732A96/TDDE15 Advanced Machine Learning

Graphical Models

Jose M. Peña
IDA, Linköping University, Sweden

Lecture 1: Bayesian and Markov Networks

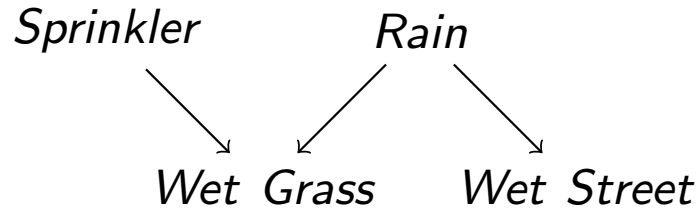
Contents

Literature

- Main source
  - Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006. Chapter 8.
- Additional source
  - Koski, T. J. T. and Noble, J. M. A Review of Bayesian Networks and Structure Learning. *Mathematica Applicanda* 40, 51-103, 2012.

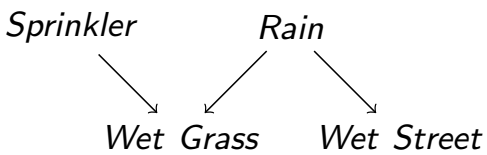
Causal Structures

- Assume that we want to represent the causal relations between a set of random variables, e.g. the variables may represent the state of the components of a system.
- A natural and intuitive representation consists of a **graph** where the nodes are the random variables, and the edges are the causal relations between the variables. We call such a graph a **causal structure**.



- **Exercise**. Produce a causal structure for the domain *Temperature*, *Ice cream sales* and *Soda sales*.
- **Exercise**. Produce a causal structure for Boyle's law, which relates the pressure and volume of a gas as $Pressure \cdot Volume = constant$ if the temperature and amount of gas remain unchanged within a closed system.

Bayesian Networks: Definition

DAG	Parameter values for the conditional probability distributions
	$q(s) = (0.3, 0.7) = (\theta_{s_0}, \theta_{s_1})$ $q(r) = (0.5, 0.5) = (\theta_{r_0}, \theta_{r_1})$ $q(wg\|r_0, s_0) = (0.1, 0.9) = (\theta_{wg_0\|r_0,s_0}, \theta_{wg_1\|r_0,s_0})$ $q(wg\|r_0, s_1) = (0.7, 0.3) = (\theta_{wg_0\|r_0,s_1}, \theta_{wg_1\|r_0,s_1})$ $q(wg\|r_1, s_0) = (0.8, 0.2) = (\theta_{wg_0\|r_1,s_0}, \theta_{wg_1\|r_1,s_0})$ $q(wg\|r_1, s_1) = (0.9, 0.1) = (\theta_{wg_0\|r_1,s_1}, \theta_{wg_1\|r_1,s_1})$ $q(ws\|r_0) = (0.1, 0.9) = (\theta_{ws_0\|r_0}, \theta_{ws_1\|r_0})$ $q(ws\|r_1) = (0.7, 0.3) = (\theta_{ws_0\|r_1}, \theta_{ws_1\|r_1})$ $p(s, r, wg, ws) = q(s)q(r)q(wg\|s, r)q(ws\|r)$

- A **Bayesian network (BN)** over a finite set of **discrete** random variables $X = X_{1:n} = \{X_1, \ldots, X_n\}$ consists of
  - a directed acyclic graph (DAG) G whose nodes are the elements in X, and
  - parameter values θ specifying probability distributions $q(x_i|pa_i)$, where Pa_i are the parents of X_i in G, i.e. the nodes with an edge into X_i.
- The BN represents a causal model of the system.
- And also a probabilistic model of the system as $p(x) = \prod_i q(x_i|pa_i)$.

Bayesian Networks: Definition

- We now show that $p(x) = \prod_i q(x_i|pa_i)$ is a probability distribution.
- Clearly, $0 \leq \prod_i q(x_i|pa_i) \leq 1$.
- Assume without loss of generality that $Pa_i \subseteq X_{1:i-1}$ for all i. Then

$$\sum_x \prod_i q(x_i|pa_i) = \sum_{x_1} [q(x_1) \dots \sum_{x_{n-1}} [q(x_{n-1}|pa_{n-1}) \sum_{x_n} q(x_n|pa_n)] \dots] = 1$$

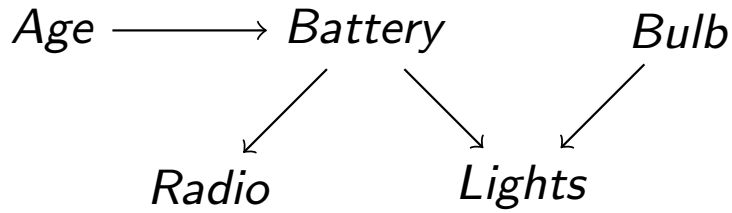
- Moreover, $p(x_j|pa_j) = q(x_j|pa_j)$. To see it, note that

$$p(x_j|pa_j) = \frac{p(x_j, pa_j)}{p(pa_j)} = \frac{\sum_{x \setminus \{x_j, pa_j\}} \prod_i q(x_i|pa_i)}{\sum_{x \setminus pa_j} \prod_i q(x_i|pa_i)}$$

$$= \frac{\sum_{x_{1:j} \setminus \{x_j, pa_j\}} \prod_{i \leq j} q(x_i|pa_i)}{\sum_{x_{1:j} \setminus pa_j} \prod_{i \leq j} q(x_i|pa_i)} = q(x_j|pa_j)$$

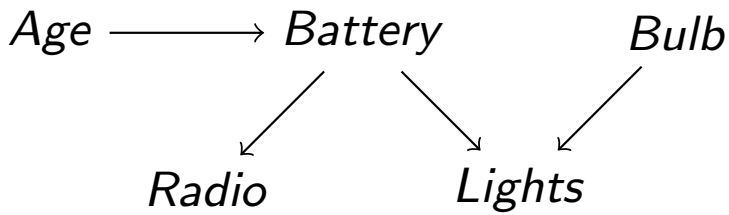
Bayesian Networks: Separation

- We now show that many of the independencies in p can be read off G without numerical calculations. Consider the following DAG.



- **Chain**: $Age \to Battery \to Radio$
  - $Age \not\perp Radio | \varnothing$
  - $Age \perp Radio | Battery$
- **Fork**: $Radio \leftarrow Battery \to Lights$
  - $Radio \not\perp Lights | \varnothing$
  - $Radio \perp Lights | Battery$
- **Collider**: $Battery \to Lights \leftarrow Bulb$
  - $Battery \perp Bulb | \varnothing$
  - $Battery \not\perp Bulb | Lights$
- **Chain + collider**: $Age \to Battery \to Lights \leftarrow Bulb$
  - $Age \perp Bulb | \varnothing$
  - $Age \not\perp Bulb | Lights$
  - $Age \perp Bulb | Lights, Battery$

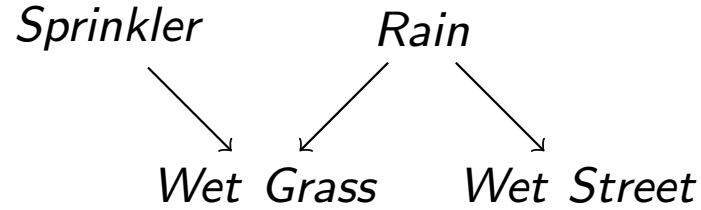
Bayesian Networks: Separation



- A path in G is a sequence of distinct and adjacent nodes, i.e. the direction of the edge is irrelevant. A node B is a descendant of a node A in G if there is a path $A \to \ldots \to B$.
  - E.g., *Age* $\to$ *Battery* $\to$ *Lights* $\leftarrow$ *Bulb* is a path.
  - E.g., *Lights* is a descendant of *Age*.
- Let ρ be a path in G between the nodes α and β.
- A node B in ρ is a **collider** when $A \to B \leftarrow C$ is a subpath of ρ.
  - E.g., *Lights* is a collider in the path *Age* $\to$ *Battery* $\to$ *Lights* $\leftarrow$ *Bulb*.
- Moreover, ρ is Z-**open** with $Z \subseteq X \setminus \{\alpha, \beta\}$ when
  - no non-collider in ρ is in Z, and
  - every collider in ρ is in Z or has a descendant in Z.
  - E.g., the path *Age* $\to$ *Battery* $\to$ *Lights* $\leftarrow$ *Bulb* is Z-open with $Z = \{Lights\}$.
- Let U, V and Z be three disjoint subsets of X. Then, U and V are **separated** given Z in G (i.e. $U \perp_G V | Z$) when there is no Z-open path in G between a node in U and a node in V.
  - E.g., $Age, Battery \perp_G Bulb | \varnothing$.

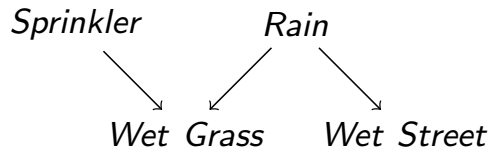
Bayesian Networks: Separation

- The separation criterion is **sound**, i.e. if $U \perp_G V|Z$ then $U \perp_p V|Z$.
- For instance, $S \perp_p R$ and $S \perp_p WS|WG, R$ follow from the DAG



- Note that we read independencies from G, never dependencies. That is, $S \not\perp_G R|WG$ and $S \not\perp_G WS|WG$ follow from the DAG, but not $S \not\perp_p R|WG$ or $S \not\perp_p WS|WG$.
- **Exercise**. Prove that $A \perp_p B|C$ for the DAGs $A \to C \to B$, $A \leftarrow C \to B$ and $A \leftarrow C \leftarrow B$, i.e. prove that $p(a, b|c) = p(a|c)p(b|c)$.
- **Exercise**. Prove that $A \perp_p B|\varnothing$ for the DAG $A \to C \leftarrow B$, i.e. prove that $p(a, b) = p(a)p(b)$.
- **Exercise**. Find the minimal set of nodes that separates a given node from the rest. This set is called the Markov blanket of the given node.
- **Exercise**. How many free parameters do we have in the wet grass BN ? How many do we have if we specify the distribution without the assistance of a BN, i.e. as a table ?

Bayesian Networks: Causal Inference

Original	After $do(r_1)$
	
$q(s) = (0.3, 0.7)$ $q(r) = (0.5, 0.5)$ $q(wg\|r_0, s_0) = (0.1, 0.9)$ $q(wg\|r_0, s_1) = (0.7, 0.3)$ $q(wg\|r_1, s_0) = (0.8, 0.2)$ $q(wg\|r_1, s_1) = (0.9, 0.1)$ $q(ws\|r_0) = (0.1, 0.9)$ $q(ws\|r_1) = (0.7, 0.3)$ $p(s, r, wg, ws) = q(s)q(r)q(wg\|s, r)q(ws\|r)$	$q(s) = (0.3, 0.7)$ $q(wg\|s_0) = (0.8, 0.2)$ $q(wg\|s_1) = (0.9, 0.1)$ $q(ws) = (0.7, 0.3)$ $p(s, wg, ws) = q(s)q(wg\|s)q(ws)$

- What would be the state of the system if a random variable X_j is forced to take the state x_j ?
  - Remove X_j and all the edges from and to X_j from G.
  - Remove $q(x_j|pa_j)$.
  - If $X_j \in Pa_i$, then replace $q(x_i|pa_i)$ with $q(x_i|pa_i \setminus x_j, x_j)$
  - Set $p(x \setminus x_j|do(x_j)) = \prod_i q(x_i|pa_i)$.
- So, the result of $do(x)$ on a BN **is a BN**. No more on causality in this course.

Bayesian Networks: Probabilistic Inference

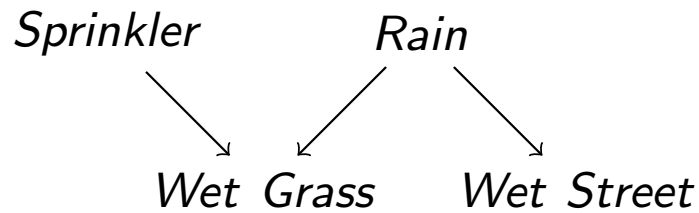
- What is the state of a random variable X_k if a random variable X_i is observed to be in the state x_i ?

$$p(x_k|x_i) = \frac{p(x_k, x_i)}{p(x_i)} = \frac{\sum_{x \setminus \{x_i, x_k\}} p(x)}{\sum_{x \setminus x_i} p(x)}$$

- For instance,

$$p(ws|s) = \frac{\sum_{r,wg} p(r, wg, ws, s)}{\sum_{r,wg,ws} p(r, wg, ws, s)} = \frac{\sum_{r,wg} q(s)q(r)q(wg|s, r)q(ws|r)}{\sum_{r,wg,ws} q(s)q(r)q(wg|s, r)q(ws|r)}$$

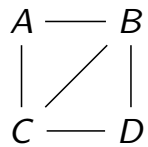
for the DAG



- Answering questions like the one above can be computationally hard.
- A BN is an efficient (because it uses the independences encoded) formalism to compute a posterior probability distribution from a prior probability distribution in the light of observations, hence the name. More on probabilistic inference in Lecture 2.

Markov Networks: Definition

- A BN represents asymmetric (causal) relations, whereas a Markov network represents **symmetric** relations, e.g. physical laws.

UG	Potentials assuming binary random variables
	$\varphi(a,b,c) = (0,0,0,0,1,1,1,1)$ $\varphi(b,c,d) = (1,2,3,4,5,6,7,8)$ $p(a,b,c,d) = \varphi(a,b,c)\varphi(b,c,d)/Z$ with $Z = \sum_{a,b,c,d} \varphi(a,b,c)\varphi(b,c,d)$

- A **Markov network (MN)** over X consists of
  - an undirected graph (UG) G whose nodes are the elements in X, and
  - a set of non-negative functions $\varphi(k)$ over the cliques $Cl(G)$ of G, i.e. the maximal complete sets of nodes in G. The functions are called potentials. They represent **compatibility** relations between the random variables in the cliques.
- The MN represents a probabilistic model of the system, namely

$$p(x) = \frac{1}{Z} \prod_{K \in Cl(G)} \varphi(k)$$

where Z is a normalization constant, i.e.

$$Z = \sum_{x} \prod_{K \in Cl(G)} \varphi(k)$$

- Clearly, $p(x)$ is a probability distribution.

