))+ggtitle("probability of the hidden state and symbol for
```

probability of the hidden state and symbol for the timestep 101
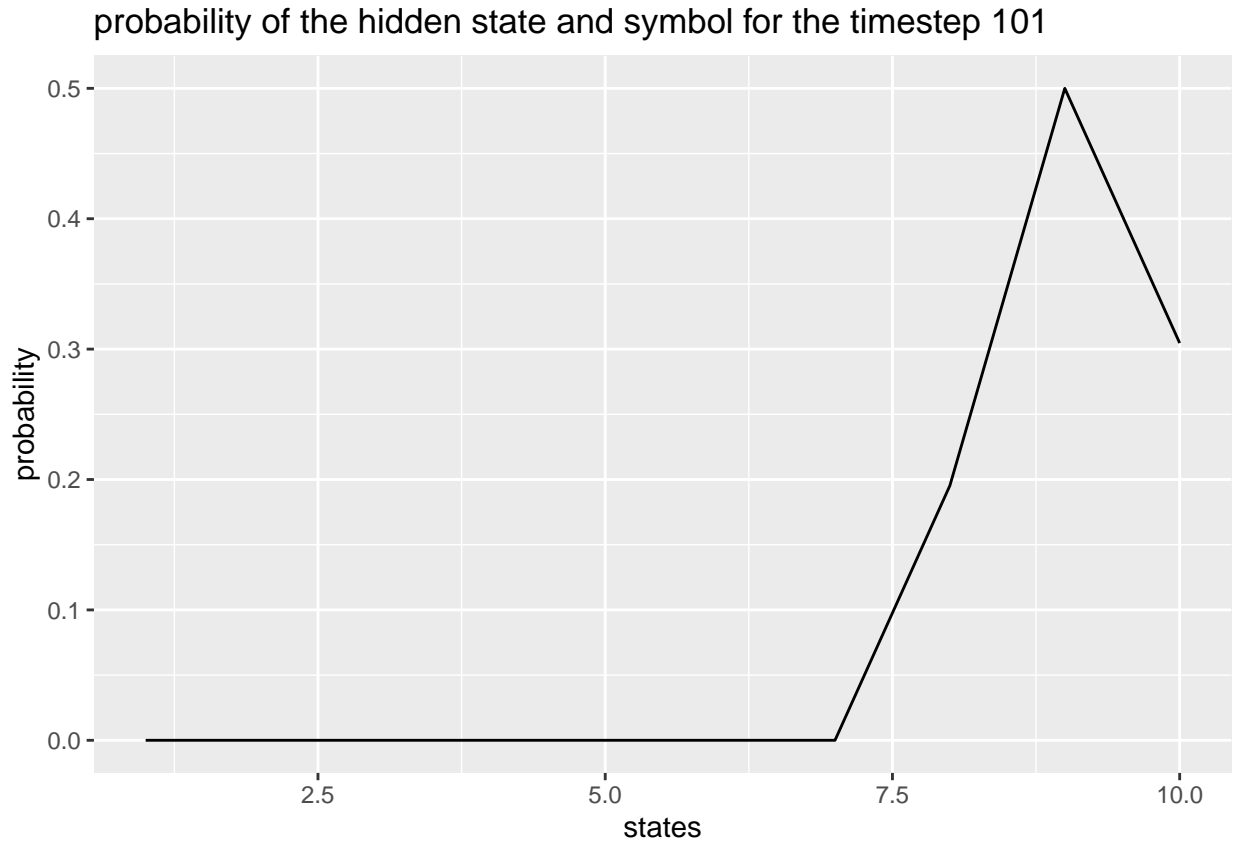
The Markovian assumption is that only relevant state is the current state. All previous states feeds though the 100 states and provides us information about the states and observations. On multiplication with transition probabilities to get the probability of the hidden state and symbol for the timestep 101. That the entropy of the filtered distributions does not decrease monotonically. It has to do with the fact that you get noisy observations and, thus, you will more often than not be uncertain as to where the robot is.

```r
knitr::opts_chunk$set(echo = TRUE)
library("dplyr")
library("ggplot2")
library("gRain")
library("bnlearn")
library("HMM")
library("entropy")
states <- paste("state",1:10,sep="")
symbols <- paste("symbol",1:10,sep="")
transition_vector <- c(0.5, 0.5, 0, 0, 0, 0, 0, 0, 0, 0,
                       0, 0.5, 0.5, 0, 0, 0, 0, 0, 0, 0,
                       0, 0, 0.5, 0.5, 0, 0, 0, 0, 0, 0,
                       0, 0, 0, 0.5, 0.5, 0, 0, 0, 0, 0,
                       0, 0, 0, 0, 0.5, 0.5, 0, 0, 0, 0,
                       0, 0, 0, 0, 0, 0.5, 0.5, 0, 0, 0,
                       0, 0, 0, 0, 0, 0, 0.5, 0.5, 0, 0,
                       0, 0, 0, 0, 0, 0, 0, 0.5, 0.5, 0,
                       0, 0, 0, 0, 0, 0, 0, 0, 0.5, 0.5,
                       0.5, 0, 0, 0, 0, 0, 0, 0, 0, 0.5)
```

```r
emission_vector <- c(0.2, 0.2, 0.2, 0, 0, 0, 0, 0, 0.2, 0.2,
                     0.2, 0.2, 0.2, 0.2, 0, 0, 0, 0, 0, 0.2,
                     0.2, 0.2, 0.2, 0.2, 0.2, 0, 0, 0, 0, 0,
                     0, 0.2, 0.2, 0.2, 0.2, 0.2, 0, 0, 0, 0,
                     0, 0, 0.2, 0.2, 0.2, 0.2, 0.2, 0, 0, 0,
                     0, 0, 0, 0.2, 0.2, 0.2, 0.2, 0.2, 0, 0,
                     0, 0, 0, 0, 0.2, 0.2, 0.2, 0.2, 0.2, 0,
                     0, 0, 0, 0, 0, 0.2, 0.2, 0.2, 0.2, 0.2,
                     0.2, 0, 0, 0, 0, 0, 0.2, 0.2, 0.2, 0.2,
                     0.2, 0.2, 0, 0, 0, 0, 0, 0.2, 0.2, 0.2)


transition_matrix <- matrix(data = transition_vector,nrow = 10,ncol = 10)

emission_matrix <- matrix(data = emission_vector,nrow = 10,ncol = 10)
# States are the hidden variables
# Symbols are the observable variables
hmm <- initHMM(States = states,
       Symbols = symbols,
       startProbs = rep(0.1,10),
       transProbs = transition_matrix,
       emissionProbs = emission_matrix)
hmm
hmm_sim <- simHMM(hmm = hmm,length = 100)
hmm_sim
robot <- function(hiddenmarkovmodel,pars){
  hmm_sim <- simHMM(hmm = hmm,length = 100)
  hmm_obs <- hmm_sim$observation
  hmm_states <- hmm_sim$states
  #filter: forward function does the filtering and returns log probabilities
  log_filter = forward(hmm = hmm,observation = hmm_obs)
  filter = exp(log_filter)
  #normalised probability distribution
  filternormalised <- prop.table(filter,margin = 2)
  # find out the most probable states
  filternormalised_probable <- apply(filternormalised,MARGIN = 2, FUN = which.max)
  accuracy_filtering <- sum(paste("state", filternormalised_probable, sep ="")
                           ==hmm_states)/length(hmm_states)
  #filternormalised
  #accuracy_filtering
  #smoothing using function posterior
  smooth <- posterior(hmm,hmm_obs)
  smoothnormalised <- prop.table(smooth, margin = 2)
  smoothnormalised_probable <- apply(smoothnormalised,MARGIN = 2, FUN = which.max)
  accuracy_smooth <- sum(paste("state", smoothnormalised_probable, sep ="")
                        ==hmm_states)/length(hmm_states)
  #smoothnormalised
  #accuracy_smooth
  #Finding the most probable path using viterbi algorithm
  probable_path <- viterbi(hmm = hmm, observation = hmm_obs)
  accuracy_probable_path <- sum(probable_path == hmm_states)/ length(hmm_states)
  probable_path #most probable path
  #accuracy_probable_path
```

```r
  if(pars == "filter"){
  return(filternormalised)
}
if(pars == "smooth"){
  return(smoothnormalised)
}
if(pars == "ProbablePath"){
  return(probable_path)
}
if(pars == "accuracy"){
  return(c(accuracy_filtering = accuracy_filtering,
           accuracy_smooth = accuracy_smooth,
           accuracy_probable_path = accuracy_probable_path))
}
}
robot(hmm,"filter")
robot(hmm,"smooth")
robot(hmm,"ProbablePath")

acc <- sapply(1:100,FUN = function(x){robot(hmm,"accuracy")})
acc <- data.frame(t(acc))
acc$number <- 1:100

ggplot(data = acc) + geom_line(aes(x=number,y=accuracy_filtering,col="filtered"))+
                     geom_line(aes(x=number,y=accuracy_smooth,col="smoothened"))+
                     geom_line(aes(x=number,y=accuracy_probable_path,col="Probable Path"))

colMeans(acc[,-4])
hmm_filter <- robot(hmm,"filter")
hmm_filter_entropy <- data.frame(index = 1:100, Entropy = apply(hmm_filter, MARGIN = 2,
                                 FUN = entropy::entropy.empirical))
ggplot(hmm_filter_entropy,aes(x = index, y = Entropy))+
  geom_line()+ggtitle("Entropy of filtered distribution")
posterior <- hmm_filter[,100]
# matrix multiplication
probability <- as.data.frame(hmm$transProbs %*% posterior)
ggplot(probability)+geom_line(aes(x=1:10,y=V1))+ggtitle("probability of the hidden state and symbol for
# Build the HMM.
library(HMM)
#set.seed(123)
States<-1:100
Symbols<-1:2 # 1=door

transProbs<-matrix(rep(0,length(States)*length(States)), nrow=length(States), ncol=length(States), byrow
for(i in 1:99){
  transProbs[i,i]<-.1
  transProbs[i,i+1]<-.9
}

emissionProbs<-matrix(rep(0,length(States)*length(Symbols)), nrow=length(States), ncol=length(Symbols),
for(i in States){
  if(i %in% c(10,11,12,20,21,22,30,31,32)){
    emissionProbs[i,1]<-.9
```

```
    emissionProbs[i,2]<-.1
  }
  else{
    emissionProbs[i,1]<-.1
    emissionProbs[i,2]<-.9
  }
}

startProbs<-rep(1/100,100)
hmm<-initHMM(States,Symbols,startProbs,transProbs,emissionProbs)

# If the robot observes a door, it can be in front of any of the three doors. If it then observes a lon
# sequence of non-doors, then it know that it was in front of the third door.

obs<-c(1,1,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2)
pt<-prop.table(exp(forward(hmm,obs)),2)

which.maxima<-function(x){ # This function is needed since which.max only returns the first maximum.
  return(which(x==max(x)))
}

apply(pt,2,which.maxima)
```

Advance Machine Learning Lab 3

Omkar Bhutra (omkbh878)

7 October 2019

Q1. The purpose of the lab is to put in practice some of the concepts covered in the lectures. To do so, you are asked to implement the particle filter for robot localization. For the particle filter algorithm, please check Section 13.3.4 of Bishops book and/or the slides for the last lecture on state space models (SSMs). The robot moves along the horizontal axis according to the following SSM:

Transition Model: $p(z_t|z_{t-1}) = (N(z_t|z_{t-1}, 1) + N(z_t|z_{t-1} + 1, 1) + N(z_t|z_{t-1} + 2, 1))/3$

Emission Model: $p(x_t|z_t) = (N(x_t|z_t, 1) + N(x_t|z_t - 1, 1) + N(x_t|z_t + 1, 1))/3$

Initial Model: $p(z_1) = Uniform(0, 100)$

A) Implement the SSM above. Simulate it for T = 100 time steps to obtain z 1:100 (i.e., states) and x 1:100 (i.e., observations). Use the observations (i.e., sensor readings) to identify the state (i.e., robot location) via particle filtering. Use 100 particles. Show the particles, the expected location and the true location for the first and last time steps, as well as for two intermediate time steps of your choice.

