732A96/TDDE15 ADVANCED MACHINE LEARNING

EXAM 2025-10-28

Teacher

Jose M. Peña. He will visit the rooms for questions.

Grades

- For 732A96 (A-E means pass):
  - A=19-20 points
  - B=17-18 points
  - C=14-16 points
  - D=12-13 points
  - E=10-11 points
  - F=0-9 points
- For TDDE15 (3-5 means pass):
  - 5=18-20 points
  - 4=14-17 points
  - 3=10-13 points
  - U=0-9 points

In each question, full points requires clear and well motivated answers and commented code.

Instructions

- This is an individual exam. No help from others is allowed. No communication with others is allowed. Answers to the exam questions may be sent to Urkund.
- This is an anonymous exam. Do not write your name on it.
- The answers to the exam should be submitted in a single PDF file. You can make a PDF from LibreOffice (similar to Microsoft Word). You can also use Markdown from RStudio (no support is provided though). Include important code needed to grade the exam (inline or at the end of the PDF file).

Allowed help

Everything on the course web page. Your individual and group solutions to the labs. This help is available on the corresponding directories of the exam system.

1. Probabilistic Graphical Models (4 p)

In this exercise, no implementation is required. You are given a DAG with edges $\{A \rightarrow B, B \rightarrow C, A \rightarrow D, D \rightarrow C\}$. List all the independencies represented by the DAG. List the adjacencies (also called skeleton) and unshielded collides in the DAG. Give two other DAGs that are Markov equivalent to the given DAG. Produce the essential graph (also called complete partially DAG or CPDAG for short) of the given DAG.

2. Hidden Markov Models (6 p)

Implement the Viterbi algorithm as described in the course slides. Use the template provided below. Consider the same scenario as in the lab (i.e., the robot walking in the ring) and run your implementation with the observations $(3, 8, 9)$. Use the HMM package to confirm that your implementation works correctly.

```
x <- c(3,8,9)

w <- matrix(NA,length(States),length(x)) # omega in the slides.
v <- matrix(NA,length(States),length(x)) # phi in the slides.

w[,1] <- # Your code here.

for (t in 1:(length(x)-1)) {
  for (i in 1:length(States)) {
    w[i,t+1] <- # Your code here.
    v[i,t+1] <- # Your code here.
  }
}

z <- vector("integer",length(x))

# Your code here.

z
```

3. Reinforcement Learning (4 p)

Q-learning is an off-policy algorithm, because it acts following an ϵ-greedy policy while it uses the maximum q-value over the next action (i.e., greedy policy) in the updating rule. Expected SARSA is an on-policy modification of Q-learning: It chooses the next action following an ϵ-greedy policy, and uses the expected q-value over the next action in the updating rule (i.e., the average q-value over actions weighted by how likely each action will be the next action when acting following an ϵ-greedy policy).

You are asked to modify your lab solution to implement expected SARSA. Run it for 10000 episodes in the environment A with the same parameter values as in the lab. Plot the moving average of the rewards obtained in those 10000 episodes. Compare the results with those from Q-learning. Comment the results.

4. Gaussian Processes (6 p)

In the course, we have seen the equations for the posterior mean and covariance of a GP for regression. These equations can be implemented directly, or indirectly as in Algorithm 2.1. When implemented directly, it is sometimes more convenient to implement an approximation to them. For instance, Nyström approximation consists in selecting some points Z of the dataset at random (so-called inducing points), and then approximate $K(X,X)$ as follows:

$$K(X,X) \approx K(X,Z)K(Z,Z)^{-1}K(Z,X).$$

This intuitively means that the covariance between two points is approximated as the covariance between the projections of the points onto the space spanned by the inducing points.

In the original equations for the posterior mean and covariance of a GP for regression, $K(X,X)$ appears in the term

$$[K(X,X)+\sigma_n^2 I]^{-1}$$

which, under Nyström approximation, should be replaced by

$$[K(X,Z)K(Z,Z)^{-1}K(Z,X)+\sigma_n^2 I]^{-1}.$$

This matrix inversion can efficiently be computed with the help of Woodbury matrix identity (a.k.a. matrix inversion lemma):

$$(A+UCV)^{-1} = A^{-1} - A^{-1}U[C^{-1}+VA^{-1}U]^{-1}VA^{-1}$$

if we simply let $A=\sigma_n^2 I$, $U=K(X,Z)$, $C=K(Z,Z)^{-1}$ and $V=K(Z,X)$. Note that $A^{-1}=I/\sigma_n^2$.

Your task is to implement the equations for the posterior mean and covariance of a GP under Nyström approximation. Use 100 randomly sampled data points as inducing points. When calling the function `solve`, use the parameter `tol = 0` to avoid numerical problems. Run your code on the Tullinge dataset to predict temperature from time, and compare the results with those you obtained in the lab. Use the template below. Use the local copy of the dataset in the help folder. Use the same setting as in the lab (every 5th observation, squared exponential kernel, $\sigma_F = 20$ and $\ell = 100$). Finally, answer the following question: When may Nyström approximation be particularly suitable?

```
SEKernel <- function(x1,x2){
  n1 <- length(x1)
  n2 <- length(x2)
  K <- matrix(NA,n1,n2)
  for (i in 1:n2){
    K[,i] <- (sigmaF^2)*exp(-0.5*( (x1-x2[i])/l)^2 )
  }
  return(K)
}

# Your code here.

Kxs <- SEKernel(x,xs)
Kss <- SEKernel(xs,xs)
Kxz <- SEKernel(x,z)
Kzz <- SEKernel(z,z)

# Your code here.
```