Markov Networks: Separation

- We now show that many of the independencies in p can be read off G without numerical calculations.
- A path ρ in G between two nodes α and β is Z-**open** with $Z \subseteq X \setminus \{\alpha, \beta\}$ when no node in ρ is in Z.
- Let U, V and Z be three disjoint subsets of X. Then, U and V are **separated** given Z in G (i.e. $U \perp_G V|Z$) when there is no Z-open path in G between a node in U and a node in V.
- The separation criterion is **sound**, i.e. if $U \perp_G V|Z$ then $U \perp_p V|Z$.

Markov Networks: Separation

- **Exercise**. Prove that $A \perp_p B | C$ for the UG $A - C - B$, i.e. prove that $p(a, b|c) = f(a, c)g(b, c)$ for some functions f and g.
- **Exercise**. Find the minimal set of nodes that separates a given node from the rest. This set is called the Markov blanket of the given node.
- **Exercise**. How many free parameters do we have in the ABCD MN ? How many do we have if we specify the distribution without the assistance of a MN ? How many if the variables have three states ?

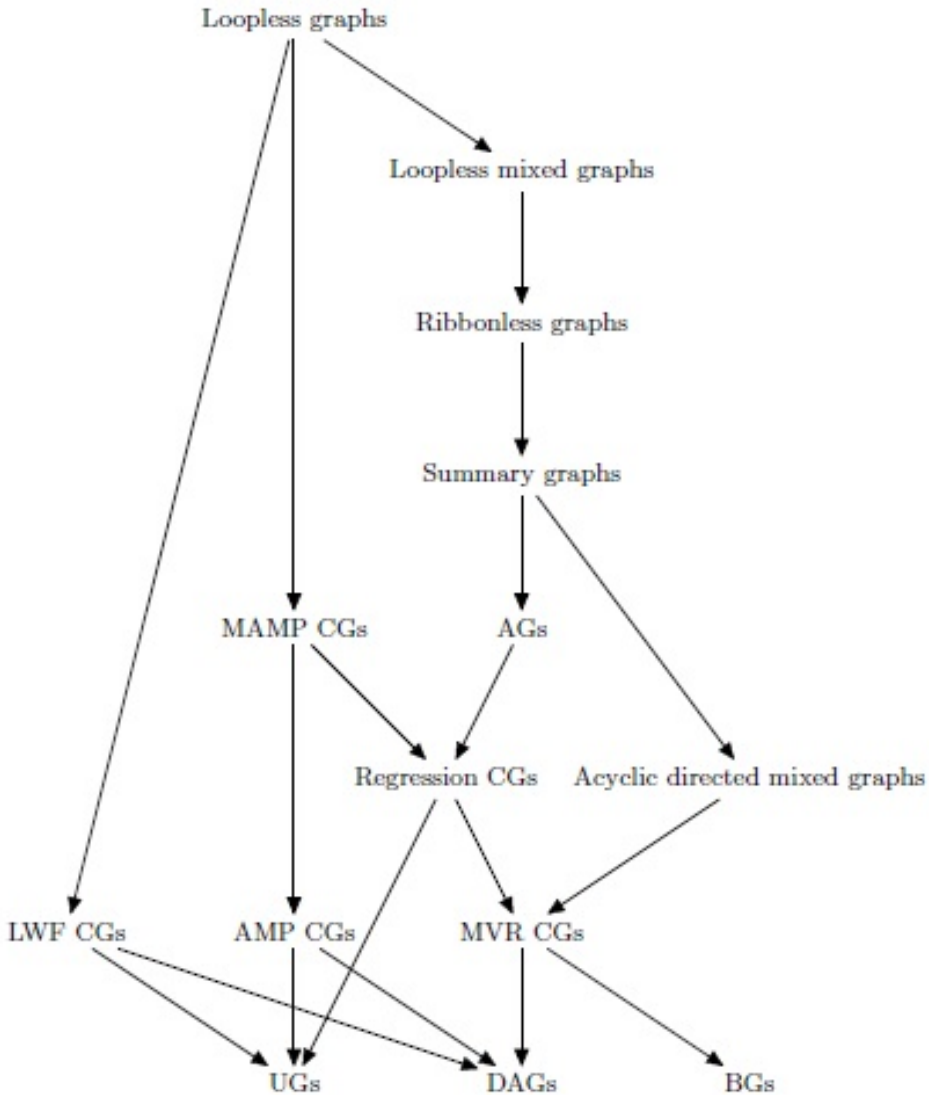
Markov Networks: Probabilistic Inference

- What is the state of a random variable A if a random variable B is observed to be in the state b ?

$$p(a|b) = \frac{\sum_{c,d} \varphi(a,b,c)\varphi(b,c,d)/Z}{\sum_{a,c,d} \varphi(a,b,c)\varphi(b,c,d)/Z}$$

- Answering questions like the one above can be computationally hard.
- A MN is an efficient (because it uses the independences encoded) formalism to answer such questions. More on probabilistic inference in Lecture 2.

Families of Graphical Models



Contents

Thank you

732A96/TDDE15 Advanced Machine Learning

Graphical Models

Jose M. Peña
IDA, Linköping University, Sweden

Lecture 2: Probabilistic Inference

Contents

Literature

- Main source
  - Koller, D. and Friedman, N. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009. Chapter 9.2-9.4.
- Additional source
  - Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006. Chapter 8.

Probabilistic Inference for BNs: Naive Solution

- What is the state of a random variable X_k if a random variable X_i is observed to be in the state x_i ?

$$p(x_k|x_i) = \frac{p(x_k, x_i)}{p(x_i)} = \frac{\sum_{x \setminus \{x_i, x_k\}} p(x)}{\sum_{x \setminus x_i} p(x)}$$

- For instance, $p(d) = \sum_{a,b,c} p(a,b,c,d) = \sum_{a,b,c} p(a)p(b|a)p(c|b)p(d|c)$ for the DAG $A \to B \to C \to D$.

$$\begin{array}{cccc}
 & P(a^1) & P(b^1 \mid a^1) & P(c^1 \mid b^1) & P(d^1 \mid c^1) \\
+ & P(a^2) & P(b^1 \mid a^2) & P(c^1 \mid b^1) & P(d^1 \mid c^1) \\
+ & P(a^1) & P(b^2 \mid a^1) & P(c^1 \mid b^2) & P(d^1 \mid c^1) \\
+ & P(a^2) & P(b^2 \mid a^2) & P(c^1 \mid b^2) & P(d^1 \mid c^1) \\
+ & P(a^1) & P(b^1 \mid a^1) & P(c^2 \mid b^1) & P(d^1 \mid c^2) \\
+ & P(a^2) & P(b^1 \mid a^2) & P(c^2 \mid b^1) & P(d^1 \mid c^2) \\
+ & P(a^1) & P(b^2 \mid a^1) & P(c^2 \mid b^2) & P(d^1 \mid c^2) \\
+ & P(a^2) & P(b^2 \mid a^2) & P(c^2 \mid b^2) & P(d^1 \mid c^2)
\end{array}$$

$$\begin{array}{cccc}
 & P(a^1) & P(b^1 \mid a^1) & P(c^1 \mid b^1) & P(d^2 \mid c^1) \\
+ & P(a^2) & P(b^1 \mid a^2) & P(c^1 \mid b^1) & P(d^2 \mid c^1) \\
+ & P(a^1) & P(b^2 \mid a^1) & P(c^1 \mid b^2) & P(d^2 \mid c^1) \\
+ & P(a^2) & P(b^2 \mid a^2) & P(c^1 \mid b^2) & P(d^2 \mid c^1) \\
+ & P(a^1) & P(b^1 \mid a^1) & P(c^2 \mid b^1) & P(d^2 \mid c^2) \\
+ & P(a^2) & P(b^1 \mid a^2) & P(c^2 \mid b^1) & P(d^2 \mid c^2) \\
+ & P(a^1) & P(b^2 \mid a^1) & P(c^2 \mid b^2) & P(d^2 \mid c^2) \\
+ & P(a^2) & P(b^2 \mid a^2) & P(c^2 \mid b^2) & P(d^2 \mid c^2)
\end{array}$$

Figure 9.2 Computing $P(D)$ by summing over the joint distribution for a chain $A \to B \to C \to D$; all of the variables are binary valued.

Probabilistic Inference for BNs: Naive Solution

$$
\begin{array}{llll}
 & P(a^1) & P(b^1 \mid a^1) & P(c^1 \mid b^1) & P(d^1 \mid c^1) \\
+ & P(a^2) & P(b^1 \mid a^2) & P(c^1 \mid b^1) & P(d^1 \mid c^1) \\
+ & P(a^1) & P(b^2 \mid a^1) & P(c^1 \mid b^2) & P(d^1 \mid c^1) \\
+ & P(a^2) & P(b^2 \mid a^2) & P(c^1 \mid b^2) & P(d^1 \mid c^1) \\
+ & P(a^1) & P(b^1 \mid a^1) & P(c^2 \mid b^1) & P(d^1 \mid c^2) \\
+ & P(a^2) & P(b^1 \mid a^2) & P(c^2 \mid b^1) & P(d^1 \mid c^2) \\
+ & P(a^1) & P(b^2 \mid a^1) & P(c^2 \mid b^2) & P(d^1 \mid c^2) \\
+ & P(a^2) & P(b^2 \mid a^2) & P(c^2 \mid b^2) & P(d^1 \mid c^2)
\end{array}
$$

$$
\begin{array}{llll}
 & P(a^1) & P(b^1 \mid a^1) & P(c^1 \mid b^1) & P(d^2 \mid c^1) \\
+ & P(a^2) & P(b^1 \mid a^2) & P(c^1 \mid b^1) & P(d^2 \mid c^1) \\
+ & P(a^1) & P(b^2 \mid a^1) & P(c^1 \mid b^2) & P(d^2 \mid c^1) \\
+ & P(a^2) & P(b^2 \mid a^2) & P(c^1 \mid b^2) & P(d^2 \mid c^1) \\
+ & P(a^1) & P(b^1 \mid a^1) & P(c^2 \mid b^1) & P(d^2 \mid c^2) \\
+ & P(a^2) & P(b^1 \mid a^2) & P(c^2 \mid b^1) & P(d^2 \mid c^2) \\
+ & P(a^1) & P(b^2 \mid a^1) & P(c^2 \mid b^2) & P(d^2 \mid c^2) \\
+ & P(a^2) & P(b^2 \mid a^2) & P(c^2 \mid b^2) & P(d^2 \mid c^2)
\end{array}
$$

Figure 9.2 Computing $P(D)$ by summing over the joint distribution for a chain $A \rightarrow B \rightarrow C \rightarrow D$; all of the variables are binary valued.

$$
\begin{array}{lll}
 & (P(a^1)P(b^1 \mid a^1) + P(a^2)P(b^1 \mid a^2)) & P(c^1 \mid b^1) & P(d^1 \mid c^1) \\
+ & (P(a^1)P(b^2 \mid a^1) + P(a^2)P(b^2 \mid a^2)) & P(c^1 \mid b^2) & P(d^1 \mid c^1) \\
+ & (P(a^1)P(b^1 \mid a^1) + P(a^2)P(b^1 \mid a^2)) & P(c^2 \mid b^1) & P(d^1 \mid c^2) \\
+ & (P(a^1)P(b^2 \mid a^1) + P(a^2)P(b^2 \mid a^2)) & P(c^2 \mid b^2) & P(d^1 \mid c^2)
\end{array}
$$

$$
\begin{array}{lll}
 & (P(a^1)P(b^1 \mid a^1) + P(a^2)P(b^1 \mid a^2)) & P(c^1 \mid b^1) & P(d^2 \mid c^1) \\
+ & (P(a^1)P(b^2 \mid a^1) + P(a^2)P(b^2 \mid a^2)) & P(c^1 \mid b^2) & P(d^2 \mid c^1) \\
+ & (P(a^1)P(b^1 \mid a^1) + P(a^2)P(b^1 \mid a^2)) & P(c^2 \mid b^1) & P(d^2 \mid c^2) \\
+ & (P(a^1)P(b^2 \mid a^1) + P(a^2)P(b^2 \mid a^2)) & P(c^2 \mid b^2) & P(d^2 \mid c^2)
\end{array}
$$

Figure 9.3 The first transformation on the sum of figure 9.2

Probabilistic Inference for BNs: Naive Solution

$$
\begin{array}{rlll}
 & (P(a^1)P(b^1 \mid a^1) + P(a^2)P(b^1 \mid a^2)) & P(c^1 \mid b^1) & P(d^1 \mid c^1) \\
+ & (P(a^1)P(b^2 \mid a^1) + P(a^2)P(b^2 \mid a^2)) & P(c^1 \mid b^2) & P(d^1 \mid c^1) \\
+ & (P(a^1)P(b^1 \mid a^1) + P(a^2)P(b^1 \mid a^2)) & P(c^2 \mid b^1) & P(d^1 \mid c^2) \\
+ & (P(a^1)P(b^2 \mid a^1) + P(a^2)P(b^2 \mid a^2)) & P(c^2 \mid b^2) & P(d^1 \mid c^2) \\
 & & & \\
 & (P(a^1)P(b^1 \mid a^1) + P(a^2)P(b^1 \mid a^2)) & P(c^1 \mid b^1) & P(d^2 \mid c^1) \\
+ & (P(a^1)P(b^2 \mid a^1) + P(a^2)P(b^2 \mid a^2)) & P(c^1 \mid b^2) & P(d^2 \mid c^1) \\
+ & (P(a^1)P(b^1 \mid a^1) + P(a^2)P(b^1 \mid a^2)) & P(c^2 \mid b^1) & P(d^2 \mid c^2) \\
+ & (P(a^1)P(b^2 \mid a^1) + P(a^2)P(b^2 \mid a^2)) & P(c^2 \mid b^2) & P(d^2 \mid c^2)
\end{array}
$$