```
initModel <- function(len){
  x <- runif(len,0,100)
  return(x)
}

transitionmodel <- function(zt){
  probs = rep(1/3,3)
  draw = sample(1:3,1,prob = probs)
  if(draw==1){
    normTrans <- rnorm(1,zt,transition_sd)
  }
  else if(draw==2){
    normTrans <- rnorm(1,zt+1,transition_sd)
  }
  else{
    normTrans <- rnorm(1,zt+2,transition_sd)
  }
  return(normTrans)
}

emmisionmodel <- function(zt,emission_sd){
  probs = rep(1/3,3)
```

```r
  draw = sample(1:3,1,prob = probs)

  if(draw==1){
    normEmis <- rnorm(1,zt,emission_sd)
  }
  else if(draw==2){
    normEmis <- rnorm(1,zt-1,emission_sd)
  }
  else{
    normEmis <- rnorm(1,zt+1,emission_sd)
  }
  return(normEmis)
}

emission_density <- function(xt,zt){
  x <- (dnorm(xt,zt,emission_sd)+dnorm(xt,zt-1,emission_sd)+dnorm(xt,zt+1,emission_sd))/3
  return(x)
}


emission_sd = 1
transition_sd = 1

ssm <- function(emission_sd,transition_sd){
T = 100
particles = 100
zt = xt = error = rep(NA,T)

# for zt and xt
zt[1] = initModel(1)
for(i in 2:T){
  zt[i] <- transitionmodel(zt[i-1])
  xt[i] <- emmisionmodel(zt[i],emission_sd)
}

initParticles <- initModel(particles)
particleWeights<- rep(1/particles,particles)
estimation <- c()
Particles25 = Particles75 = NA

for(i in 2:T){
  newParticles <- sample(1:particles,particles,prob=particleWeights,replace = T)
  initParticles <- sapply(initParticles[newParticles],transitionmodel)

  if(i == 25){
    Particles25 <- c(initParticles)
  }
  else if(i == 75){
    Particles75 <- c(initParticles)
  }


  for(j in 1:particles){
```

```
    particleWeights[j] <- emission_density(xt[i],initParticles[j])
  }
  #Normalise
  particleWeights <- particleWeights/sum(particleWeights)

  Ezt <- sum(particleWeights*initParticles)
  error[i] <- abs(zt[i]-Ezt)
  estimation[i] <- sum(particleWeights * initParticles)
}

data = data.frame(zt,xt,estimation,particles = initModel(particles),Particles25,Particles75,initParticle

ggplot(data,aes(x=c(1:T),y=value,color=variable,xlab="timestep"))+
  geom_line(aes(y=data$zt,col='True Zt'))+
  geom_line(aes(y=data$xt,col='True Xt'))+
  geom_line(aes(y=data$estimation,col='Estimation'))+
  geom_point(aes(x=rep(1,T),y=data$particles,col='particle'))+
  geom_point(aes(x=rep(25,T),y=data$Particles25,col='particle at 25'))+
  geom_point(aes(x=rep(75,T),y=data$Particles75,col='particle at 75'))+
  geom_point(aes(x=rep(T,T),y=data$initParticles,col='particle'))+
  ggtitle(paste('At SD = ',emission_sd))
}
ssm(emission_sd ,transition_sd )
```


Intermediate particles number 25 and 75 are chosen and their locations are mapped in the figure.
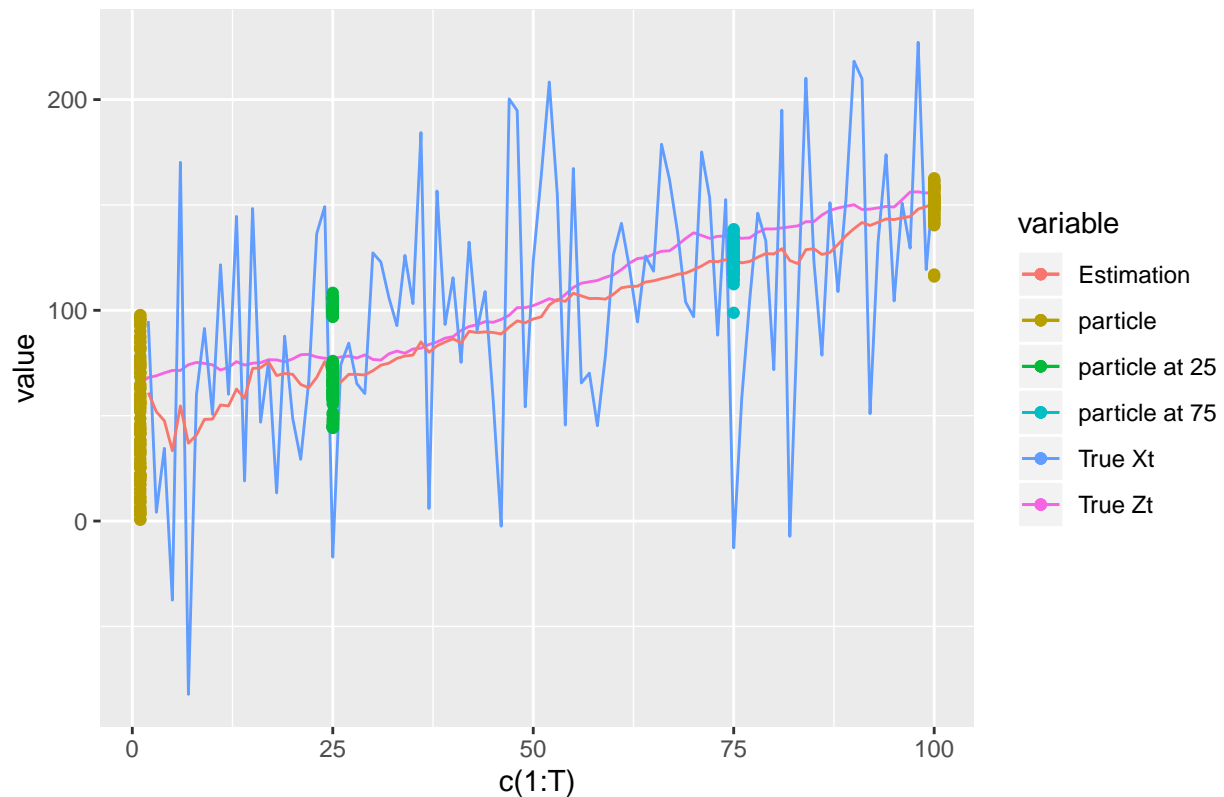
B)Repeat the exercise above replacing the standard deviation of the emission model with 5 and then with 50. Comment on how this affects the results.

```
emission_sd = 5
ssm(emission_sd ,transition_sd )
```
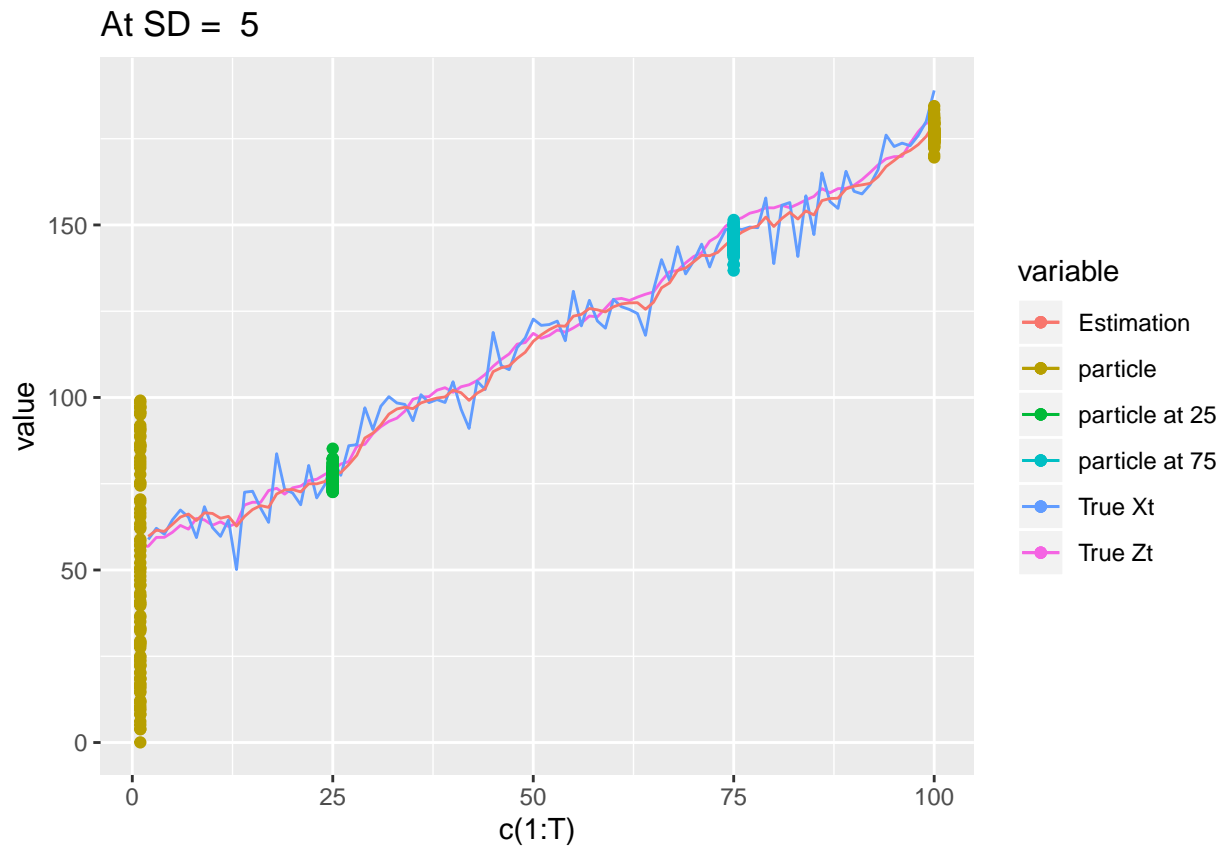

At SD = 5

```
emission_sd = 50
ssm(emission_sd ,transition_sd )
```

At SD = 50

Increasing the standard deviation not only makes the true x_t scattered but also scatters the particles both intermediate and endstep.

C) Finally, show and explain what happens when the weights in the particle filter are always equal to 1, i.e. there is no correction.

```r
ssm <- function(emission_sd,transition_sd){
T = 100
particles = 100
zt = xt = error = rep(NA,T)

# for zt and xt
zt[1] = initModel(1)
for(i in 2:T){
  zt[i] <- transitionmodel(zt[i-1])
  xt[i] <- emmisionmodel(zt[i],emission_sd)
}

initParticles <- initModel(particles)
particleWeights<- rep(1/particles,particles)
estimation <- c()
Particles25 = Particles75 = NA

for(i in 2:T){
```

```r
    newParticles <- sample(1:particles,particles,prob=particleWeights,replace = T)
    initParticles <- sapply(initParticles[newParticles],transitionmodel)

    if(i == 25){
      Particles25 <- c(initParticles)
    }
    else if(i == 75){
      Particles75 <- c(initParticles)
    }


    for(j in 1:particles){
      particleWeights[j] <- 1
    }
    #Normalise
    particleWeights <- particleWeights/sum(particleWeights)