Figure 9.3 The first transformation on the sum of figure 9.2

$$
\begin{array}{rlll}
 & \tau_1(b^1) & P(c^1 \mid b^1) & P(d^1 \mid c^1) \\
+ & \tau_1(b^2) & P(c^1 \mid b^2) & P(d^1 \mid c^1) \\
+ & \tau_1(b^1) & P(c^2 \mid b^1) & P(d^1 \mid c^2) \\
+ & \tau_1(b^2) & P(c^2 \mid b^2) & P(d^1 \mid c^2) \\
 & & & \\
 & \tau_1(b^1) & P(c^1 \mid b^1) & P(d^2 \mid c^1) \\
+ & \tau_1(b^2) & P(c^1 \mid b^2) & P(d^2 \mid c^1) \\
+ & \tau_1(b^1) & P(c^2 \mid b^1) & P(d^2 \mid c^2) \\
+ & \tau_1(b^2) & P(c^2 \mid b^2) & P(d^2 \mid c^2)
\end{array}
$$

Figure 9.4 The second transformation on the sum of figure 9.2

Probabilistic Inference for BNs: Naive Solution

$$
\begin{array}{lll}
 & \tau_1(b^1) & P(c^1 \mid b^1) \quad P(d^1 \mid c^1) \\
+ & \tau_1(b^2) & P(c^1 \mid b^2) \quad P(d^1 \mid c^1) \\
+ & \tau_1(b^1) & P(c^2 \mid b^1) \quad P(d^1 \mid c^2) \\
+ & \tau_1(b^2) & P(c^2 \mid b^2) \quad P(d^1 \mid c^2)
\end{array}
$$

$$
\begin{array}{lll}
 & \tau_1(b^1) & P(c^1 \mid b^1) \quad P(d^2 \mid c^1) \\
+ & \tau_1(b^2) & P(c^1 \mid b^2) \quad P(d^2 \mid c^1) \\
+ & \tau_1(b^1) & P(c^2 \mid b^1) \quad P(d^2 \mid c^2) \\
+ & \tau_1(b^2) & P(c^2 \mid b^2) \quad P(d^2 \mid c^2)
\end{array}
$$

Figure 9.4 The second transformation on the sum of figure 9.2

$$
\begin{array}{lll}
 & (\tau_1(b^1)P(c^1 \mid b^1) + \tau_1(b^2)P(c^1 \mid b^2)) & P(d^1 \mid c^1) \\
+ & (\tau_1(b^1)P(c^2 \mid b^1) + \tau_1(b^2)P(c^2 \mid b^2)) & P(d^1 \mid c^2)
\end{array}
$$

$$
\begin{array}{lll}
 & (\tau_1(b^1)P(c^1 \mid b^1) + \tau_1(b^2)P(c^1 \mid b^2)) & P(d^2 \mid c^1) \\
+ & (\tau_1(b^1)P(c^2 \mid b^1) + \tau_1(b^2)P(c^2 \mid b^2)) & P(d^2 \mid c^2)
\end{array}
$$

Figure 9.5 The third transformation on the sum of figure 9.2

Probabilistic Inference for BNs: Naive Solution

$$
\begin{array}{rll}
 & (\tau_1(b^1)P(c^1 \mid b^1) + \tau_1(b^2)P(c^1 \mid b^2)) & P(d^1 \mid c^1) \\
+ & (\tau_1(b^1)P(c^2 \mid b^1) + \tau_1(b^2)P(c^2 \mid b^2)) & P(d^1 \mid c^2) \\
 & & \\
 & (\tau_1(b^1)P(c^1 \mid b^1) + \tau_1(b^2)P(c^1 \mid b^2)) & P(d^2 \mid c^1) \\
+ & (\tau_1(b^1)P(c^2 \mid b^1) + \tau_1(b^2)P(c^2 \mid b^2)) & P(d^2 \mid c^2)
\end{array}
$$

Figure 9.5 The third transformation on the sum of figure 9.2

$$
\begin{array}{rll}
 & \tau_2(c^1) & P(d^1 \mid c^1) \\
+ & \tau_2(c^2) & P(d^1 \mid c^2) \\
 & & \\
 & \tau_2(c^1) & P(d^2 \mid c^1) \\
+ & \tau_2(c^2) & P(d^2 \mid c^2)
\end{array}
$$

Figure 9.6 The fourth transformation on the sum of figure 9.2

- Then, instead of $p(d) = \sum_{a,b,c} p(a)p(b|a)p(c|b)p(d|c)$, we can do

$$
\begin{aligned}
p(d) &= \sum_c p(d|c)\tau_2(c) \\
&= \sum_c p(d|c) \sum_b p(c|b)\tau_1(b) \\
&= \sum_c p(d|c) \sum_b p(c|b) \sum_a p(a)p(b|a)
\end{aligned}
$$

- Do we gain anything ? **Yes**, the former case implies 62 operations (multiplications and additions) and the latter only 18.

Probabilistic Inference for BNs: Variable Elimination

- Let us define a factor $\phi(U)$ as a function $\phi : \mathit{Values}(U) \to \mathbb{R}$. Let us also define $\mathit{Scope}(\phi) = U$.

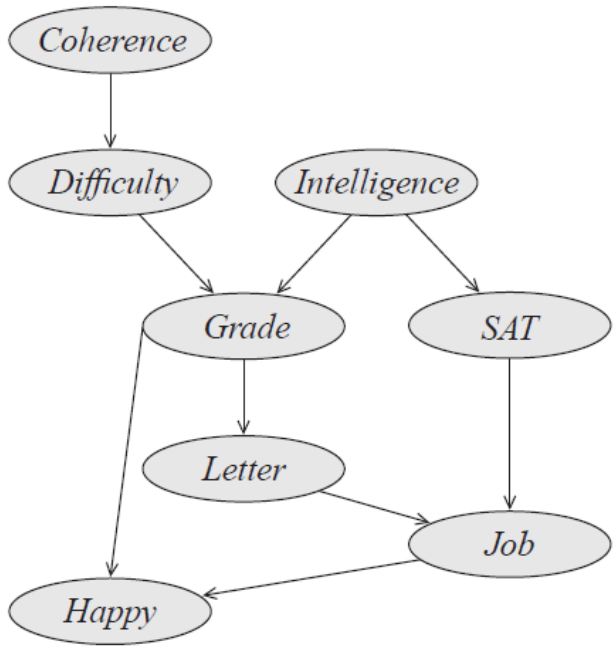


Figure 9.8 **The** Extended-Student **Bayesian network**

$$\begin{aligned} P(C,D,I,G,S,L,J,H) &= P(C)P(D \mid C)P(I)P(G \mid I,D)P(S \mid I) \\ &\quad P(L \mid G)P(J \mid L,S)P(H \mid G,J) \\ &= \phi_C(C)\phi_D(D,C)\phi_I(I)\phi_G(G,I,D)\phi_S(S,I) \\ &\quad \phi_L(L,G)\phi_J(J,L,S)\phi_H(H,G,J). \end{aligned}$$

Probabilistic Inference for BNs: Variable Elimination

- The following algorithm returns $p(y)$ where $Y = X \setminus Z$.

Algorithm 9.1 Sum-product variable elimination algorithm

Procedure Sum-Product-VE (
  Φ, // Set of factors
  $\boldsymbol{Z}$, // Set of variables to be eliminated
  $\prec$ // Ordering on $\boldsymbol{Z}$
)
1 Let $Z_1, \ldots, Z_k$ be an ordering of $\boldsymbol{Z}$ such that
2 $Z_i \prec Z_j$ if and only if $i < j$
3 **for** $i = 1, \ldots, k$
4 $\Phi \leftarrow$ Sum-Product-Eliminate-Var(Φ, Z_i)
5 $\phi^* \leftarrow \prod_{\phi \in \Phi} \phi$
6 **return** ϕ^*

Procedure Sum-Product-Eliminate-Var (
  Φ, // Set of factors
  Z // Variable to be eliminated
)
1 $\Phi' \leftarrow \{\phi \in \Phi : Z \in Scope[\phi]\}$
2 $\Phi'' \leftarrow \Phi - \Phi'$
3 $\psi \leftarrow \prod_{\phi \in \Phi'} \phi$
4 $\tau \leftarrow \sum_Z \psi$
5 **return** $\Phi'' \cup \{\tau\}$

Probabilistic Inference for BNs: Variable Elimination

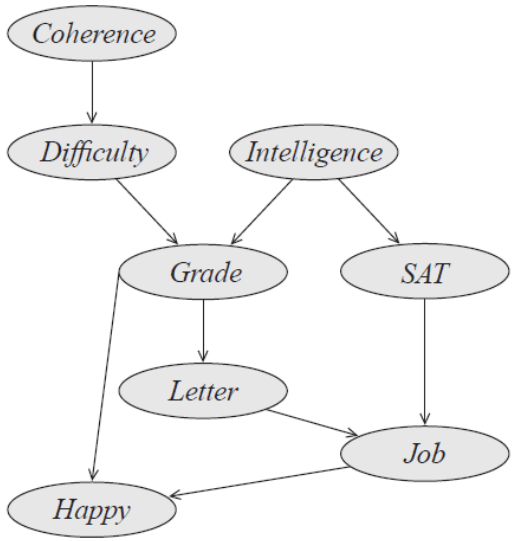


Figure 9.8 **The** Extended-Student **Bayesian network**

$$
\begin{aligned}
P(C,D,I,G,S,L,J,H) &= P(C)P(D \mid C)P(I)P(G \mid I,D)P(S \mid I) \\
&\quad P(L \mid G)P(J \mid L,S)P(H \mid G,J) \\
&= \phi_C(C)\phi_D(D,C)\phi_I(I)\phi_G(G,I,D)\phi_S(S,I) \\
&\quad \phi_L(L,G)\phi_J(J,L,S)\phi_H(H,G,J).
\end{aligned}
$$

Step	Variable eliminated	Factors used	Variables involved	New factor
1	C	$\phi_C(C)$, $\phi_D(D,C)$	C,D	$\tau_1(D)$
2	D	$\phi_G(G,I,D)$, $\tau_1(D)$	G,I,D	$\tau_2(G,I)$
3	I	$\phi_I(I)$, $\phi_S(S,I)$, $\tau_2(G,I)$	G,S,I	$\tau_3(G,S)$
4	H	$\phi_H(H,G,J)$	H,G,J	$\tau_4(G,J)$
5	G	$\tau_4(G,J)$, $\tau_3(G,S)$, $\phi_L(L,G)$	G,J,L,S	$\tau_5(J,L,S)$
6	S	$\tau_5(J,L,S)$, $\phi_J(J,L,S)$	J,L,S	$\tau_6(J,L)$
7	L	$\tau_6(J,L)$	J,L	$\tau_7(J)$

Table 9.1 **A run of variable elimination for the query** $P(J)$

Probabilistic Inference for BNs: Variable Elimination

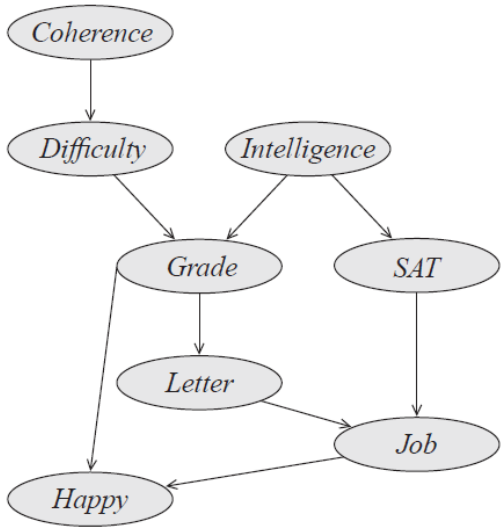


Figure 9.8 The Extended-Student **Bayesian network**

$$\begin{aligned} P(C,D,I,G,S,L,J,H) &= P(C)P(D \mid C)P(I)P(G \mid I,D)P(S \mid I) \\ &\quad P(L \mid G)P(J \mid L,S)P(H \mid G,J) \\ &= \phi_C(C)\phi_D(D,C)\phi_I(I)\phi_G(G,I,D)\phi_S(S,I) \\ &\quad \phi_L(L,G)\phi_J(J,L,S)\phi_H(H,G,J). \end{aligned}$$

Step	Variable eliminated	Factors used	Variables involved	New factor
1	G	$\phi_G(G,I,D)$, $\phi_L(L,G)$, $\phi_H(H,G,J)$	G,I,D,L,J,H	$\tau_1(I,D,L,J,H)$
2	I	$\phi_I(I)$, $\phi_S(S,I)$, $\tau_1(I,D,L,S,J,H)$	S,I,D,L,J,H	$\tau_2(D,L,S,J,H)$
3	S	$\phi_J(J,L,S)$, $\tau_2(D,L,S,J,H)$	D,L,S,J,H	$\tau_3(D,L,J,H)$
4	L	$\tau_3(D,L,J,H)$	D,L,J,H	$\tau_4(D,J,H)$
5	H	$\tau_4(D,J,H)$	D,J,H	$\tau_5(D,J)$
6	C	$\phi_C(C)$, $\phi_D(D,C)$	D,J,C	$\tau_6(D)$
7	D	$\tau_5(D,J)$, $\tau_6(D)$	D,J	$\tau_7(J)$

Table 9.2 A different run of variable elimination for the query $P(J)$

Probabilistic Inference for BNs: Variable Elimination

- What is the state of a random variable X_k if a random variable X_i is observed to be in the state x_i ?

$$p(x_k|x_i) = \frac{p(x_k, x_i)}{p(x_i)} = \frac{\sum_{x \setminus \{x_i, x_k\}} p(x)}{\sum_{x \setminus x_i} p(x)}$$

- E.g., $p(d|a) = \frac{p(d,a)}{p(a)} = \frac{\sum_{b,c} p(a)p(b|a)p(c|b)p(d|c)}{\sum_{b,c,d} p(a)p(b|a)p(c|b)p(d|c)}$ for $A \to B \to C \to D$.
- The following algorithm returns $p(y|e) = \frac{p(y,e)}{p(e)} = \frac{\phi^*}{\alpha}$ where $Y \subseteq X \setminus E$.
- Given a factor $\phi(U)$, let us define the reduced factor $\phi[E = e](Y)$ as a factor with scope $Y = U \setminus E$ such that $\phi[E = e](y) = \phi(y, z)$ where $Z = U \cap E$.

Algorithm 9.2 Using Sum-Product-VE **for computing conditional probabilities**

Procedure Cond-Prob-VE (
  $\mathcal{K}$, // A network over $\mathcal{X}$
  $\boldsymbol{Y}$, // Set of query variables
  $\boldsymbol{E} = \boldsymbol{e}$ // Evidence
)

1 $\Phi \leftarrow$ Factors parameterizing $\mathcal{K}$
2 Replace each $\phi \in \Phi$ by $\phi[\boldsymbol{E} = \boldsymbol{e}]$
3 Select an elimination ordering $\prec$
4 $\boldsymbol{Z} \leftarrow = \mathcal{X} - \boldsymbol{Y} - \boldsymbol{E}$
5 $\phi^* \leftarrow$ Sum-Product-VE$(\Phi, \prec, \boldsymbol{Z})$
6 $\alpha \leftarrow \sum_{\boldsymbol{y} \in Val(\boldsymbol{Y})} \phi^*(\boldsymbol{y})$
7 **return** α, ϕ^*

Probabilistic Inference for BNs: Variable Elimination

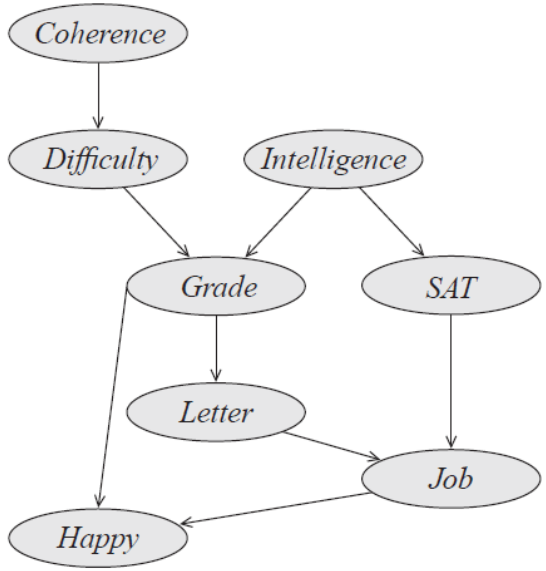


Figure 9.8 The Extended-Student **Bayesian network**

$$
\begin{aligned}
P(C,D,I,G,S,L,J,H) &= P(C)P(D \mid C)P(I)P(G \mid I,D)P(S \mid I) \\
&\quad P(L \mid G)P(J \mid L,S)P(H \mid G,J) \\
&= \phi_C(C)\phi_D(D,C)\phi_I(I)\phi_G(G,I,D)\phi_S(S,I) \\
&\quad \phi_L(L,G)\phi_J(J,L,S)\phi_H(H,G,J).
\end{aligned}
$$

Step	Variable eliminated	Factors used	Variables involved	New factor
1'	C	$\phi_C(C)$, $\phi_D(D,C)$	C,D	$\tau'_1(D)$
2'	D	$\phi_G[I=i^1](G,D)$, $\phi_I[I=i^1]()$, $\tau'_1(D)$	G,D	$\tau'_2(G)$
5'	G	$\tau'_2(G)$, $\phi_L(L,G)$, $\phi_H[H=h^0](G,J)$	G,L,J	$\tau'_5(L,J)$
6'	S	$\phi_S[I=i^1](S)$, $\phi_J(J,L,S)$	J,L,S	$\tau'_6(J,L)$
7'	L	$\tau'_6(J,L)$, $\tau'_5(J,L)$	J,L	$\tau'_7(J)$

Table 9.3 A run of sum-product variable elimination for $P(J,i^1,h^0)$

Probabilistic Inference for MNs

- The VE algorithm can also be used for probabilistic inference in MNs. Simply,
  - initialize the set of factors Φ to the MN's clique potentials $\{\varphi(k)\}$,
  - run the VE algorithm, and
  - normalize the returned unnormalized probability distribution by dividing with the MN's normalization constant Z.

Beyond Variable Elimination

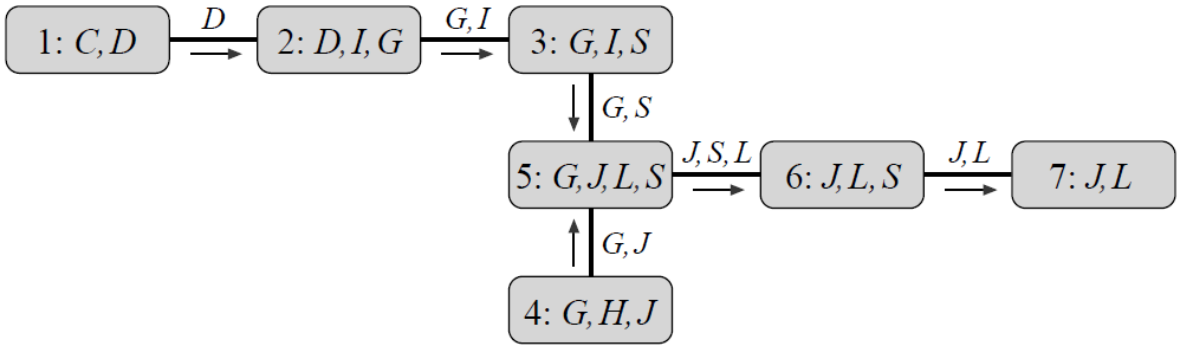


Figure 10.1 Cluster tree for the VE execution in table 9.1

- The execution of VE defines a cluster tree, a graphical flowchart of the factor-manipulation process:
  - Each node C_i in the tree is called a cluster and it contains the variables in $Scope(\psi_i)$.
  - The tree has an edge annotated with an arrow from C_i to C_j if the factor (a.k.a. message) τ_i is used in the computation of τ_j.
- Now, consider passing messages towards the cluster C_6 in order to compute $p(s)$. Notice the **reusing** of previously computed messages.
- Therefore, a clique tree is a data structure to perform **repeated** probabilistic inference efficiently.
- The process of building the cluster tree from a given DAG is known as compilation. You will learn more about cluster trees in the lab.

Contents

Thank you

732A96/TDDE15 Advanced Machine Learning

Graphical Models

Jose M. Peña
IDA, Linköping University, Sweden

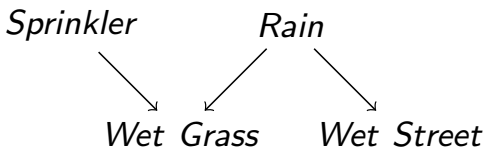
Lecture 3: Parameter Learning

Contents

Literature

- Main source
  - Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006. Chapter 8.
- Additional source
  - Koski, T. J. T. and Noble, J. M. A Review of Bayesian Networks and Structure Learning. *Mathematica Applicanda* 40, 51-103, 2012.

Parameter Learning for BNs: Maximum Likelihood

DAG	Parameter values for the conditional probability distributions
	$q(s) = (0.3, 0.7) = (\theta_{s_0}, \theta_{s_1})$ $q(r) = (0.5, 0.5) = (\theta_{r_0}, \theta_{r_1})$ $q(wg\|r_0, s_0) = (0.1, 0.9) = (\theta_{wg_0\|r_0,s_0}, \theta_{wg_1\|r_0,s_0})$ $q(wg\|r_0, s_1) = (0.7, 0.3) = (\theta_{wg_0\|r_0,s_1}, \theta_{wg_1\|r_0,s_1})$ $q(wg\|r_1, s_0) = (0.8, 0.2) = (\theta_{wg_0\|r_1,s_0}, \theta_{wg_1\|r_1,s_0})$ $q(wg\|r_1, s_1) = (0.9, 0.1) = (\theta_{wg_0\|r_1,s_1}, \theta_{wg_1\|r_1,s_1})$ $q(ws\|r_0) = (0.1, 0.9) = (\theta_{ws_0\|r_0}, \theta_{ws_1\|r_0})$ $q(ws\|r_1) = (0.7, 0.3) = (\theta_{ws_0\|r_1}, \theta_{ws_1\|r_1})$ $p(s, r, wg, ws) = q(s)q(r)q(wg\|s, r)q(ws\|r)$

- In general,

$$q(X_i = k|Pa_i = j) = \theta_{X_i=k|Pa_i=j}$$

- Recall that

$$p(X_i = k|Pa_i = j) = q(X_i = k|Pa_i = j)$$

Parameter Learning for BNs: Maximum Likelihood

- Given a sample $d_{1:N}$, the log likelihood function is

$$\log p(d_{1:N}|\theta, G) = \log \prod_l p(d_l|\theta, G) = \log \prod_l \prod_i p(d_l[X_i]|d_l[Pa_i], \theta)$$

$$= \log \prod_l \prod_i \theta_{X_i=d_l[X_i]|Pa_i=d_l[Pa_i]} = \log \prod_i \prod_j \prod_k \theta_{X_i=k|Pa_i=j}^{N_{ijk}}$$

$$= \sum_i \sum_j \sum_k N_{ijk} \log \theta_{X_i=k|Pa_i=j}$$

  where N_{ijk} is the number of instances in $d_{1:N}$ with $X_i = k$ and $Pa_i = j$.

- To maximize the log likelihood function subject to the constraint $\sum_k \theta_{X_i=k|Pa_i=j} = 1$ for all i and j, we maximize

$$\sum_i \sum_j \sum_k N_{ijk} \log \theta_{X_i=k|Pa_i=j} + \sum_i \sum_j \lambda_{ij} (\sum_k \theta_{X_i=k|Pa_i=j} - 1)$$

  where λ_{ij} are called Lagrange multipliers.[1]

- Setting to zero the derivative with respect to $\theta_{X_i=k|Pa_i=j}$ gives

$$\theta_{X_i=k|Pa_i=j} = -N_{ijk}/\lambda_{ij}$$

- Replacing in the constraint gives $\lambda_{ij} = -N_{ij}$ and $\theta^{ML}_{X_i=k|Pa_i=j} = N_{ijk}/N_{ij}$.

[1] Any stationary point of the Lagrangian function is a stationary point of the original function subject to the constraints. Moreover, the log likelihood function is concave.

Parameter Learning for MNs: Iterative Proportional Fitting Procedure

- Given a complete sample $d_{1:N}$, the log likelihood function is

$$\log p(d_{1:N}|\theta, G) = \log \prod_l \frac{\prod_{K \in Cl(G)} \varphi(d_l[K])}{Z} = \sum_{K \in Cl(G)} \sum_k N_k \log \varphi(k) - N \log Z$$

  where N_k is the number of instances in $d_{1:N}$ with $K = k$. Then

$$\log p(d_{1:N}|\theta, G)/N = \sum_{K \in Cl(G)} \sum_k p_e(k) \log \varphi(k) - \log Z$$

  where $p_e(X)$ is the empirical probability distribution obtained from $d_{1:N}$.
- Let $Q \in Cl(G)$. The derivative with respect to $\varphi(q)$ is

$$\frac{\partial \log p(d_{1:N}|\theta, G)/N}{\partial \varphi(q)} = \frac{p_e(q)}{\varphi(q)} - \frac{1}{Z} \frac{\partial Z}{\partial \varphi(q)}$$

- Let $Y = X \setminus Q$. Then

$$\frac{\partial Z}{\partial \varphi(q)} = \sum_y \prod_{K \in Cl(G) \setminus Q} \varphi(k, \overline{k}) = \frac{Z}{\varphi(q)} \sum_y \prod_{K \in Cl(G) \setminus Q} \varphi(k, \overline{k}) \frac{\varphi(q)}{Z} = \frac{Z}{\varphi(q)} p(q|\theta, G)$$

  where $\overline{k}$ denotes the elements of q corresponding to the elements of $K \cap Q$.
- Putting together the results above, we have that

$$\frac{\partial \log p(d_{1:N}|\theta, G)/N}{\partial \varphi(q)} = \frac{p_e(q)}{\varphi(q)} - \frac{p(q|\theta, G)}{\varphi(q)}$$

Parameter Learning for MNs: Iterative Proportional Fitting Procedure

- Setting the derivative to zero gives [2]

$$\varphi^{ML}(q) = \varphi(q)p_e(q)/p(q|\theta, G)$$

No closed form solution but ...

IPFP

Initialize $\varphi(k)$ for all $K \in Cl(G)$
Repeat until convergence
    Set $\varphi(k) = \varphi(k)p_e(k)/p(k|\theta, G)$ for all $K \in Cl(G)$

- IPFP increases $\log p(d_{1:N}|\theta, G)$ in each iteration. So, it is globally optimal.
- Iterative coordinate ascend method.
- Note that computing $p(k|\theta, G)$ in the last line requires inference. Moreover, the multiplication and division are elementwise.
- Note also that Z needs to be computed in each iteration, which is computationally hard. This can be avoided by a careful initialization.

[2] The log likelihood function is concave.

Contents

- Parameter Learning for BNs
  - Maximum Likelihood
- Parameter Learning for MNs
  - Iterative Proportional Fitting Procedure

Thank you

732A96/TDDE15 Advanced Machine Learning

Graphical Models

Jose M. Peña
IDA, Linköping University, Sweden

Lecture 4: Structure Learning

Contents

Literature

- Main source
  - Koski, T. J. T. and Noble, J. M. A Review of Bayesian Networks and Structure Learning. *Mathematica Applicanda* 40, 51-103, 2012.