    Ezt <- sum(particleWeights*initParticles)
    error[i] <- abs(zt[i]-Ezt)
    estimation[i] <- sum(particleWeights * initParticles)
}

data = data.frame(zt,xt,estimation,particles = initModel(particles),Particles25,Particles75,initParticl

ggplot(data,aes(x=c(1:T),y=value,color=variable,xlab="timestep"))+
  geom_line(aes(y=data$zt,col='True Zt'))+
  geom_line(aes(y=data$xt,col='True Xt'))+
  geom_line(aes(y=data$estimation,col='Estimation'))+
  geom_point(aes(x=rep(1,T),y=data$particles,col='particle'))+
  geom_point(aes(x=rep(25,T),y=data$Particles25,col='particle at 25'))+
  geom_point(aes(x=rep(75,T),y=data$Particles75,col='particle at 75'))+
  geom_point(aes(x=rep(T,T),y=data$initParticles,col='particle'))+
  ggtitle(paste('At SD = ',emission_sd))
}

emission_sd = 1
ssm(emission_sd ,transition_sd )
```
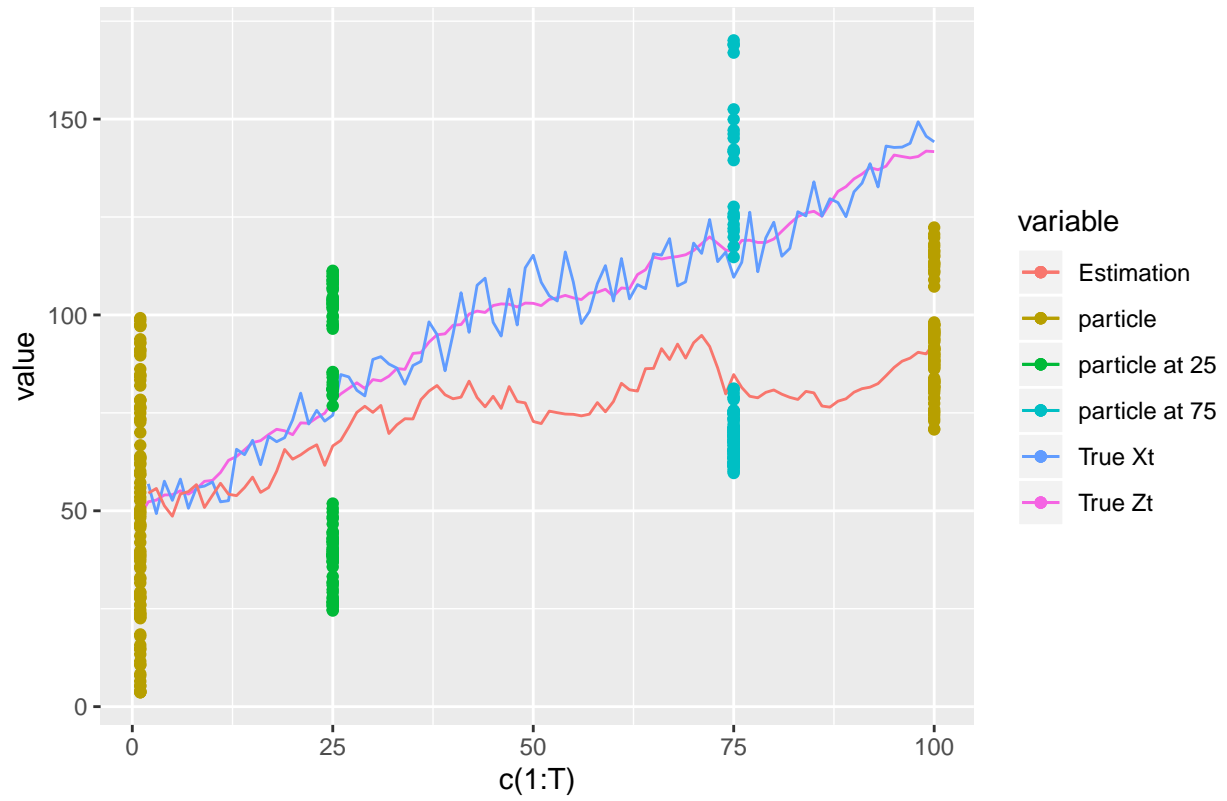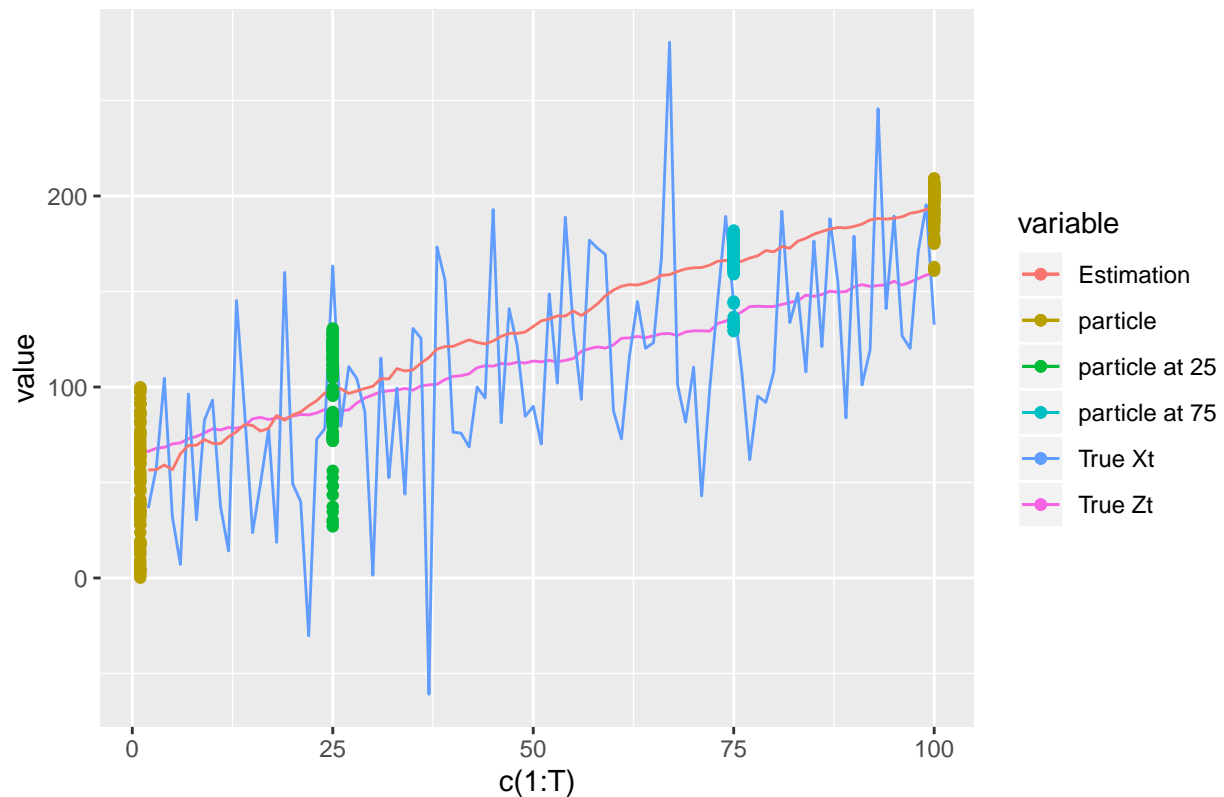
At SD = 1

```
emission_sd = 5
ssm(emission_sd ,transition_sd )
```

At SD = 5


```
emission_sd = 50
ssm(emission_sd ,transition_sd )
```

At SD = 50

It is seen that the estimation and true values do not match and accuracy is reduced. This change is consistent from all the standard deviation runs of 1,5 and 50. value of the particle at all timesteps are scattered as compared to earlier runs. It reduces the quality of the filtering.

```r
knitr::opts_chunk$set(echo = TRUE)
library("dplyr")
library("ggplot2")
library("KFAS")
initModel <- function(len){
  x <- runif(len,0,100)
  return(x)
}

transitionmodel <- function(zt){
  probs = rep(1/3,3)
  draw = sample(1:3,1,prob = probs)
  if(draw==1){
    normTrans <- rnorm(1,zt,transition_sd)
  }
  else if(draw==2){
    normTrans <- rnorm(1,zt+1,transition_sd)
  }
  else{
    normTrans <- rnorm(1,zt+2,transition_sd)
  }
  return(normTrans)
```

```r
}

emmisionmodel <- function(zt,emission_sd){
  probs = rep(1/3,3)
  draw = sample(1:3,1,prob = probs)

  if(draw==1){
    normEmis <- rnorm(1,zt,emission_sd)
  }
  else if(draw==2){
    normEmis <- rnorm(1,zt-1,emission_sd)
  }
  else{
    normEmis <- rnorm(1,zt+1,emission_sd)
  }
  return(normEmis)
}

emission_density <- function(xt,zt){
  x <- (dnorm(xt,zt,emission_sd)+dnorm(xt,zt-1,emission_sd)+dnorm(xt,zt+1,emission_sd))/3
  return(x)
}


emission_sd = 1
transition_sd = 1

ssm <- function(emission_sd,transition_sd){
T = 100
particles = 100
zt = xt = error = rep(NA,T)

# for zt and xt
zt[1] = initModel(1)
for(i in 2:T){
  zt[i] <- transitionmodel(zt[i-1])
  xt[i] <- emmisionmodel(zt[i],emission_sd)
}

initParticles <- initModel(particles)
particleWeights<- rep(1/particles,particles)
estimation <- c()
Particles25 = Particles75 = NA

for(i in 2:T){
  newParticles <- sample(1:particles,particles,prob=particleWeights,replace = T)
  initParticles <- sapply(initParticles[newParticles],transitionmodel)

  if(i == 25){
    Particles25 <- c(initParticles)
  }
  else if(i == 75){
    Particles75 <- c(initParticles)
```

```r
  }

  for(j in 1:particles){
    particleWeights[j] <- emission_density(xt[i],initParticles[j])
  }
  #Normalise
  particleWeights <- particleWeights/sum(particleWeights)

  Ezt <- sum(particleWeights*initParticles)
  error[i] <- abs(zt[i]-Ezt)
  estimation[i] <- sum(particleWeights * initParticles)
}

data = data.frame(zt,xt,estimation,particles = initModel(particles),Particles25,Particles75,initParticl

ggplot(data,aes(x=c(1:T),y=value,color=variable,xlab="timestep"))+
  geom_line(aes(y=data$zt,col='True Zt'))+
  geom_line(aes(y=data$xt,col='True Xt'))+
  geom_line(aes(y=data$estimation,col='Estimation'))+
  geom_point(aes(x=rep(1,T),y=data$particles,col='particle'))+
  geom_point(aes(x=rep(25,T),y=data$Particles25,col='particle at 25'))+
  geom_point(aes(x=rep(75,T),y=data$Particles75,col='particle at 75'))+
  geom_point(aes(x=rep(T,T),y=data$initParticles,col='particle'))+
  ggtitle(paste('At SD = ',emission_sd))
}
ssm(emission_sd ,transition_sd )
emission_sd = 5
ssm(emission_sd ,transition_sd )

emission_sd = 50
ssm(emission_sd ,transition_sd )
ssm <- function(emission_sd,transition_sd){
T = 100
particles = 100
zt = xt = error = rep(NA,T)

# for zt and xt
zt[1] = initModel(1)
for(i in 2:T){
  zt[i] <- transitionmodel(zt[i-1])
  xt[i] <- emmisionmodel(zt[i],emission_sd)
}

initParticles <- initModel(particles)
particleWeights<- rep(1/particles,particles)
estimation <- c()
Particles25 = Particles75 = NA

for(i in 2:T){
  newParticles <- sample(1:particles,particles,prob=particleWeights,replace = T)
  initParticles <- sapply(initParticles[newParticles],transitionmodel)
```

```r
  if(i == 25){
    Particles25 <- c(initParticles)
  }
  else if(i == 75){
    Particles75 <- c(initParticles)
  }


  for(j in 1:particles){
    particleWeights[j] <- 1
  }
  #Normalise
  particleWeights <- particleWeights/sum(particleWeights)

  Ezt <- sum(particleWeights*initParticles)
  error[i] <- abs(zt[i]-Ezt)
  estimation[i] <- sum(particleWeights * initParticles)
}

data = data.frame(zt,xt,estimation,particles = initModel(particles),Particles25,Particles75,initParticl

ggplot(data,aes(x=c(1:T),y=value,color=variable,xlab="timestep"))+
  geom_line(aes(y=data$zt,col='True Zt'))+
  geom_line(aes(y=data$xt,col='True Xt'))+
  geom_line(aes(y=data$estimation,col='Estimation'))+
  geom_point(aes(x=rep(1,T),y=data$particles,col='particle'))+
  geom_point(aes(x=rep(25,T),y=data$Particles25,col='particle at 25'))+
  geom_point(aes(x=rep(75,T),y=data$Particles75,col='particle at 75'))+
  geom_point(aes(x=rep(T,T),y=data$initParticles,col='particle'))+
  ggtitle(paste('At SD = ',emission_sd))
}

emission_sd = 1
ssm(emission_sd ,transition_sd )

emission_sd = 5
ssm(emission_sd ,transition_sd )

emission_sd = 50
ssm(emission_sd ,transition_sd )
# Question 4: SSMs
# Kalman filter implementation.
set.seed(12345)
start_time <- Sys.time()