Structure Learning for BNs: Independence Test Based Approach

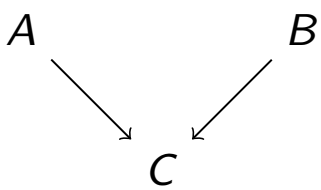
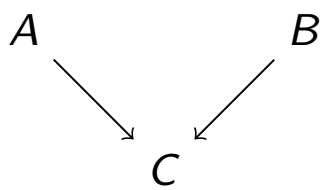
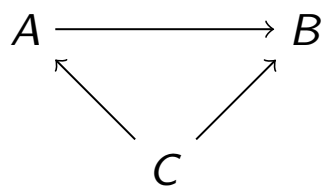
- We can get a DAG G to be used for **probabilistic** reasoning as follows:

Let $Y_{1:n}$ be any ordering of the random variables $X_{1:n}$
For each Y_i do
    Set Pa_i to be any minimal subset of $Y_{1:i-1}$ such that $Y_i \perp_p Y_{1:i-1} \setminus Pa_i | Pa_i$

- **Exercise**. Prove the previous statement.

Structure Learning for BNs: Independence Test Based Approach

- Note that G has the minimum number of edges among the DAGs that are consistent with the ordering considered.
- However, G may not have the minimum number of edges among all the DAGs, i.e. the ordering considered may not be optimal.

$A \perp_p B$	G with ordering A, B, C	G with ordering C, A, B
		

- Since the ordering of the variables in the construction of the DAG is arbitrary, we cannot interpret the DAG as a causal structure and, thus, we should not use it for **causal** reasoning.
- As we will see next, we can get a DAG with minimum number of edges without searching over the $n!$ orderings assuming that p is **faithful** to the true DAG G^*, i.e. $U \perp_p V|Z$ if and only if $U \perp_{G^*} V|Z$. Yet the DAG learned should **only** be used for probabilistic reasoning.

Structure Learning for BNs: Independence Test Based Approach

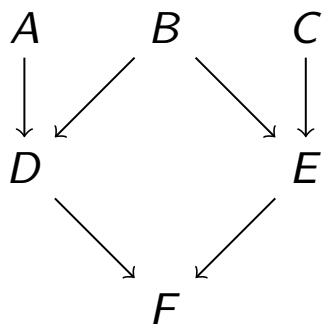
Inductive Causation (IC) algorithm

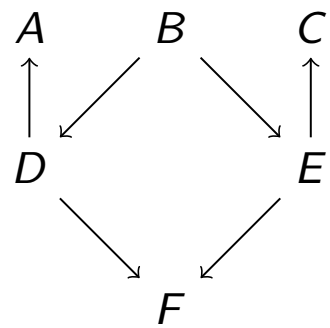
Let G be the complete undirected graph
For each ordered pair of nodes X_i and X_j in G do
    If there is a set $S \subseteq X \setminus \{X_i, X_j\}$ such that $X_i \perp_p X_j | S$
    then remove the edge $X_i - X_j$ from G and set $S_{ij} := S_{ji} := S$
For each ordered pair of non-adjacent nodes X_i and X_j in G do
    If there is a node $X_k \notin S_{ij}$ that is adjacent to both X_i and X_j
    then orient $X_i - X_k - X_j$ as $X_i \rightarrow X_k \leftarrow X_j$ (a.k.a. unshielded collider) in G
For each undirected edge $X_i - X_j$ in G do
    Orient $X_i - X_j$ as $X_i \rightarrow X_j$ if the opposite orientation creates an unshielded collider or a directed cycle

- Parents and children (PC) algorithm: Refinement of the IC algorithm with efficient procedures to find the set S in line 3 and orient the edges in line 9.

Structure Learning for BNs: Independence Test Based Approach

- **Exercise**. Run the IC algorithm assuming that p is faithful to the following DAGs.





Structure Learning for BNs: Independence Test Based Approach

- In practice, we do not have access to p but to a finite sample from it. Then, replace $X_i \perp_p X_j | S$ in the IC algorithm with an independece test, preferably with one that is consistent so that the algorithm is **asymptotically** correct.
- Let $d_{1:N}$ be a complete sample. Then, $X_i \perp_p X_j | S$ implies that $p(x_i, x_j | s) = p(x_i | s) p(x_j | s)$ and thus that

$$N_{x_i, x_j, s} \approx N_{x_i, s} N_{x_j, s} / N_s$$

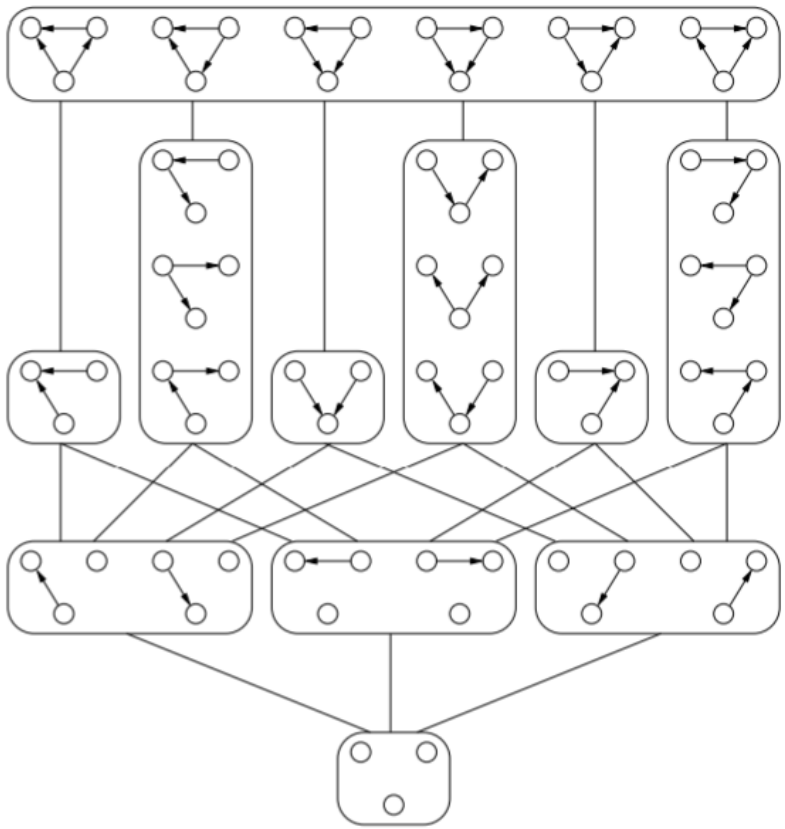
  where $N_{x_i, x_j, s}$ is the number of instances in $d_{1:N}$ where x_i, x_j and s, and $N_{x_i, s} = \sum_{x_j} N_{x_i, x_j, s}$ and $N_{x_j, s} = \sum_{x_i} N_{x_i, x_j, s}$ and $N_s = \sum_{x_i, x_j} N_{x_i, x_j, s}$.
- We can measure the deviance from the expected situation above by

$$\mathit{deviance} = \sum_{x_i, x_j, s} \frac{\left[N_{x_i, x_j, s} - N_{x_i, s} N_{x_j, s} / N_s \right]^2}{N_{x_i, s} N_{x_j, s} / N_s}$$

- If the deviance is too large, then reject the hypothesis that $X_i \perp_p X_j | S$.
- Asymptotically, the deviance follows a χ^2 distribution with the appropriate number of degrees of freedom, i.e. $|S|(|X_i| - 1)(|X_j| - 1)$. Then, we can control the probability of falsely rejecting the hypothesis, a.k.a. p-value.

Structure Learning for BNs: Independence Test Based Approach

- Two DAGs represent the same independencies according to the separation criterion (i.e. they are **equivalent**) if and only if they have the same adjacencies and **unshielded colliders**, i.e. subgraphs $X_i \rightarrow X_k \leftarrow X_j$ where X_i and X_j are not adjacent.



Hasse diagram of the space of Markov equivalence classes of Bayesian network structures over three variables.

Structure Learning for BNs: Independence Test Based Approach

- The output of the IC algorithm is not a DAG in general, but an **essential graph** (EG):
  - The EG G has an edge $X_i \to X_j$ if and only if $X_i \to X_j$ is in **every** DAG that is equivalent to the true DAG G^*.
  - In other words, G has an edge $X_i - X_j$ if and only if $X_i \to X_j$ is in some DAG that is equivalent to G^* and $X_i \leftarrow X_j$ is in some other DAG that is equivalent to G^*.
- A naive way to convert G into a DAG that is equivalent to G^* is as follows:

Repeat while possible

  Replace any edge $X_i - X_j$ in G with $X_i \to X_j$ if this does not create a directed cycle or a new unshielded collider

If G is not a DAG, then backtrack

- Again, G can be used for probabilistic reasoning but not for causal reasoning.

Structure Learning for BNs: Score Based Approach

- Alternatively, we can choose the DAG G with maximum posterior probability (a.k.a **Bayesian score**):

$$p(G|d_{1:N}) = p(d_{1:N}|G)p(G)/P(d_{1:N}) \propto p(d_{1:N}|G)p(G)$$

where $p(d_{1:N}|G)$ is the marginal likelihood of $d_{1:N}$ given G, $p(G)$ is a prior probability distribution, and $p(d_{1:N})$ is a normalization constant.

- Moreover

$$p(d_{1:N}|G) = \int p(d_{1:N}|\theta, G)p(\theta|G)d\theta$$

where $p(d_{1:N}|\theta, G)$ is the likelihood function of $d_{1:N}$ given G and θ, and $p(\theta|G)$ is a prior probability distribution.

- **Assuming** that $p(\theta|G) = \prod_i \prod_j p(\theta_{x_i|Pa_i=j}|G)$ and $p(\theta_{x_i|Pa_i=j}|G) \sim Dirichlet(\alpha_{ij1}, \ldots, \alpha_{ijk_i})$, we have that

$$p(d_{1:N}|G) = \prod_i \prod_j \frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij} + N_{ij})} \prod_k \frac{\Gamma(\alpha_{ijk} + N_{ijk})}{\Gamma(\alpha_{ijk})}$$

where $\alpha_{ij} = \sum_k \alpha_{ijk}$, N_{ijk} is the number of instances in $d_{1:N}$ where $X_i = k$ and $Pa_i = j$, and $N_{ij} = \sum_k N_{ijk}$.

Structure Learning for BNs: Score Based Approach

- The Bayesian score is **score equivalent** (i.e. it gives the same score to equivalent DAGs) if

$$\alpha_{ijk} = \frac{\alpha}{|X_i| \prod_{X_l \in Pa_i} |X_l|}$$

  where α is the user-defined imaginary sample size (the higher the less regularization). This is call the BDeu score.

- Under the Dirichlet parameter prior assumption and when $N \to \infty$, we have that

$$\log p(d_{1:N}|G) \approx \log p(d_{1:N}|\theta^{ML}, G) - \frac{\log N}{2} dim(G)$$

  where $dim(G)$ is the dimension or number of free parameters of G, i.e. $\sum_i (|X_i| - 1) \prod_{X_l \in Pa_i} |X_l|$.

- This approximation is called **Bayesian information criterion** (BIC), and it shows that the Bayesian score favours models that trade off fit of data and model complexity.

Structure Learning for BNs: Score Based Approach

- Number of DAGs with 1-12 nodes: 1, 3, 25, 543, 29281, 3781503, 1138779265, 783702329343, 1213442454842881, 4175098976430598143, 31603459396418917607425, 521939651343829405020504063
- Then, an exhaustive search is prohibitive. Then, a heuristic search must be performed instead.

Hill-climbing (HC)

Let G be the empty DAG
Repeat until no change occurs
    Add, remove or reverse any edge in G that improves the Bayesian score the most

---

- Unlike the IC algorithm, HC is not asymptotically correct under faithfulness, i.e. it may get trapped in local optima. Still, HC is very popular.

Structure Learning for MNs: Independence Test Based Approach

- We can get an UG G to be used for probabilistic reasoning as follows:

For each X_i do
  Set $Ad(X_i)$ to be any minimal subset of $X \setminus X_i$ such that $X_i \perp_p X \setminus Ad(X_i) | Ad(X_i)$

- Luckily, we can get G without searching over the 2^{n-1} possible adjacent sets for each node if we assume that p is **faithful** to the true MN G^*, i.e. $U \perp_p V | Z$ if and only if $U \perp_{G^*} V | Z$.

Incremental associative Markov boundary algorithm (IAMB)

For each X_i do
  $Ad(X_i) := \varnothing$
  Repeat until no change occurs
    if there exists $X_j \notin Ad(X_i) \cup X_i$ such that $X_i \not\perp_p X_j | Ad(X_i)$ then
      $Ad(X_i) := Ad(X_i) \cup X_j$
  Repeat until no change occurs
    if there exists $X_j \in Ad(X_i)$ such that $X_i \perp_p X_j | Ad(X_i) \setminus X_j$ then
      $Ad(X_i) := Ad(X_i) \setminus X_j$

- **Exercise**. How would you perform score based structure learning for MNs ?

Contents

Thank you

732A96/TDDE15 Advanced Machine Learning

Hidden Markov Models

Jose M. Peña
IDA, Linköping University, Sweden

Lecture 5: Hidden Markov Models

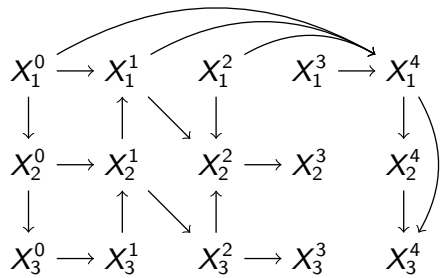
Contents

Literature

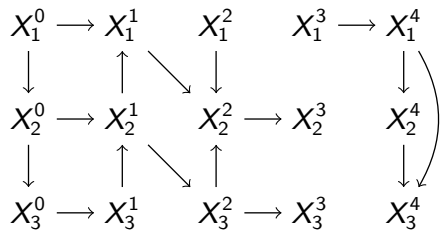
- Main source
  - Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006. Chapter 13.1-13.2.
- Additional source
  - Ghahramani, Z. An Introduction to Hidden Markov Models and Bayesian Networks. *International Journal of Pattern Recognition and Artificial Intelligence* 15, 9-42, 2001.