T<-10000
mu_0<-50
Sigma_0<-10
R<-1
Q<-5

x<-vector(length=T)
z<-vector(length=T)
```

```r
err<-vector(length=T)

for(t in 1:T){
  x[t]<-ifelse(t==1,rnorm(1,mu_0,Sigma_0),x[t-1]+1+rnorm(1,0,R))
  z[t]<-x[t]+rnorm(1,0,Q)
}

mu<-mu_0
Sigma<-Sigma_0*Sigma_0 # KF uses covariances
for(t in 2:T){
  pre_mu<-mu+1
  pre_Sigma<-Sigma+R*R # KF uses covariances
  K<-pre_Sigma/(pre_Sigma+Q*Q) # KF uses covariances
  mu<-pre_mu+K*(z[t]-pre_mu)
  Sigma<-(1-K)*pre_Sigma

  err[t]<-abs(x[t]-mu)

  cat("t: ",t,", x_t: ",x[t],", E[x_t]: ",mu," , error: ",err[t],"\n")
  flush.console()
}

mean(err[2:T])
sd(err[2:T])

end_time <- Sys.time()
end_time - start_time

# Repetition with the particle filter.

set.seed(12345)
start_time <- Sys.time()

T<-10000
n_par<-100
tra_sd<-1
emi_sd<-5
mu_0<-50
Sigma_0<-10

ini_dis<-function(n){
  return (rnorm(n,mu_0,Sigma_0))
}

tra_dis<-function(zt){
  return (rnorm(1,mean=zt+1,sd=tra_sd))
}

emi_dis<-function(zt){
  return (rnorm(1,mean=zt,sd=emi_sd))
}

den_emi_dis<-function(xt,zt){
```

```r
    return (dnorm(xt,mean=zt,sd=emi_sd))
}

z<-vector(length=T)
x<-vector(length=T)

for(t in 1:T){
  z[t]<-ifelse(t==1,ini_dis(1),tra_dis(z[t-1]))
  x[t]<-emi_dis(z[t])
}

err<-vector(length=T)

bel<-ini_dis(n_par)
w<-rep(1/n_par,n_par)
for(t in 2:T){
  com<-sample(1:n_par,n_par,replace=TRUE,prob=w)
  bel<-sapply(bel[com],tra_dis)

  for(i in 1:n_par){
    w[i]<-den_emi_dis(x[t],bel[i])
  }
  w<-w/sum(w)

  Ezt<-sum(w * bel)
  err[t]<-abs(z[t]-Ezt)

  cat("t: ",t,", z_t: ",z[t],", E[z_t]: ",Ezt," , error: ",err[t],"\n")
  flush.console()
}

mean(err[2:T])
sd(err[2:T])

end_time <- Sys.time()
end_time - start_time

# KF works optimally (i.e. it computes the exact belief function in closed-form) since the SSM sampled
# linear-Gaussian. The particle filter on the other hand is approximate. The more particles the closer
# performance to the KF's but at the cost of increasing the running time.
# SSMs
set.seed(12345)
start_time <- Sys.time()

T<-100
mu_0<-50
Sigma_0<-10
R<-1
Q<-5

x<-vector(length=T)
z<-vector(length=T)
err<-vector(length=T)
```

```r
for(t in 1:T){
  x[t]<-ifelse(t==1,rnorm(1,mu_0,Sigma_0),x[t-1]+1+rnorm(1,0,R))
  z[t]<-x[t]+rnorm(1,0,Q)
}

mu<-mu_0
Sigma<-Sigma_0*Sigma_0 # KF uses covariances
for(t in 2:T){
  pre_mu<-mu+1
  pre_Sigma<-Sigma+R*R # KF uses covariances
  K<-pre_Sigma/(pre_Sigma+Q*Q) # KF uses covariances
  mu<-pre_mu+K*(z[t]-pre_mu)
  Sigma<-(1-K)*pre_Sigma

  err[t]<-abs(x[t]-mu)

  cat("t: ",t,", x_t: ",x[t],", E[x_t]: ",mu," , error: ",err[t],"\n")
  flush.console()
}

mean(err[2:T])
sd(err[2:T])

end_time <- Sys.time()
end_time - start_time
```

Advanced Machine Learning Lab4

Omkar Bhutra (omkbh878), Obaid Ur Rehman (obaur539) ,Ruben Jared Muñoz Roquet (rubmu773)

October 16, 2019

Q1. GP Regression

2.1 Implement GP Regression

1)

```r
library("mvtnorm")
```

```
## Warning: package 'mvtnorm' was built under R version 3.5.3
```

```r
library(kernlab)
```

```
## Warning: package 'kernlab' was built under R version 3.5.2
```

```r
mySqKernel <- function(x1,x2,sigf=1,l=3){
  K <- matrix(0,nrow = length(x1),ncol = length(x2))
  for (i in 1:ncol(K)){
    K[,i] <- sigf^2*exp(-0.5*( (x1-x2[i])/l)^2 )
  }
  return(K)
}

posteriorGP <- function(X,y,Xstar,hyperParam,sigmaNoise){
  # K(x,x)
  k <- mySqKernel(X,X,hyperParam[1],hyperParam[2])
  L <- t(chol(k + diag(sigmaNoise^2,length(X))))
  alpha_ <- solve(L,y)
  alpha <- solve(t(L),alpha_)

  # K(x,xstar)
  kstar <- mySqKernel(X,Xstar,hyperParam[1],hyperParam[2])
  meanVector<- t(kstar)%*%alpha
  v <- solve(L,kstar)
  kstarstar <- mySqKernel(Xstar,Xstar,hyperParam[1],hyperParam[2])
  var <- kstarstar - t(v)%*%v
  return(list("mean" = meanVector, "var" = var))
}
```

2)

```r
x <- c(0.4)
y <- c(0.719)
Xstar <- seq(-1,1,length=20)
hyperParam <- c(sigf=1,l=0.3)
noise <- 0.1

getBounds <- function(mean,var){
  lower <- mean - 1.96 * sqrt(diag(var))
  upper <-  mean + 1.96 * sqrt(diag(var))
  return(list(lower=lower,upper=upper))
}
```
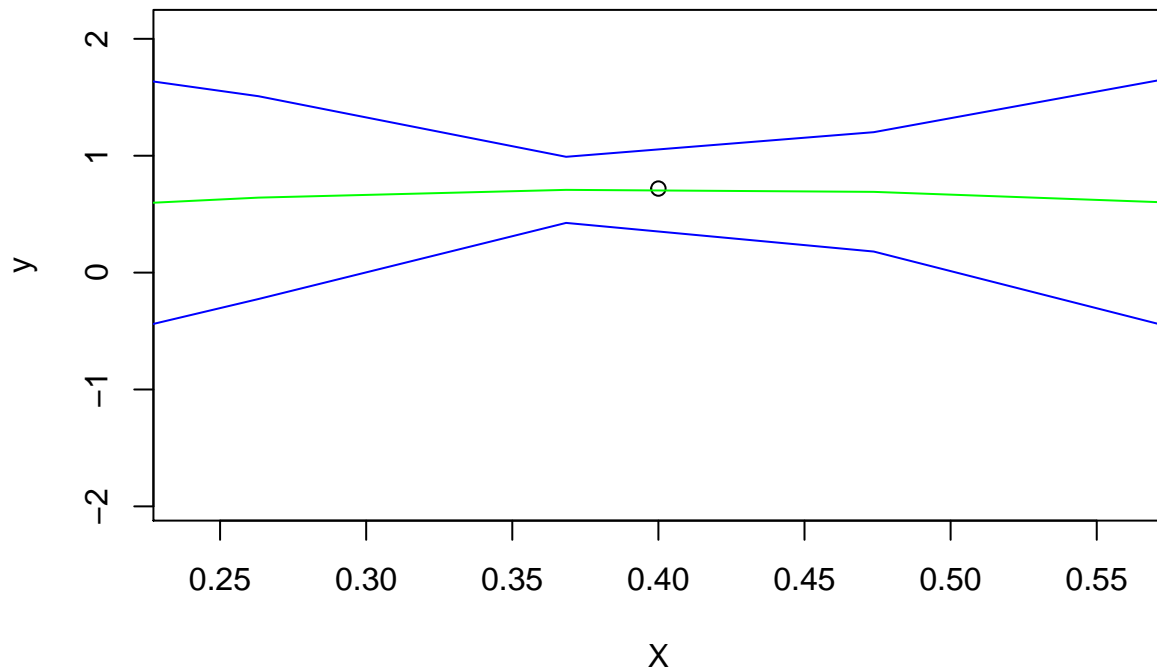
```
plotGP <- function(X,y,Xstar,hyperParam,sigNoise){
  posteriror <- posteriorGP(X,y,Xstar,hyperParam,sigNoise)
  meanVal <- posteriror$mean
  varVal <- posteriror$var
  bounds <- getBounds(meanVal,varVal)

  plot(X,y, type='p' ,ylim=c(min(bounds$lower), max(bounds$upper)))
  lines(Xstar,meanVal,col='green')
  lines(Xstar,bounds$lower,col='blue')
  lines(Xstar,bounds$upper,col='blue')
}
plotGP(x,y,Xstar,hyperParam ,noise)
```
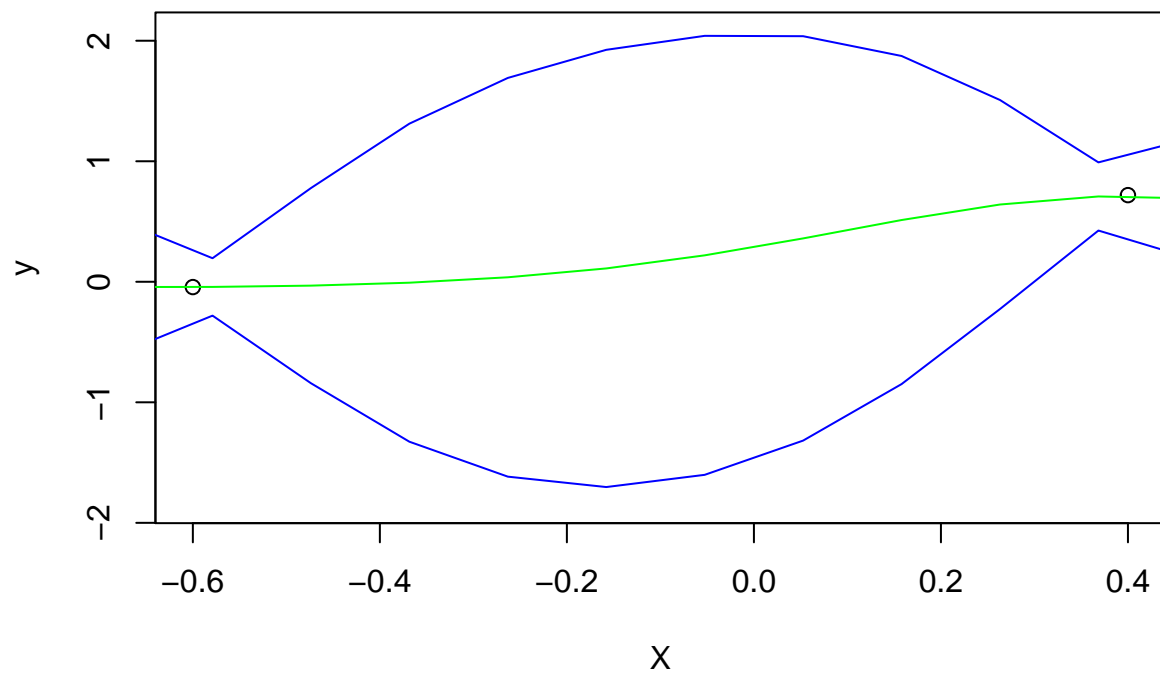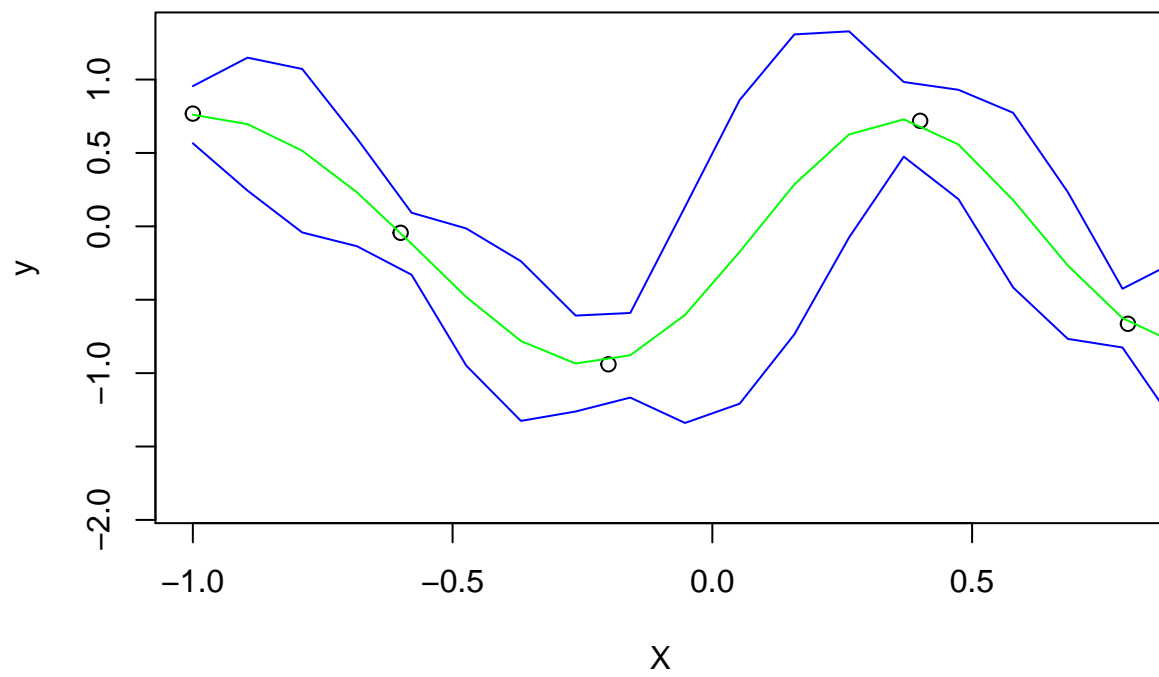

3)

```
# Dont know how to update, following hint
x <- c(0.4,-0.6)
y <- c(0.719,-0.044)
plotGP(x,y,Xstar,hyperParam,noise)
```
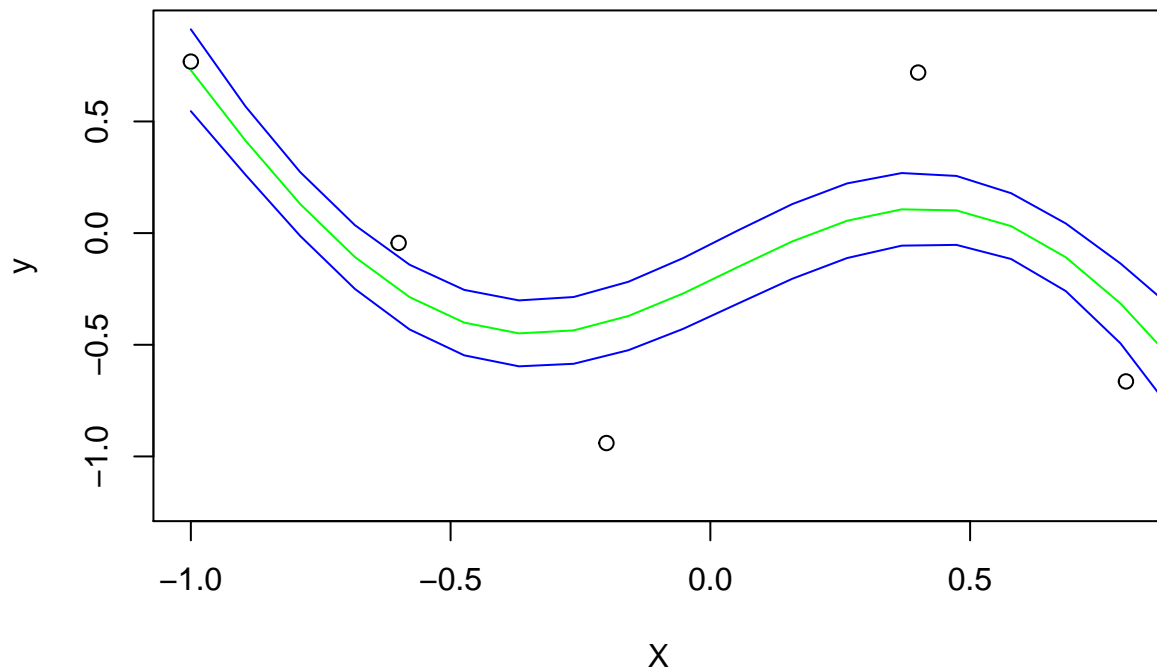
4)

```r
x <- c(-1,-0.6,-0.2,0.4,0.8)
y <- c(0.768,-0.044,-0.940,0.719,-0.664)
plotGP(x,y,Xstar,hyperParam,noise)
```

5)

```r
hyperParam <- c(sigf = 1, l = 1)
plotGP(x,y,Xstar,hyperParam,noise)
```

The probability bands are smmoother now as compared to that obtained previosuly, but the bands does not include all the data points.

2.2) GP Regression with kernlab

1)

```r
data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullinge.c
data$time <- c(1:nrow(data))
data$day <- c(1:365)

dataSampled = data[seq(1, nrow(data), 5), ]

mykern <- function(sigf = 1, l = 1)
{
  rval <- function(x, y = NULL) {
  r = sqrt(crossprod(x-y));
  return(sigf^2 * exp(-0.5 * ( r / l )^2 ))
  }
  class(rval) <- "kernel"
  return(rval)
}
kernFunc <- mykern(sigf = 1, l = 1)
kernFunc(1,2) # Evaluate Kernel at (1,2)
```

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

```
## [1,] 0.6065307
```

```
x <- matrix(c(1,3,4), 3, 1)
xtran <- matrix(c(2,3,4), 3, 1)
K <- kernelMatrix(kernel = kernFunc, x = x, y = xtran)
K
```

```
## An object of class "kernelMatrix"
##           [,1]      [,2]      [,3]
## [1,] 0.6065307 0.1353353 0.0111090
## [2,] 0.6065307 1.0000000 0.6065307
## [3,] 0.1353353 0.6065307 1.0000000
```

2)

```
fit <- lm(temp ~ I(time)+I(time^2),data = dataSampled) # According to equation given
sigmaNoise <- sd(resid(fit))
sigmaNoise
```

```
## [1] 8.176288
```

```
# gp with square exp kernel
sigf <- 30
l <- 0.2
hyperParam <- c(sigf,l)
kern = mykern(sigf,l)
gpFit <- gausspr(dataSampled$time,dataSampled$temp,kernel=kern,var = sigmaNoise^2)
pred <- predict(gpFit,dataSampled$time)
{plot(dataSampled$time, dataSampled$temp, ylim=c(-25,30), main="Time Model")
lines(dataSampled$time,pred, col="blue", lwd = 2)}
```

Time Model
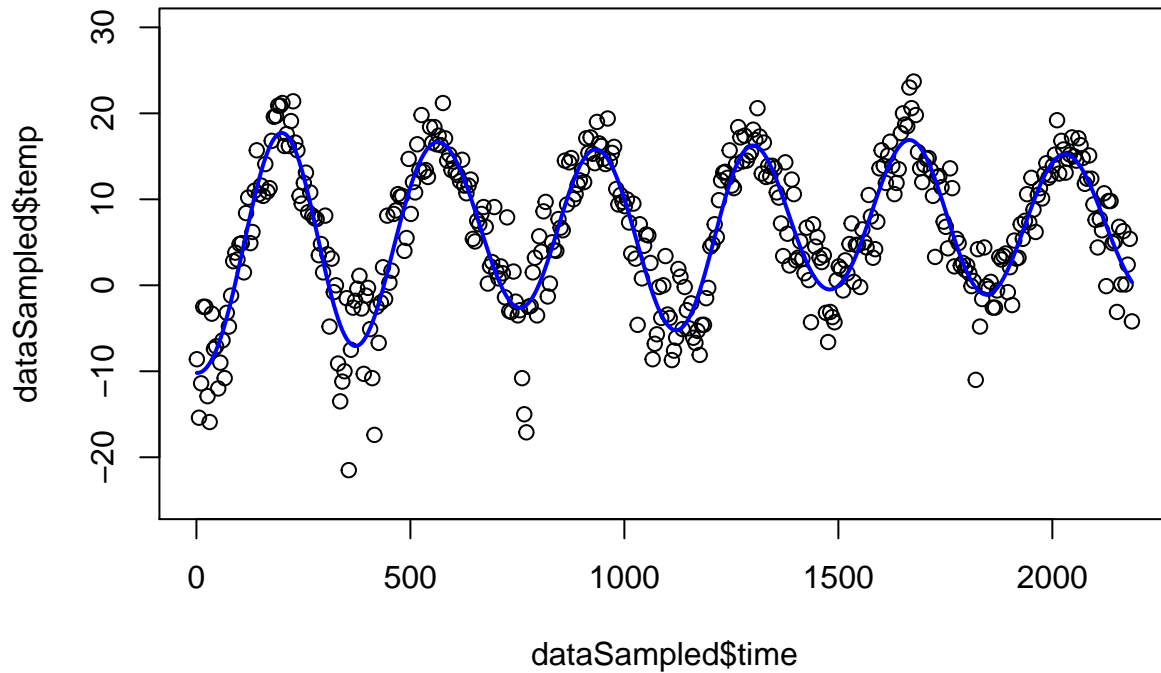
3)

```r
tempS <- scale(dataSampled$temp)
timeS <- scale(dataSampled$time)

x <- seq(1,2190,length=1000)
xS <- scale(x)

post <- posteriorGP(timeS,tempS,xS,hyperParam,sigmaNoise)
meanPost <- post$mean
varPost <- post$var

meanPost <- meanPost *attr(tempS, 'scaled:scale') + attr(tempS, 'scaled:center')