Dynamic Bayesian Networks

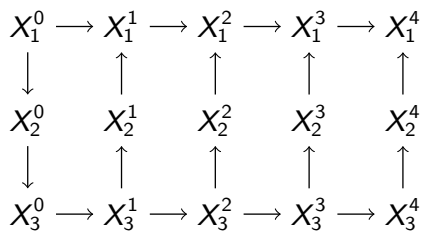
- To model **sequential data**, e.g. time series data.
- **Simplification**: Time is discretized in equal width intervals, i.e. $t = 0, 1, \ldots$
- Consider a finite set of discrete random variables $X^t = \{X_1^t, \ldots, X_n^t\}$ representing the state at time t of a system described by $X = \{X_1, \ldots, X_n\}$.
- A **dynamic Bayesian network** (DBN) is a BN over $X^{0:T} = \{X^0, \ldots, X^T\}$. Thus, it defines $p(x^{0:T})$.



- **Assumption**: The system is Markovian, i.e. $X^{t+1} \perp_p X^{0:t-1} | X^t$.

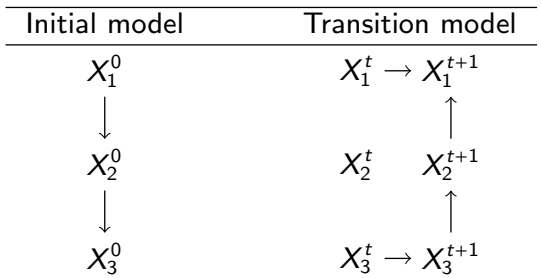


- **Assumption**: The system is stationary, i.e. $p(x^{t+1}|x^t) = p(x'|x)$.

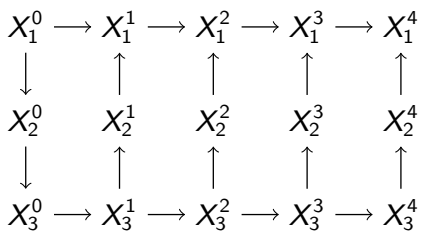


Dynamic Bayesian Networks

- Then, a DBN over $X^{0:T}$ can be defined as
  - a BN over X^0, and
  - a BN over $X^t \cup X^{t+1}$ where the nodes in X^t are parentless.



- DBN unrolled for $T = 4$.

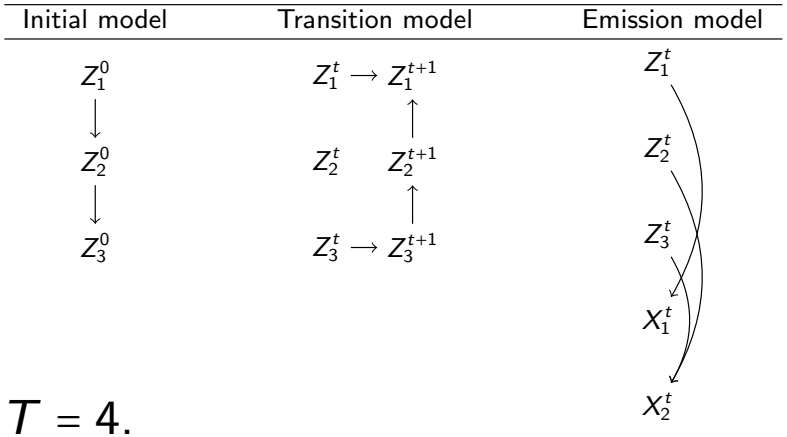


- The DBN defines

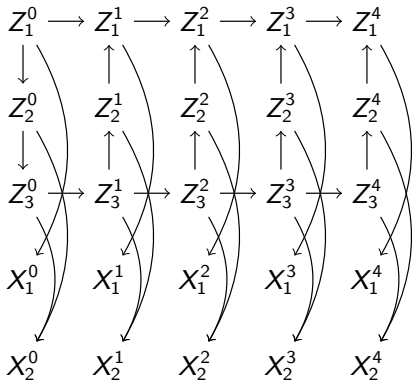
$$p(x^{0:T}) = p(x^0) \prod_{t=0}^{T-1} p(x^{t+1}|x^t) = [\prod_{i=1}^{n} p(x_i^0|pa_i^0)][\prod_{t=0}^{T-1} \prod_{i=1}^{n} p(x_i^{t+1}|pa_i^{t+1})]$$

Hidden Markov Models

- To overcome the **Markovian limitation** of DBNs, while keeping sparsity.
- A **hidden Markov model** (HMM) over $\{Z^{0:T}, X^{0:T}\}$ where $X^{0:T}$ are **observed** and $Z^{0:T}$ are **unobserved** consists of
  - a DBN over $Z^{0:T}$, and
  - a BN over $Z^t \cup X^t$ where the nodes in Z^t are parentless.



- HMM unrolled for $T = 4$.

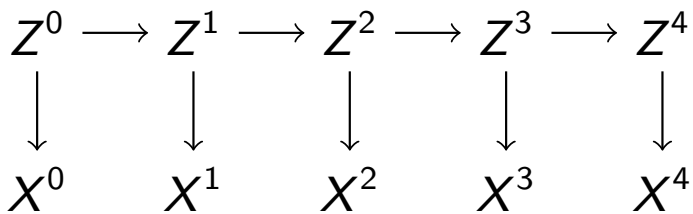


- A HMM is a DBN that defines

$$p(z^{0:T}, x^{0:T}) = p(z^0) \prod_{t=1}^{T-1} p(z^{t+1}|z^t) \prod_{t=0}^{T} p(x^t|z^t)$$

Hidden Markov Models: Lab

- 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.



- Where is the robot at time t ? I.e., compute the **filtered** distribution $p(z^t|x^{0:t})$ via the **forward-backward algorithm**.
- Where was the robot at time t ? I.e., compute the **smoothed** distribution $p(z^t|x^{0:T})$ via the **forward-backward algorithm**.
- Which is the most likely path that the robot followed ? I.e., run the **Viterbi algorithm**.

Hidden Markov Models: Forward-Backward Algorithm

- Smoothing:

$$
\begin{aligned}
p(z^t|x^{0:T}) &= \frac{p(x^{0:T}|z^t)p(z^t)}{p(x^{0:T})} \\
&= \frac{p(x^{0:t}|z^t)p(z^t)p(x^{t+1:T}|z^t)}{p(x^{0:T})} \text{ by } X^{0:t} \perp_p X^{t+1:T}|Z^t \\
&= \frac{p(x^{0:t}, z^t)p(x^{t+1:T}|z^t)}{p(x^{0:T})} = \frac{\alpha(z^t)\beta(z^t)}{\sum_{z^t}\alpha(z^t)\beta(z^t)}
\end{aligned}
$$

- Filtering:

$$
p(z^t|x^{0:t}) = \frac{\alpha(z^t)}{\sum_{z^t}\alpha(z^t)}
$$

Hidden Markov Models: Forward-Backward Algorithm

$$
\begin{aligned}
\boldsymbol{\alpha(z^t)} &= p(x^{0:t}, z^t) = p(x^{0:t}|z^t)p(z^t) = p(x^t|z^t)p(z^t)p(x^{0:t-1}|z^t) \text{ by } X^{0:t-1} \perp_p X^t | Z^t \\
&= p(x^t|z^t)p(x^{0:t-1}, z^t) = p(x^t|z^t) \sum_{z^{t-1}} p(x^{0:t-1}, z^t|z^{t-1})p(z^{t-1}) \\
&= p(x^t|z^t) \sum_{z^{t-1}} p(x^{0:t-1}|z^{t-1})p(z^t|z^{t-1})p(z^{t-1}) \text{ by } X^{0:t-1} \perp_p Z^t | Z^{t-1} \\
&= p(x^t|z^t) \sum_{z^{t-1}} p(x^{0:t-1}, z^{t-1})p(z^t|z^{t-1}) = p(x^t|z^t) \sum_{z^{t-1}} \boldsymbol{\alpha(z^{t-1})} p(z^t|z^{t-1}) \\
\alpha(z^0) &= p(x^0, z^0) = p(x^0|z^0)p(z^0)
\end{aligned}
$$

$$
\begin{aligned}
\boldsymbol{\beta(z^t)} &= p(x^{t+1:T}|z^t) = \sum_{z^{t+1}} p(x^{t+1:T}, z^{t+1}|z^t) = \sum_{z^{t+1}} p(x^{t+1:T}|z^{t+1}, z^t)p(z^{t+1}|z^t) \\
&= \sum_{z^{t+1}} p(x^{t+1:T}|z^{t+1})p(z^{t+1}|z^t) \text{ by } X^{t+1:T} \perp_p Z^t | Z^{t+1} \\
&= \sum_{z^{t+1}} p(x^{t+2:T}|z^{t+1})p(x^{t+1}|z^{t+1})p(z^{t+1}|z^t) \text{ by } X^{t+2:T} \perp_p X^{t+1} | Z^{t+1} \\
&= \sum_{z^{t+1}} \boldsymbol{\beta(z^{t+1})} p(x^{t+1}|z^{t+1})p(z^{t+1}|z^t) \\
\beta(z^T) &= 1 \text{ by } p(z^T|x^{0:T}) = \frac{\alpha(z^T)\beta(z^T)}{p(x^{0:T})} = p(z^T|x^{0:T})\beta(z^T)
\end{aligned}
$$

Hidden Markov Models: Forward-Backward Algorithm

FB algorithm

$\alpha(z^0) := p(x^0|z^0)p(z^0)$
For $t = 1, \ldots, T$ do
    $\alpha(z^t) := p(x^t|z^t) \sum_{z^{t-1}} \alpha(z^{t-1})p(z^t|z^{t-1})$
$\beta(z^T) := 1$
For $t = T-1, \ldots, 0$ do
    $\beta(z^t) := \sum_{z^{t+1}} \beta(z^{t+1})p(x^{t+1}|z^{t+1})p(z^{t+1}|z^t)$
Return $\alpha(z^0), \ldots, \alpha(z^T), \beta(z^0), \ldots, \beta(z^T)$

- Filtering: $p(z^t|x^{0:t}) = \frac{\alpha(z^t)}{\sum_{z^t} \alpha(z^t)}$.
- Smoothing: $p(z^t|x^{0:T}) = \frac{\alpha(z^t)\beta(z^t)}{\sum_{z^t} \alpha(z^t)\beta(z^t)}$.
- Note that the FB algorithm consists of two independent (parallelizable) steps.

Hidden Markov Models: Viterbi Algorithm

- To compute the most probable configuration for HMMs, i.e.

$$z_{\max}^{0:T} = \arg\max_{z^{0:T}} p(z^{0:T}|x^{0:T}) = \arg\max_{z^{0:T}} p(z^{0:T}, x^{0:T})$$

Viterbi algorithm

$\omega(z^0) := \log p(z^0) + \log p(x^0|z^0)$
For $t = 0, \ldots, T-1$ do
    $\omega(z^{t+1}) := \log p(x^{t+1}|z^{t+1}) + \max_{z^t} [\log p(z^{t+1}|z^t) + \omega(z^t)]$
    $\psi(z^{t+1}) := \arg\max_{z^t} [\log p(z^{t+1}|z^t) + \omega(z^t)]$
$z_{\max}^T = \arg\max_{z^T} \omega(z^T)$
For $t = T-1, \ldots, 0$ do
    $z_{\max}^t := \psi(z_{\max}^{t+1})$
Return $z_{\max}^{0:T}$

- **Exercise**. Prove that the Viterbi algorithm is correct.

Contents

- Dynamic Bayesian Networks
- Hidden Markov Models
  - Definition
  - Lab
  - Forward-Backward Algorithm
  - Viterbi Algorithm

Thank you

732A96/TDDE15 Advanced Machine Learning

Hidden Markov Models

Jose M. Peña
IDA, Linköping University, Sweden

Lecture 6: Autoregressive and Explicit-Duration Hidden Markov Models

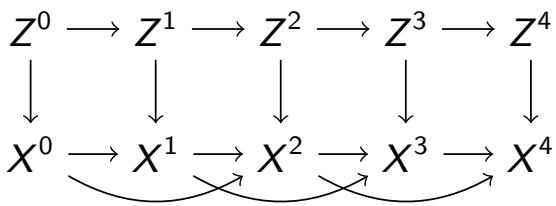
Contents

Literature

- Main source
  - Chiappa, S. Explicit-Duration Markov Switching Models. *Foundations and Trends in Machine Learning* 7, 803-886, 2014. Sections 1-3.3.
- Additional source
  - Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006. Chapter 13.1-13.2.

Autoregressive Hidden Markov Models: Definition

- To overcome the poor modeling of long range correlations in HMMs, by allowing $Pa(X^t) \subseteq \{Z^t, X^{0:t}\}$.



- Hereinafter, we focus on the simplest AR-HMM, i.e. $Pa(X^t) = \{Z^t, X^{t-1}\}$.