upper <- matrix(NA,length(meanPost),1)
lower <- matrix(nrow = length(meanPost), ncol = 1 ,0)
for (i in 1:length(post$mean)){
  var <- post$var[i,i]
  var <- var * attr(tempS, 'scaled:scale') + attr(tempS, 'scaled:center')
  upper[i] <- meanPost[i] - 1.96*sqrt(var)
  lower[i] <- meanPost[i] + 1.96*sqrt(var)
}
{plot(dataSampled$time, dataSampled$temp, type="p", ylim=c(min(upper), max(lower)), main="Probabilty bar
lines(x, upper, col="blue")
lines(x, lower, col="blue")}
```
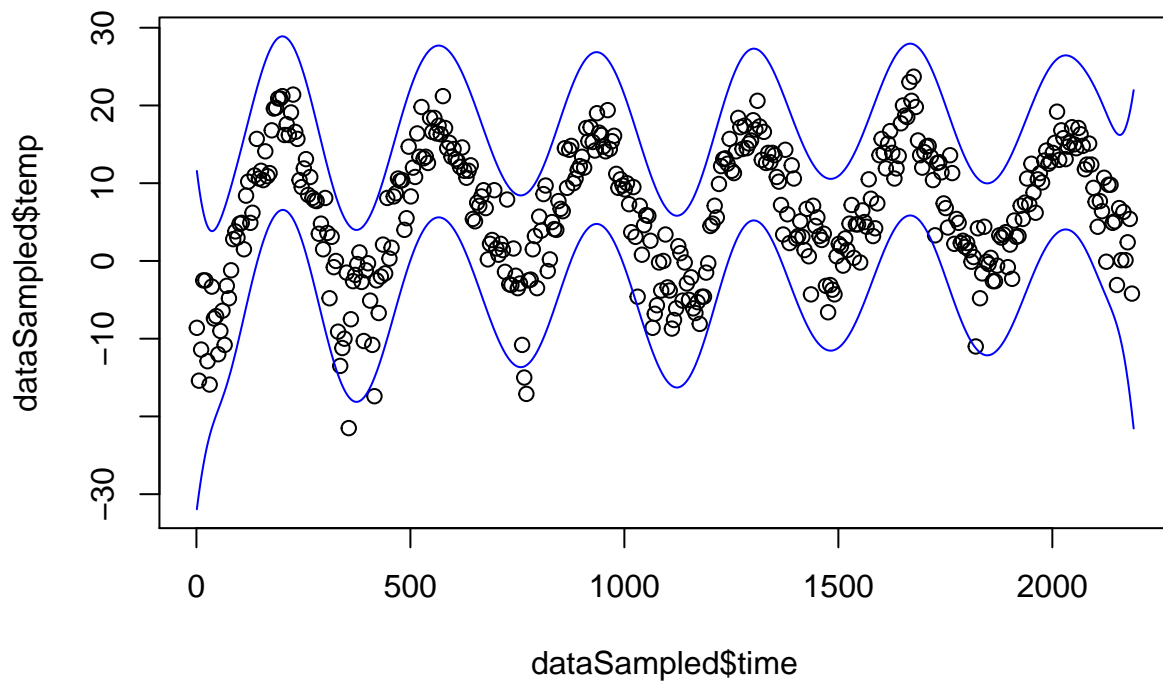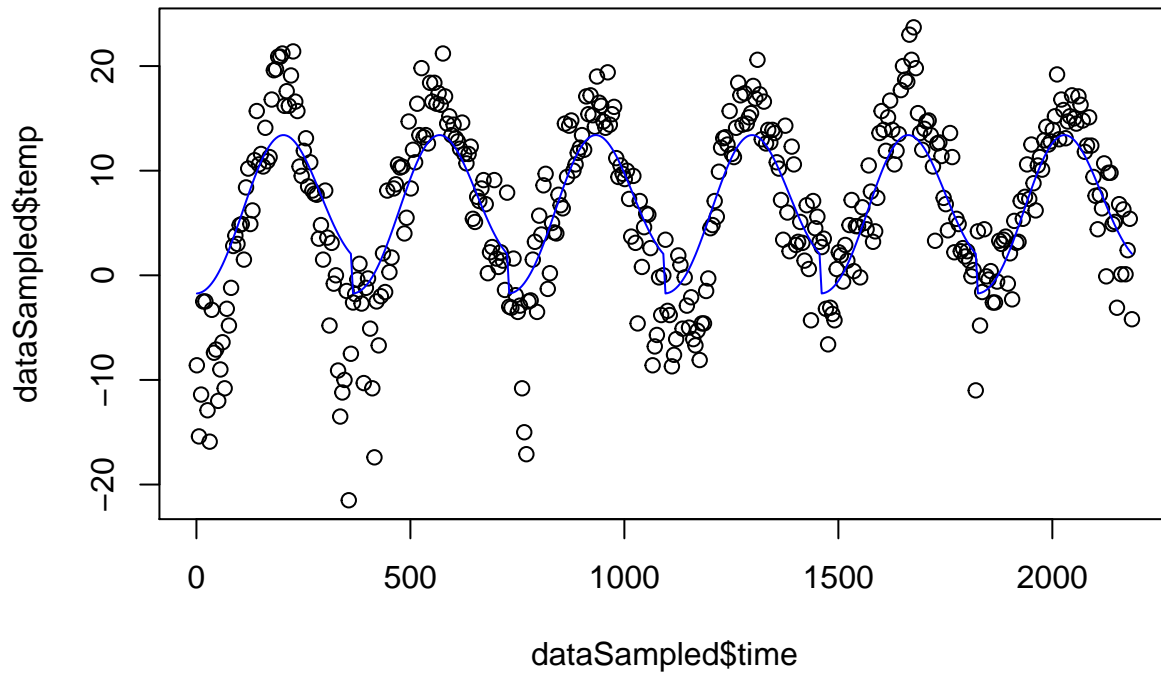
Probabilty band

4)

```r
fit2 <- lm(dataSampled$temp ~ I(dataSampled$day) + I(dataSampled$day^2) )
sigmaNoise = sd(fit2$residuals)

gpfit2 <- gausspr(dataSampled$day, dataSampled$temp, kernel = kernFunc, var = sigmaNoise^2)
postMean2 <- predict(gpfit2,dataSampled$day)
plot(dataSampled$time, dataSampled$temp, main="Day Model")
lines(dataSampled$time,postMean2, col="blue")
```

Day Model

5)

```
sigf <- 20;l1<-1;l2<-10;

fit <- lm(dataSampled$temp ~ dataSampled$time + I(dataSampled$time^2))

timeSd <- sqrt(var(dataSampled$time)); d <- 365/timeSd;
sigNoise = sd(resid(fit))

preodicKern <- function(sigf,l1,l2){
  rval <- function(x, y = NULL) {
  r = sqrt(crossprod(x-y));
    return( sigf^2 * exp(-0.5*( (r)/l2)^2 ) * exp(-2*( (sin(pi*r/d))/l1)^2) )
  }
  class(rval) <- "kernel"
  return(rval)
}

preodKern <- preodicKern(sigf,l1,l2)
gpPrediodic <- gausspr(dataSampled$time, dataSampled$temp, kernel = preodKern, var = sigmaNoise^2)
preodicMeanPred <- predict(gpPrediodic, dataSampled$time)
cat('Time Model Error')


## Time Model Error
```

```
(timeSE = gpFit@error) #time fit
```

[1] 0.1795078

```
cat('Day Model error')
```

Day Model error

```
(datSE = gpfit2@error) # day fit
```

[1] 0.3216524

```
cat('Periodic Model error')
```

Periodic Model error

```
(errorPrTime = gpPrediodic@error) # preodic time fit
```

[1] 0.1929214

```
{plot(dataSampled$time, dataSampled$temp, pch=20, col="lightblue", ylim=c(-25,25),main="Time vs tempratu
lines(dataSampled$time, pred, lwd = 3, type = "l", col="green")
lines(dataSampled$time, postMean2, lwd = 3, type = "l", col="red")
lines(dataSampled$time, preodicMeanPred, lwd = 3, type = "l", col="blue")
legend(x=1550,y=-5, legend=c("Time Model", "Periodic Kernel Model","Day Model"),col=c("green","blue","re
```

Time vs temprature

11
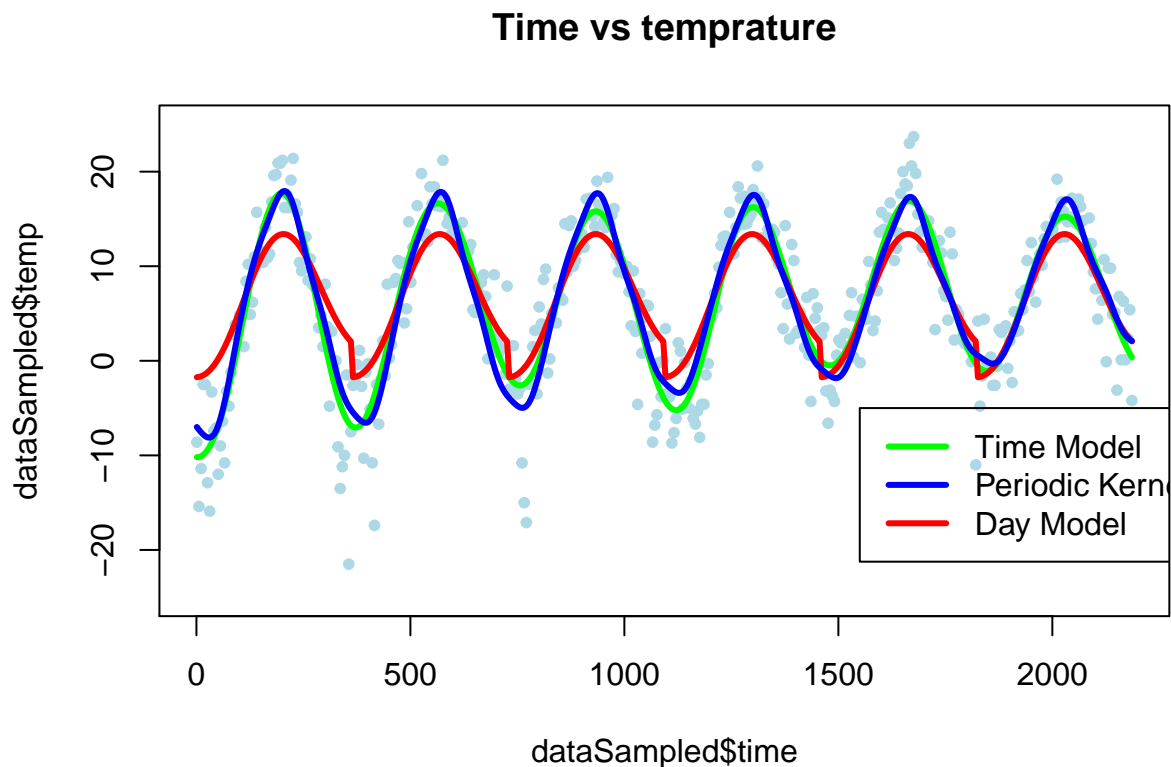
The day model has the highest error and is not smooth. Time model fits the data wl and has the minimum error of all three models.

2.3. GP Classification with kernlab. Download the banknote fraud data

a)You can read about this dataset here. Choose 1000 observations as training data using the following command (i.e., use the vector SelectTraining to subset the training observations):Use the R package kernlab to fit a Gaussian process classification model for fraud on the training data. Use the default kernel and hyperparameters. Start using only the covariates varWave and skewWave in the model. Plot contours of the prediction probabilities over a suitable grid of values for varWave and skewWave. Overlay the training data for fraud = 1 (as blue points) and fraud = 0 (as red points). You can reuse code from the file KernLabDemo.R available on the course website. Compute the confusion matrix for the classifier and its accuracy.

```
library(AtmRay)
```

```
## Warning: package 'AtmRay' was built under R version 3.5.3
```

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.5.3
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:kernlab':
##
##      alpha
```

```
#for meshgrid
# meshgrid <- function(x, y = x) {
#      if (!is.numeric(x) || !is.numeric(y))
#            stop("Arguments 'x' and 'y' must be numeric vectors.")
#
#      x <- c(x); y <- c(y)
#      n <- length(x)
#      m <- length(y)
#
#      X <- matrix(rep(x, each = m),  nrow = m, ncol = n)
#      Y <- matrix(rep(y, times = n), nrow = m, ncol = n)
#
#      return(list(X = X, Y = Y))
# }
data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/
GaussianProcess/Code/banknoteFraud.csv", header=FALSE, sep=",")
names(data) <- c("varWave","skewWave","kurtWave","entropyWave","fraud")
data[,5] <- as.factor(data[,5])
set.seed(111)
SelectTraining <- sample(1:dim(data)[1], size = 1000, replace = FALSE)
train <- data[SelectTraining,]
test <- data[-SelectTraining,]
colnames(data)
```

```
## [1] "varWave"     "skewWave"     "kurtWave"     "entropyWave" "fraud"

#Using automatic sigma estimation or sigest for RBF or laplace kernel
GPfraud_model <- gausspr(fraud ~ varWave + skewWave, data = train)


## Using automatic sigma estimation (sigest) for RBF or laplace kernel

GPfraud_model


## Gaussian Processes object of class "gausspr"
## Problem type: classification
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  1.2043047635594
##
## Number of training instances learned : 1000
## Train error : 0.068

summary(GPfraud_model)


##  Length   Class    Mode
##       1 gausspr      S4

#predict using the test data
GPfraud_pred_train <- predict(GPfraud_model, train[,c("varWave","skewWave")])
#confusion matrix
table(GPfraud_pred_train, train$fraud)


##
## GPfraud_pred_train   0    1
##                  0 512   24
##                  1  44  420

#accuracy
mean(GPfraud_pred_train == train$fraud)


## [1] 0.932

#creating grid points for contour plot
contour_x <- seq(min(data[,"varWave"]),max(data[,"varWave"]),length = 100)
contour_y <- seq(min(data[,"skewWave"]),max(data[,"skewWave"]),length = 100)
grid <- AtmRay::meshgrid(contour_x,contour_y)
grid <- cbind(c(grid$x),c(grid$y))
grid <- data.frame(grid)
names(grid) <- c("varWave","skewWave")
prediction_probability <- predict(GPfraud_model, grid, type = "probabilities")
z <- t(matrix(prediction_probability[,1],100))
contour(contour_x,contour_y, z ,nlevels = 20,
        xlab = "varWave",ylab = "skewWave",
        main = "Probabilities of Fraud")
points(data[data[,5]==1,"varWave"],data[data[,5]==1,"skewWave"],col = "blue")
points(data[data[,5]==0,"varWave"],data[data[,5]==0,"skewWave"],col = "red")
```
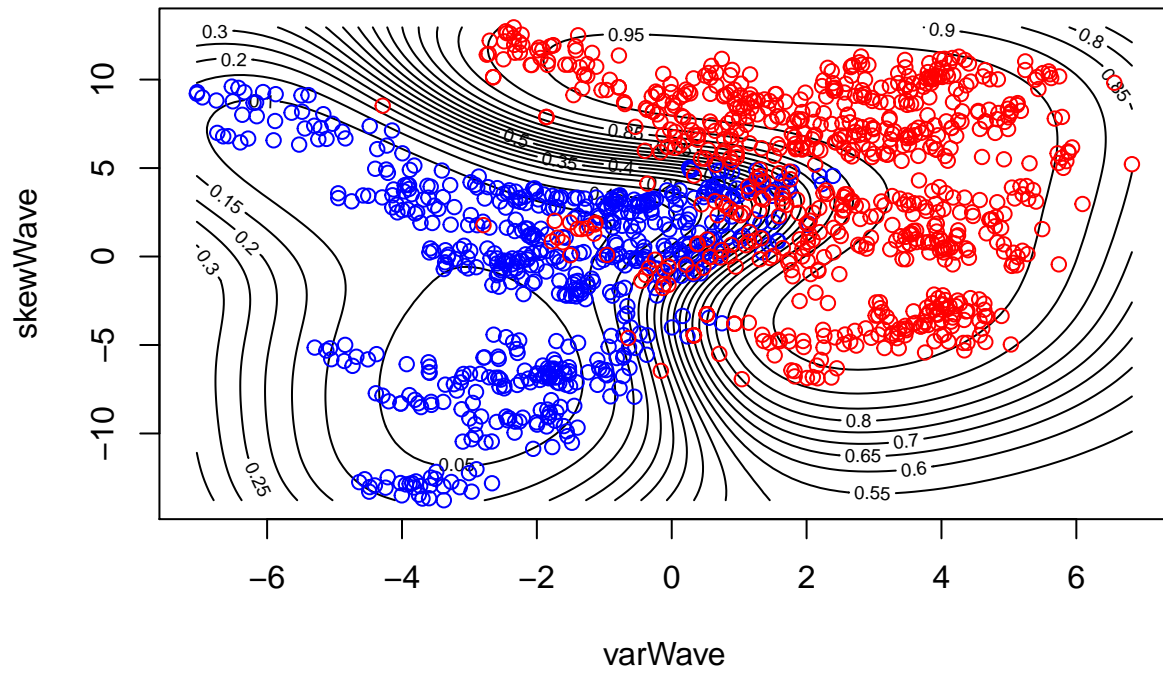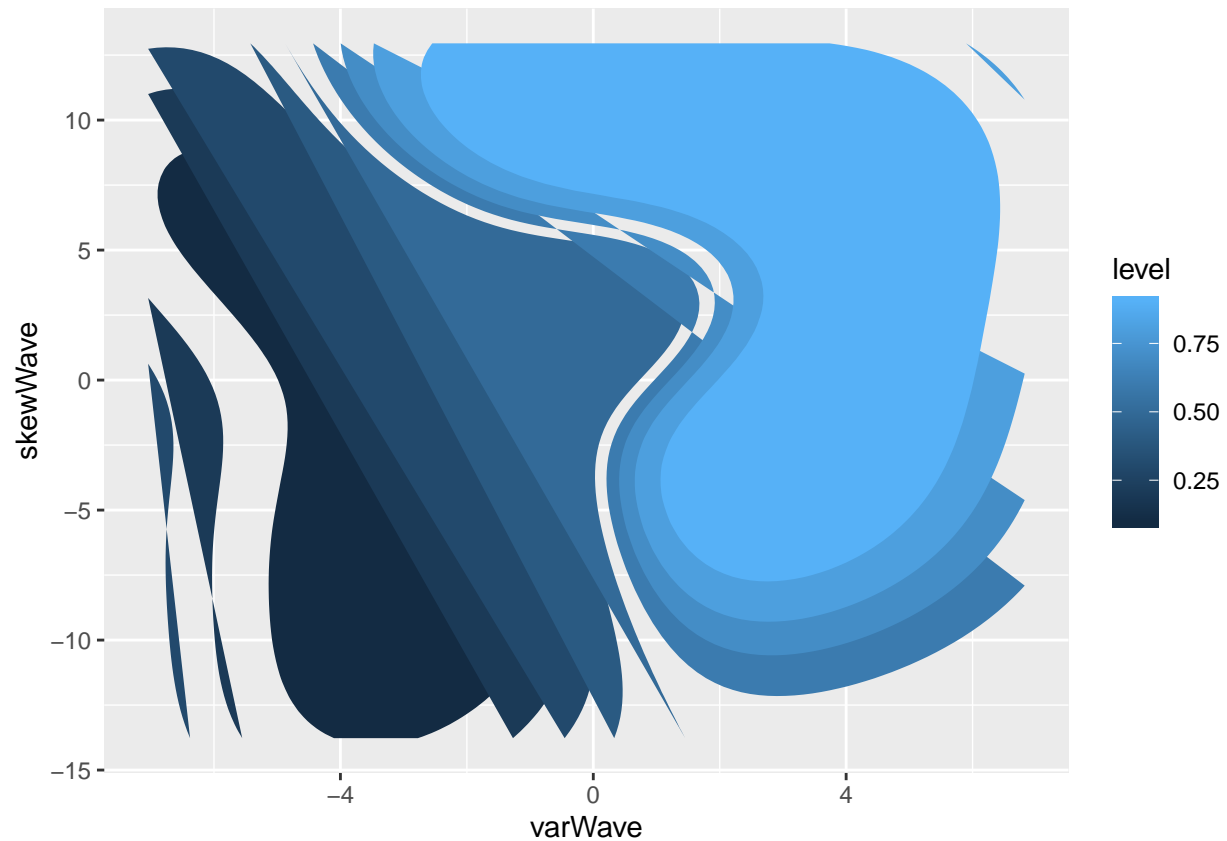
Probabilities of Fraud


```r
ggplot(grid, aes(x = varWave,y = skewWave, z = prediction_probability[,1]))+
stat_contour(geom="polygon",aes(fill=..level..))
```

b)Using the estimated model from (1), make predictions for the test set. Compute the accuracy.

```
# predict on the test set
GPfraud_pred_test <- predict(GPfraud_model, test[,c("varWave","skewWave")])
#confusion matrix and accuracy
table(GPfraud_pred_test, test$fraud)
```

```
##
## GPfraud_pred_test   0   1
##                 0 191   9
##                 1  15 157
```

```
mean(GPfraud_pred_test == test$fraud)
```

```
## [1] 0.9354839
```

c)Train a model using all four covariates. Make predictions on the test set and compare the accuracy to the model with only two covariates.

```
GPfraud_model_allvars <- gausspr(fraud ~ ., data = train)
```

```
## Using automatic sigma estimation (sigest) for RBF or laplace kernel
```

```
GPfraud_model_allvars
```

```
## Gaussian Processes object of class "gausspr"
## Problem type: classification
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.399933221120042
##
## Number of training instances learned : 1000
## Train error : 0.004
```

```
# prediction using test data
GPfraud_pred_test_allvars<- predict(GPfraud_model_allvars,test[,-ncol(test)])
# confusion matrix and accuracy
table(GPfraud_pred_test_allvars, test$fraud)
```

```
##
## GPfraud_pred_test_allvars   0   1
##                         0 205   0
##                         1   1 166
```

```
mean(GPfraud_pred_test_allvars == test$fraud)
```

```
## [1] 0.9973118
```

Question: Explain the difference between the results from ii) and iii). Answer: ii) is about the uncertainty of the function f, which is the MEAN of y iii) is about the uncertainty of individual y values. They are uncertain for two reasons: you don't know f at the test point, and you don't know the error (epsilon) that will hit this individual observation

Question: Discuss the differences in results from using the two length scales. Answer: shorter length scale gives less smooth f. We are overfitting the data. Answer: longer length scale gives more smoothness.

Question: Do you think a GP with a squared exponential kernel is a good model for this data? If not, why? Answer: One would have to experiment with other length scales, or estimate the length scales (see question 3c), but this is not likely to help here. The issue is that the data seems to be have different smoothness for small x than it has for large x (where the function seems much more flat) The solution is probably to have different length scales for different x

Question 3(c)

mention EITHER of the following two approaches: 1. The marginal likelihood can be used to select optimal hyperparameters, and also the noise variance. We can optimize the log marginal likelihood with respect to the hyperparameters. In Gaussian Process Regression the marginal likelihood is availble in closed form (a formula). 2. We can use sampling methods (e.g. MCMC) to sample from the marginal posterior of the hyperparameters We need a prior p(theta) for the hyperparameter and then Bayes rule gives the marginal posterior p(theta | data) propto p(data | theta)*p(theta) where p(data | theta) is the marginal likelihood (f has been integrated out).