Autoregressive Hidden Markov Models: Forward-Backward Algorithm

- Smoothing:

$$
\begin{aligned}
p(z^t|x^{0:T}) &= \frac{p(x^{0:t-1}, x^{t+1:T}|z^t, x^t)p(z^t, x^t)}{p(x^{0:T})} \\
&= \frac{p(x^{0:t-1}|z^t, x^t)p(z^t, x^t)p(x^{t+1:T}|z^t, x^t)}{p(x^{0:T})} \quad \text{by } X^{0:t-1} \perp_p X^{t+1:T}|Z^t \cup X^t \\
&= \frac{p(x^{0:t}, z^t)p(x^{t+1:T}|z^t, x^t)}{p(x^{0:T})} = \frac{\alpha(z^t)\beta(z^t)}{\sum_{z^t}\alpha(z^t)\beta(z^t)}
\end{aligned}
$$

- Filtering:

$$
p(z^t|x^{0:t}) = \frac{\alpha(z^t)}{\sum_{z^t}\alpha(z^t)}
$$

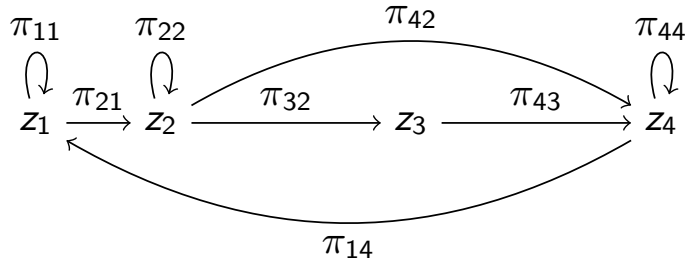
Autoregressive Hidden Markov Models: Forward-Backward Algorithm

$$
\begin{aligned}
\boldsymbol{\alpha(z^t)} &= p(x^t|z^t,x^{t-1})p(z^t,x^{t-1})p(x^{0:t-2}|z^t,x^{t-1}) \text{ by } X^{0:t-2} \perp_p X^t | Z^t \cup X^{t-1} \\
&= p(x^t|z^t,x^{t-1})p(x^{0:t-1},z^t) = p(x^t|z^t,x^{t-1}) \sum_{z^{t-1}} p(x^{0:t-1},z^t|z^{t-1})p(z^{t-1}) \\
&= p(x^t|z^t,x^{t-1}) \sum_{z^{t-1}} p(x^{0:t-1}|z^{t-1})p(z^t|z^{t-1})p(z^{t-1}) \text{ by } X^{0:t-1} \perp_p Z^t | Z^{t-1} \\
&= p(x^t|z^t,x^{t-1}) \sum_{z^{t-1}} p(x^{0:t-1},z^{t-1})p(z^t|z^{t-1}) \\
&= p(x^t|z^t,x^{t-1}) \sum_{z^{t-1}} \boldsymbol{\alpha(z^{t-1})} p(z^t|z^{t-1}) \text{ with } \alpha(z^0) = p(x^0|z^0)p(z^0) \\
\boldsymbol{\beta(z^t)} &= \sum_{z^{t+1}} p(x^{t+1:T},z^{t+1}|z^t,x^t) = \sum_{z^{t+1}} p(x^{t+1:T}|z^{t+1},z^t,x^t)p(z^{t+1}|z^t,x^t) \\
&= \sum_{z^{t+1}} p(x^{t+1:T}|z^{t+1},x^t)p(z^{t+1}|z^t) \text{ by } X^{t+1:T} \perp_p Z^t | Z^{t+1} \cup X^t \text{ and } X^t \perp_p Z^{t+1} | Z^t \\
&= \sum_{z^{t+1}} p(x^{t+2:T}|z^{t+1},x^{t+1})p(x^{t+1}|z^{t+1},x^t)p(z^{t+1}|z^t) \text{ by } X^{t+2:T} \perp_p X^t | Z^{t+1} \cup X^{t+1} \\
&= \sum_{z^{t+1}} \boldsymbol{\beta(z^{t+1})} p(x^{t+1}|z^{t+1},x^t)p(z^{t+1}|z^t) \\
\beta(z^T) &= 1 \text{ by } p(z^T|x^{0:T}) = \frac{\alpha(z^T)\beta(z^T)}{p(x^{0:T})} = p(z^T|x^{0:T})\beta(z^T)
\end{aligned}
$$

- The Viterbi algorithm can also be adapted to AR-HMMs.

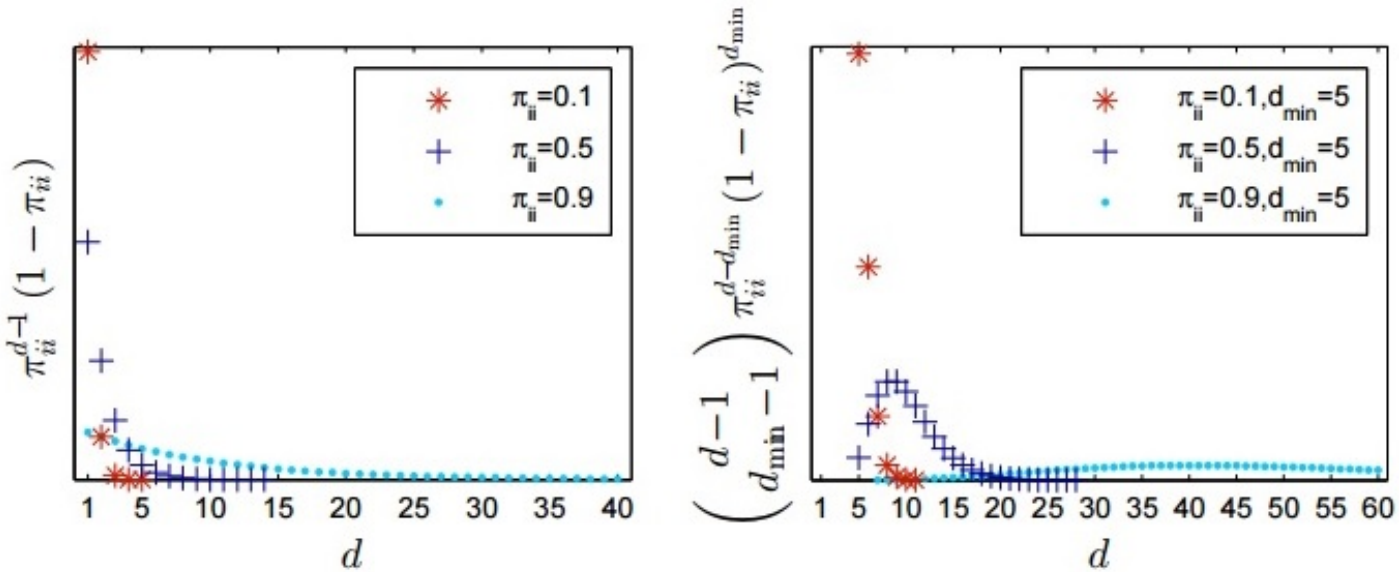
Explicit-Duration Hidden Markov Models: Definition

- To control the distribution of the number of time steps for which the HMM remains in a given state.
- Consider the following Markov chain over the states:



- The probability of remaining in state z_i for exactly $d-1$ time steps is

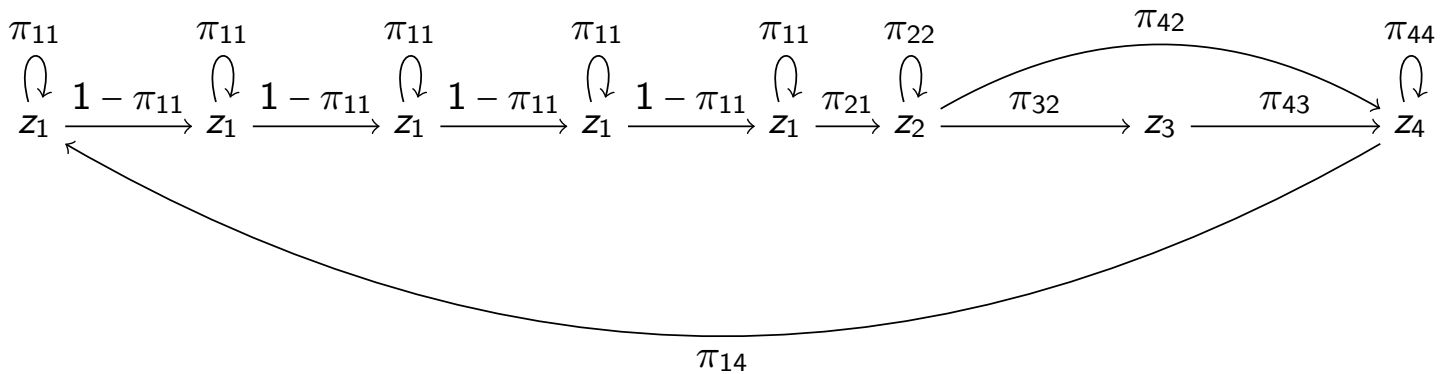
$$p(Z^{t+1:t+d-1} = z_i, Z^{t+d} \neq z_i | Z^t = z_i) = \pi_{ii}^{d-1}(1-\pi_{ii}) = \mathit{Geometric}(d; 1-\pi_{ii})$$



Source: Chiappa (2014).

Explicit-Duration Hidden Markov Models: Definition

- We can obtain a negative binomial distribution by imposing a minimum duration on the time spent in a state, and duplicating the corresponding observation model. For instance, when $d_{min} = 5$



- The probability of remaining in state z_i for exactly $d-1$ time steps is

$$\begin{aligned} p(Z^{t+1:t+d-1} &= z_i, Z^{t+d} \neq z_i | Z^t = z_i) = \binom{d-1}{d_{min}-1} \pi_{ii}^{d-d_{min}} (1-\pi_{ii})^{d_{min}} \\ &= NegativeBinomial(d; d_{min}, 1-\pi_{ii}) \end{aligned}$$

Explicit-Duration Hidden Markov Models: Decreasing

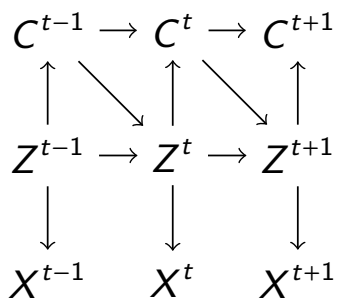
- The geometric and negative binomial cases define a segment duration distribution implicitly. We now define the segment duration distribution explicitly, by introducing auxiliary unobserved random variables.
- Specifically, let us use the random variables $C^{1:T}$ which take **decreasing** values within a segment, starting with the duration of the segment and ending with 1. Then

$$p(z_i^t | z_j^{t-1}, c^{t-1}) = \begin{cases} \pi_{ij} & \text{if } c^{t-1} = 1 \\ 1_{i=j} & \text{if } c^{t-1} > 1 \end{cases}$$

$$p(c^t | z^t, c^{t-1}) = \begin{cases} \rho(z^t, c^t) & \text{if } c^{t-1} = 1 \\ 1_{c^t = c^{t-1}-1} & \text{if } c^{t-1} > 1 \end{cases}$$

where $\rho(z^t, c^t)$ is the segment duration distribution.

- In graphical terms, we have



Explicit-Duration Hidden Markov Models: Increasing

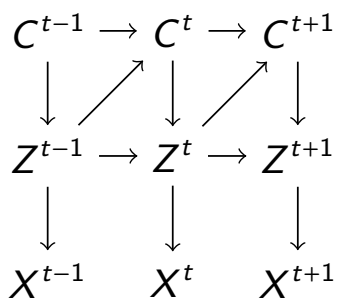
- Alternatively, we can use the random variables $C^{1:T}$ which take **increasing** values within a segment, starting 1 and ending with the duration of the segment. Then

$$p(z_i^t | z_j^{t-1}, c^t) = \begin{cases} \pi_{ij} & \text{if } c^t = 1 \\ 1_{i=j} & \text{if } c^t > 1 \end{cases}$$

$$p(c^t | z^{t-1}, c^{t-1}) = \begin{cases} 1 - \lambda(z^{t-1}, c^{t-1}) & \text{if } c^t = 1 \\ \lambda(z^{t-1}, c^{t-1}) & \text{if } c^t = c^{t-1} + 1 \end{cases}$$

  where $\lambda(z^{t-1}, c^{t-1})$ is the probability of the segment continuing.

- In graphical terms, we have



Explicit-Duration Hidden Markov Models: Decreasing-Duration

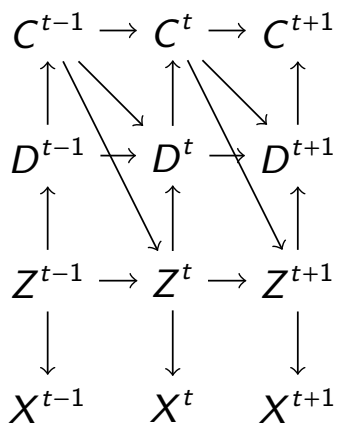
- Alternatively, we can use the random variables $C^{1:T}$ which take **decreasing** values within a segment, starting with the duration of the segment and ending with 1. And the random variables $D^{1:T}$ which indicate the **duration** of the current segment. Then

$$p(z_i^t | z_j^{t-1}, c^{t-1}) = \begin{cases} \pi_{ij} & \text{if } c^{t-1} = 1 \\ 1_{i=j} & \text{if } c^{t-1} > 1 \end{cases}$$

$$p(c^t | c^{t-1}, d^t) = \begin{cases} 1_{c^t = d^t} & \text{if } c^{t-1} = 1 \\ 1_{c^t = c^{t-1} - 1} & \text{if } c^{t-1} > 1 \end{cases}$$

$$p(d^t | z^t, c^{t-1}, d^{t-1}) = \begin{cases} \rho(z^t, d^t) & \text{if } c^{t-1} = 1 \\ 1_{d^t = d^{t-1}} & \text{if } c^{t-1} > 1 \end{cases}$$

  where $\rho(z^t, d^t)$ is the segment duration distribution.
- In graphical terms, we have



Explicit-Duration Hidden Markov Models: Increasing-Duration

- **Exercise**. Derive the transition model for the increasing-duration case.
- Relatively easy to adapt the FB and Viterbi algorithms to explicit-duration HMMs.