If the noise variance is unknown, we can treat like any of the kernel hyperparameters and infer the noise variance jointly with the length scale and the prior variance sigma_f

```r
knitr::opts_chunk$set(echo = TRUE)
library("mvtnorm")
library(kernlab)

mySqKernel <- function(x1,x2,sigf=1,l=3){
  K <- matrix(0,nrow = length(x1),ncol = length(x2))
  for (i in 1:ncol(K)){
    K[,i] <- sigf^2*exp(-0.5*( (x1-x2[i])/l)^2 )
  }
  return(K)
}

posteriorGP <- function(X,y,Xstar,hyperParam,sigmaNoise){
  # K(x,x)
  k <- mySqKernel(X,X,hyperParam[1],hyperParam[2])
  L <- t(chol(k + diag(sigmaNoise^2,length(X))))
  alpha_ <- solve(L,y)
  alpha <- solve(t(L),alpha_)

  # K(x,xstar)
  kstar <- mySqKernel(X,Xstar,hyperParam[1],hyperParam[2])
  meanVector<- t(kstar)%*%alpha
  v <- solve(L,kstar)
  kstarstar <- mySqKernel(Xstar,Xstar,hyperParam[1],hyperParam[2])
  var <- kstarstar - t(v)%*%v
  return(list("mean" = meanVector, "var" = var))
}

x <- c(0.4)
y <- c(0.719)
Xstar <- seq(-1,1,length=20)
hyperParam <- c(sigf=1,l=0.3)
noise <- 0.1

getBounds <- function(mean,var){
  lower <- mean - 1.96 * sqrt(diag(var))
  upper <-  mean + 1.96 * sqrt(diag(var))
  return(list(lower=lower,upper=upper))
}
plotGP <- function(X,y,Xstar,hyperParam,sigNoise){
  posteriror <- posteriorGP(X,y,Xstar,hyperParam,sigNoise)
  meanVal <- posteriror$mean
  varVal <- posteriror$var
  bounds <- getBounds(meanVal,varVal)

  plot(X,y, type='p' ,ylim=c(min(bounds$lower), max(bounds$upper)))
  lines(Xstar,meanVal,col='green')
  lines(Xstar,bounds$lower,col='blue')
  lines(Xstar,bounds$upper,col='blue')
}
plotGP(x,y,Xstar,hyperParam ,noise)
# Dont know how to update, following hint
x <- c(0.4,-0.6)
```

```r
y <- c(0.719,-0.044)
plotGP(x,y,Xstar,hyperParam,noise)
x <- c(-1,-0.6,-0.2,0.4,0.8)
y <- c(0.768,-0.044,-0.940,0.719,-0.664)
plotGP(x,y,Xstar,hyperParam,noise)
hyperParam <- c(sigf = 1, l = 1)
plotGP(x,y,Xstar,hyperParam,noise)
data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullinge.
data$time <- c(1:nrow(data))
data$day <- c(1:365)

dataSampled = data[seq(1, nrow(data), 5), ]

mykern <- function(sigf = 1, l = 1)
{
  rval <- function(x, y = NULL) {
  r = sqrt(crossprod(x-y));
  return(sigf^2 * exp(-0.5 * ( r / l )^2 ))
  }
  class(rval) <- "kernel"
  return(rval)
}
kernFunc <- mykern(sigf = 1, l = 1)
kernFunc(1,2) # Evaluate Kernel at (1,2)

x <- matrix(c(1,3,4), 3, 1)
xtran <- matrix(c(2,3,4), 3, 1)
K <- kernelMatrix(kernel = kernFunc, x = x, y = xtran)
K

fit <- lm(temp ~ I(time)+I(time^2),data = dataSampled) # According to equation given
sigmaNoise <- sd(resid(fit))
sigmaNoise

# gp with square exp kernel
sigf <- 30
l <- 0.2
hyperParam <- c(sigf,l)
kern = mykern(sigf,l)
gpFit <- gausspr(dataSampled$time,dataSampled$temp,kernel=kern,var = sigmaNoise^2)
pred <- predict(gpFit,dataSampled$time)
{plot(dataSampled$time, dataSampled$temp, ylim=c(-25,30), main="Time Model")
lines(dataSampled$time,pred, col="blue", lwd = 2)}
tempS <- scale(dataSampled$temp)
timeS <- scale(dataSampled$time)

x <- seq(1,2190,length=1000)
xS <- scale(x)

post <- posteriorGP(timeS,tempS,xS,hyperParam,sigmaNoise)
meanPost <- post$mean
varPost <- post$var
```

```r
meanPost <- meanPost *attr(tempS, 'scaled:scale') + attr(tempS, 'scaled:center')

upper <- matrix(NA,length(meanPost),1)
lower <- matrix(nrow = length(meanPost), ncol = 1 ,0)
for (i in 1:length(post$mean)){
  var <- post$var[i,i]
  var <- var * attr(tempS, 'scaled:scale') + attr(tempS, 'scaled:center')
  upper[i] <- meanPost[i] - 1.96*sqrt(var)
  lower[i] <- meanPost[i] + 1.96*sqrt(var)
}
{plot(dataSampled$time, dataSampled$temp, type="p", ylim=c(min(upper), max(lower)), main="Probabilty ba
lines(x, upper, col="blue")
lines(x, lower, col="blue")}
fit2 <- lm(dataSampled$temp ~ I(dataSampled$day) + I(dataSampled$day^2) )
sigmaNoise = sd(fit2$residuals)

gpfit2 <- gausspr(dataSampled$day, dataSampled$temp, kernel = kernFunc, var = sigmaNoise^2)
postMean2 <- predict(gpfit2,dataSampled$day)
plot(dataSampled$time, dataSampled$temp, main="Day Model")
lines(dataSampled$time,postMean2, col="blue")
sigf <- 20;l1<-1;l2<-10;

fit <- lm(dataSampled$temp ~ dataSampled$time + I(dataSampled$time^2))

timeSd <- sqrt(var(dataSampled$time)); d <- 365/timeSd;
sigNoise = sd(resid(fit))

preodicKern <- function(sigf,l1,l2){
  rval <- function(x, y = NULL) {
  r = sqrt(crossprod(x-y));
    return( sigf^2 * exp(-0.5*( (r)/l2)^2 ) * exp(-2*( (sin(pi*r/d))/l1)^2) )
  }
  class(rval) <- "kernel"
  return(rval)
}

preodKern <- preodicKern(sigf,l1,l2)
gpPrediodic <- gausspr(dataSampled$time, dataSampled$temp, kernel = preodKern, var = sigmaNoise^2)
preodicMeanPred <- predict(gpPrediodic, dataSampled$time)
cat('Time Model Error')
(timeSE = gpFit@error) #time fit
cat('Day Model error')
(datSE = gpfit2@error) # day fit

cat('Periodic Model error')
(errorPrTime = gpPrediodic@error) # preodic time fit

{plot(dataSampled$time, dataSampled$temp, pch=20, col="lightblue", ylim=c(-25,25),main="Time vs tempratu
lines(dataSampled$time, pred, lwd = 3, type = "l", col="green")
lines(dataSampled$time, postMean2, lwd = 3, type = "l", col="red")
lines(dataSampled$time, preodicMeanPred, lwd = 3, type = "l", col="blue")
legend(x=1550,y=-5, legend=c("Time Model", "Periodic Kernel Model","Day Model"),col=c("green","blue","r
library(AtmRay)
```

```r
library(ggplot2)
#for meshgrid
# meshgrid <- function(x, y = x) {
#     if (!is.numeric(x) || !is.numeric(y))
#         stop("Arguments 'x' and 'y' must be numeric vectors.")
#
#     x <- c(x); y <- c(y)
#     n <- length(x)
#     m <- length(y)
#
#     X <- matrix(rep(x, each = m),  nrow = m, ncol = n)
#     Y <- matrix(rep(y, times = n), nrow = m, ncol = n)
#
#     return(list(X = X, Y = Y))
# }
data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/
GaussianProcess/Code/banknoteFraud.csv", header=FALSE, sep=",")
names(data) <- c("varWave","skewWave","kurtWave","entropyWave","fraud")
data[,5] <- as.factor(data[,5])
set.seed(111)
SelectTraining <- sample(1:dim(data)[1], size = 1000, replace = FALSE)
train <- data[SelectTraining,]
test <- data[-SelectTraining,]
colnames(data)
#Using automatic sigma estimation or sigest for RBF or laplace kernel
GPfraud_model <- gausspr(fraud ~ varWave + skewWave, data = train)
GPfraud_model
summary(GPfraud_model)
#predict using the test data
GPfraud_pred_train <- predict(GPfraud_model, train[,c("varWave","skewWave")])
#confusion matrix
table(GPfraud_pred_train, train$fraud)
#accuracy
mean(GPfraud_pred_train == train$fraud)
#creating grid points for contour plot
contour_x <- seq(min(data[,"varWave"]),max(data[,"varWave"]),length = 100)
contour_y <- seq(min(data[,"skewWave"]),max(data[,"skewWave"]),length = 100)
grid <- AtmRay::meshgrid(contour_x,contour_y)
grid <- cbind(c(grid$x),c(grid$y))
grid <- data.frame(grid)
names(grid) <- c("varWave","skewWave")
prediction_probability <- predict(GPfraud_model, grid, type = "probabilities")
z <- t(matrix(prediction_probability[,1],100))
contour(contour_x,contour_y, z ,nlevels = 20,
        xlab = "varWave",ylab = "skewWave",
        main = "Probabilities of Fraud")
points(data[data[,5]==1,"varWave"],data[data[,5]==1,"skewWave"],col = "blue")
points(data[data[,5]==0,"varWave"],data[data[,5]==0,"skewWave"],col = "red")


ggplot(grid, aes(x = varWave,y = skewWave, z = prediction_probability[,1]))+
stat_contour(geom="polygon",aes(fill=..level..))
# predict on the test set
```

```r
GPfraud_pred_test <- predict(GPfraud_model, test[,c("varWave","skewWave")])
#confusion matrix and accuracy
table(GPfraud_pred_test, test$fraud)
mean(GPfraud_pred_test == test$fraud)
GPfraud_model_allvars <- gausspr(fraud ~ ., data = train)
GPfraud_model_allvars
# prediction using test data
GPfraud_pred_test_allvars<- predict(GPfraud_model_allvars,test[,-ncol(test)])
# confusion matrix and accuracy
table(GPfraud_pred_test_allvars, test$fraud)
mean(GPfraud_pred_test_allvars == test$fraud)
### Question 3(a)
# Change to your path
library(kernlab)
library(mvtnorm)

# Simulating from the prior for l = 0.2
kernel02 <- mykern(sigf = 1, l = 0.2) # This constructs the covariance function
xGrid = seq(-1,1,by=0.1)
K = kernelMatrix(kernel = kernel02, xGrid, xGrid)

colors = list("black","red","blue","green","purple")
f = rmvnorm(n = 1, mean = rep(0,length(xGrid)), sigma = K)
plot(xGrid,f, type = "l", ylim = c(-3,3), col = colors[[1]])
for (i in 1:4){
  f = rmvnorm(n = 1, mean = rep(0,length(xGrid)), sigma = K)
  lines(xGrid,f, col = colors[[i+1]])
}

# Simulating from the prior for l = 1
kernel1 <- mykern(sigf = 1, l = 1) # This constructs the covariance function
xGrid = seq(-1,1,by=0.1)
K = kernelMatrix(kernel = kernel1, xGrid, xGrid)

colors = list("black","red","blue","green","purple")
f = rmvnorm(n = 1, mean = rep(0,length(xGrid)), sigma = K)
plot(xGrid,f, type = "l", ylim = c(-3,3), col = colors[[1]])
for (i in 1:4){
  f = rmvnorm(n = 1, mean = rep(0,length(xGrid)), sigma = K)
  lines(xGrid,f, col = colors[[i+1]])
}


# Computing the correlation functions

# l = 0.2
kernel02(0,0.1) # Note: here correlation=covariance since sigf = 1
kernel02(0,0.5)

# l = 1
kernel1(0,0.1) # Note: here correlation=covariance since sigf = 1
kernel1(0,0.5)
```

```r
# The correlation between the function at x = 0 and x=0.1 is much higher than
# between x = 0 and x=0.5, since the latter points are more distant.
# Thus, the correlation decays with distance in x-space. This decay is much more
# rapid when l = 0.2 than when l = 1. This is also visible in the simulations
# where realized functions are much more smooth when l = 1.


### Question 3(b)

load("GPdata.RData")

### l = 0.2

sigmaNoise = 0.2

# Set up the kernel function
kernelFunc <- mykern(sigf = 1, l = 0.2)

# Plot the data and the true
plot(x, y, main = "", cex = 0.5)

GPfit <- gausspr(x, y, kernel = kernelFunc, var = sigmaNoise^2)
# Alternative: GPfit <- gausspr(y ~ x, kernel = mykern, kpar = list(sigf = 1, l = 0.2), var = sigmaNois
xs = seq(min(x),max(x), length.out = 100)
meanPred <- predict(GPfit, data.frame(x = xs)) # Predicting the training data. To plot the fit.
lines(xs, meanPred, col="blue", lwd = 2)

# Compute the covariance matrix Cov(f)
n <- length(x)
Kss <- kernelMatrix(kernel = kernelFunc, x = xs, y = xs)
Kxx <- kernelMatrix(kernel = kernelFunc, x = x, y = x)
Kxs <- kernelMatrix(kernel = kernelFunc, x = x, y = xs)
Covf = Kss-t(Kxs)%*%solve(Kxx + sigmaNoise^2*diag(n), Kxs)

# Probability intervals for f
lines(xs, meanPred - 1.96*sqrt(diag(Covf)), col = "red")
lines(xs, meanPred + 1.96*sqrt(diag(Covf)), col = "red")

# Prediction intervals for y
lines(xs, meanPred - 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "purple")
lines(xs, meanPred + 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "purple")


legend("topright", inset = 0.02, legend = c("data","post mean","95% intervals for f", "95% predictive i
       col = c("black", "blue", "red", "purple"),
       pch = c('o',NA,NA,NA), lty = c(NA,1,1,1), lwd = 2, cex = 0.55)


### l = 1

sigmaNoise = 0.2
```

```r
# Set up the kernel function
kernelFunc <- mykern(sigf = 1, l = 1)

# Plot the data and the true
plot(x,y, main = "", cex = 0.5)
#lines(xGrid,fVals, type = "l", col = "black", lwd = 3) # true mean

GPfit <- gausspr(x, y, kernel = kernelFunc, var = sigmaNoise^2)
xs = seq(min(x), max(x), length.out = 100)
meanPred <- predict(GPfit, data.frame(x = xs)) # Predicting the training data. To plot the fit.
lines(xs, meanPred, col="blue", lwd = 2)

# Compute the covariance matrix Cov(f)
n <- length(x)
Kss <- kernelMatrix(kernel = kernelFunc, x = xs, y = xs)
Kxx <- kernelMatrix(kernel = kernelFunc, x = x, y = x)
Kxs <- kernelMatrix(kernel = kernelFunc, x = x, y = xs)
Covf = Kss-t(Kxs)%*%solve(Kxx + sigmaNoise^2*diag(n), Kxs)

# Probability intervals for f
lines(xs, meanPred - 1.96*sqrt(diag(Covf)), col = "red")
lines(xs, meanPred + 1.96*sqrt(diag(Covf)), col = "red")

# Prediction intervals for y
lines(xs, meanPred - 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "purple")
lines(xs, meanPred + 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "purple")


legend("topright", inset = 0.02, legend = c("data","post mean","95% intervals for f", "95% predictive i
       col = c("black", "blue", "red", "purple"),
       pch = c('o',NA,NA,NA), lty = c(NA,1,1,1), lwd = 2, cex = 0.55)
```