Contents

Thank you

732A96/TDDE15 Advanced Machine Learning

Reinforcement Learning

Jose M. Peña
IDA, Linköping University, Sweden

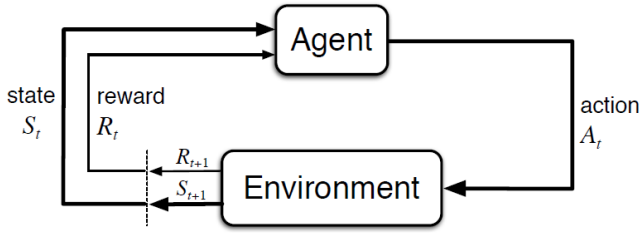
Lectures 8: Q-Learning Algorithm

Contents

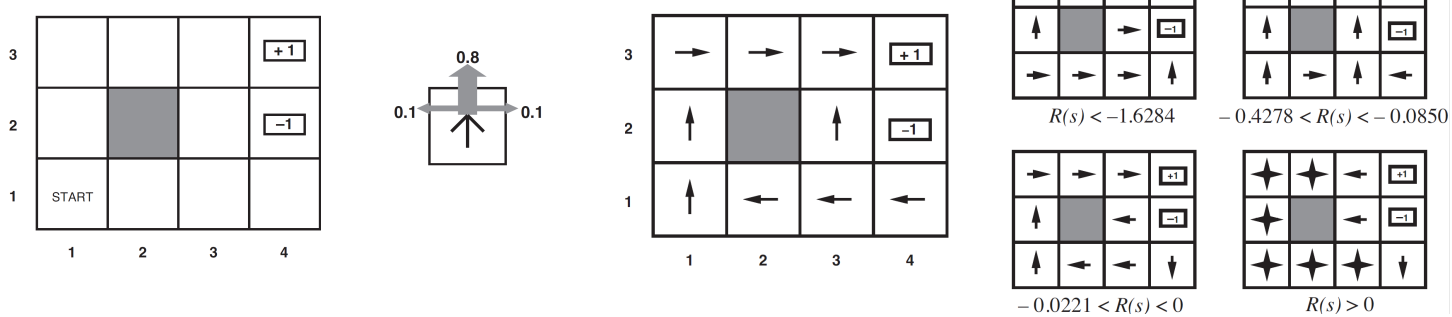
Literature

- Main source
  - Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. The MIT Press, 2018. Chapters 1-7.
- Additional source
  - Russel, S. and Norvig, P. *Artifical Intelligence: A Modern Approach*. Pearson, 2010. Chapters 16, 17 and 21.

Learning through Interaction

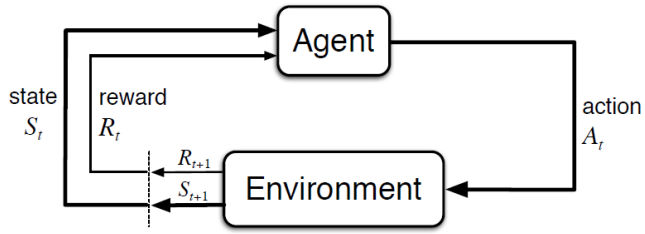


- **Agent**: The learner and decision maker.
- **Environment**: The agent interacts with it.
  - **State**: State of the agent and the environment.
  - **Action**: The agent decides the next action on the basis of the current state.
  - **Reward**: Numerical response to the action chosen by the agent. The agent aims to learn how to act so as to maximize the cumulative reward.
  - **Trajectory**: $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, S_3, \ldots$
- **Policy**: Probability of doing an action in a state. The agent acts according to it. The agent aims to learn an optimal one.
- Example: A robot moves with probability 0.8 in the intended direction, and at the right angles of it otherwise. The reward for non-terminal states is $R(s) = -0.04$. All this is unknown to the robot. Optimal policies shown.



- RL is neither supervised nor unsupervised learning.

Markov Decision Processes



- We assume that the agent-environment interaction follows a finite **Markov decision process**:
  - Finite sets of states, actions and rewards. Fully observable state.
  - Markovian and stationary transition model:

$$p(s_t, r_t | s_{0:t-1}, a_{0:t-1}, r_{1:t-1}) = p(s_t, r_t | s_{t-1}, a_{t-1}) = p(s', r | s, a).$$

- The **transition model** is typically unknown to the agent. Note the randomness of the next state and reward.
- The objective of the agent is to learn a policy $\pi(a|s)$ that maximizes the **expected discounted return**:

$$E_\pi[G_t] = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\right] = E_\pi[R_{t+1} + \gamma G_{t+1}]$$

where $0 < \gamma \leq 1$ describes our preference between present and future rewards. Note the infinite horizon. However, the expectation is finite if $\gamma < 1$. For episodic tasks, $\gamma = 1$ and $R_{t+k+1} = 0$ for all $t + k + 1 > T$.

Markov Decision Processes

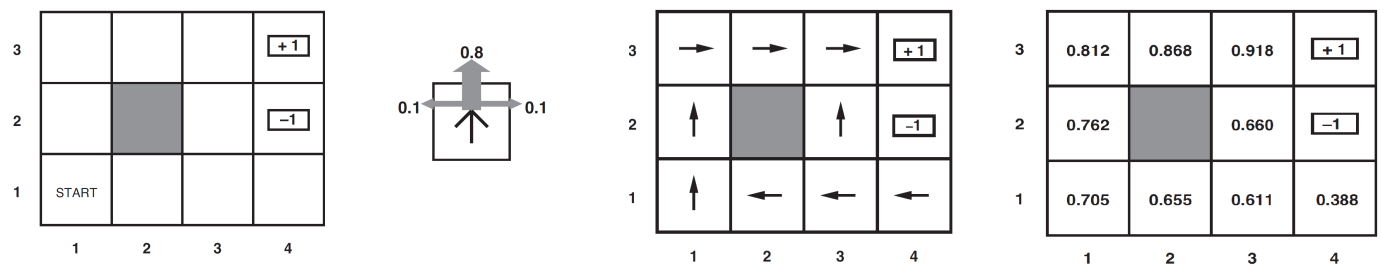
- The **state value** function $v_\pi(s)$ is the expected return of following policy π starting from state s:

$$v_\pi(s) = E_\pi[G_t|S_t = s] = \sum_a \pi(a|s) E_\pi[G_t|S_t = s, A_t = a] = \sum_a \pi(a|s) q_\pi(s, a).$$

- The **action value** function $q_\pi(s, a)$ is the expected return of doing action a in state s and then following policy π:

$$q_\pi(s, a) = E_\pi[G_t|S_t = s, A_t = a] = E_\pi[R_{t+1} + \gamma G_{t+1}|s, a] = \sum_{s',r} p(s', r|s, a)(r + \gamma v_\pi(s')).$$

- Example: Environment, policy and state values.



- We can define the objective of the agent as learning a policy π_* such that

$$v_*(s) \geq v_\pi(s) \text{ for all } \pi, s.$$

For MDPs, there is always **at least one** such optimal policy.

Bellman Equations

- The state value function satisfies a recursive relationship known as **Bellman equation**:

$$v_\pi(s) = \sum_a \pi(a|s) q_\pi(s,a) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)(r + \gamma v_\pi(s')).$$

- Moreover, v_π is the **unique solution** to the equations. Note that there are as many equations as unknowns. Since the equations are linear, they can be solved by linear algebra methods in $O(n^3)$. But this requires knowing the transition model.
- Likewise for the action value function:

$$q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)(r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s',a')).$$

- The Bellman equations of an **optimal policy** are

$$v_*(s) = \max_a \sum_{s',r} p(s',r|s,a)(r + \gamma v_*(s'))$$

and

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)(r + \gamma \max_{a'} q_*(s',a')).$$

- As before, v_* and q_* are the **unique solutions** to these equations. Note that the equations are now non-linear due to the max operator and, thus, harder to solve. Again, this requires knowing the transition model.

Bellman Equations

- Once we have v_* or q_*, it is easy to determine an **optimal policy**:

$$\pi_*(a|s) = \arg\max_a q_*(s, a)$$

or

$$\pi_*(a|s) = \arg\max_a \sum_{s', r} p(s', r|s, a)(r + \gamma v_*(s')).$$

- Note that the optimal policy is **deterministic**. So, we can consider only deterministic policies without loss of generality, i.e. $\pi(s)$ instead of $\pi(a|s)$.

Value Iteration

- We can avoid solving the Bellman equations for the state values of an optimal policy by turning them into update rules.

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $\quad v \leftarrow V(s)$
| $\quad V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
| $\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$

- VI converges asymptotically for $\gamma < 1$. Since the Bellman optimality equations have a unique solution, VI converges to v_*.
- VI still requires knowing the transition model.

Policy Iteration

- Policy evaluation: Turn the ordinary Bellman equations into update rules.

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s)) \left[r + \gamma V(s') \right]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma V(s') \right]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Policy Iteration

- Theorem: If $q_\pi(s, \pi'(s)) \geq q_\pi(s, \pi(s))$ for all s, then $v_{\pi'}(s) \geq v_\pi(s)$ for all s. Thus, we can modify π into a better policy π' by doing

$$\pi'(s) = \arg\max_a q_\pi(s, a) \text{ for all } s.$$

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\left[r + \gamma V(s')\right]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Policy Iteration

- PI terminates since each iteration improves the policy and there is a finite number of policies. When PI halts, the Bellman optimality equations hold and, thus, π is optimal. Again, PI requires knowing the transition model.

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r \,|\, s, \pi(s)) \big[r + \gamma V(s')\big]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \,|\, s, a) \big[r + \gamma V(s')\big]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Q-Learning

- If the transition model were so that s and a are always followed by s' and r, then $q_*(s,a) = r + \gamma \max_{a'} q_*(s',a')$ by the Bellman optimality equation and, thus, $0 = r + \gamma \max_{a'} q_*(s',a') - q_*(s,a)$. We can try to enforce this constraint by executing π one step from s and a and, then, updating the estimate of $q_*(s,a)$ as

$$q_*(s,a) \leftarrow q_*(s,a) + \alpha(r + \gamma \max_{a'} q_*(s',a') - q_*(s,a)).$$

where $\alpha > 0$ is the learning rate.

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal,\cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$
  until S is terminal

Q-Learning

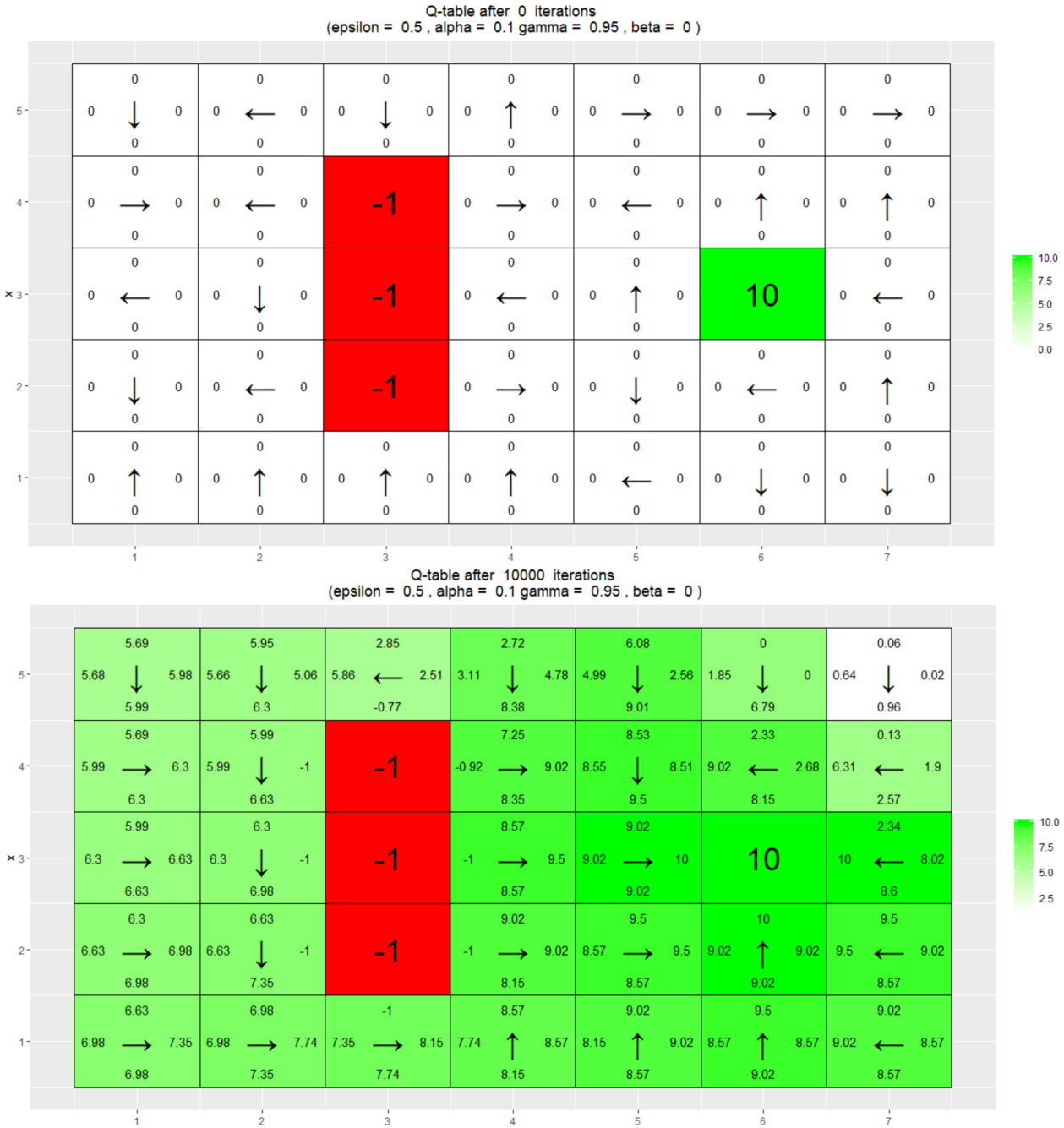
Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

- Q-learning converges asymptotically if e.g. $\alpha(t) = O(1/N(s, t))$.
- Q-learning converges asymptotically to π_* if e.g. an ϵ-**greedy** policy is used to keep updating all the state-action pairs: Choose the action with maximal estimated value with probability $1 - \epsilon$, and a random one with probability ϵ.
- Q-learning also works for stochastic transition models, since the number of times that s and a are followed by s' and r in the sampled episodes is proportional to the transition probability.
- Q-learning does **not** require knowing the transition model.

Example: Grid Worlds



Summary

- Learning through Interaction
- Markov Decision Processes
- Bellman Equations
- Value Iteration
- Policy Iteration
- Q-Learning
- Example: Grid Worlds
- Interested in more ? Check out AlphaGo - The Movie.

Thank you

732A96/TDDE15 Advanced Machine Learning

Reinforcement Learning

Jose M. Peña
IDA, Linköping University, Sweden

Lectures 9: REINFORCE and Deep Q-Learning Algorithms

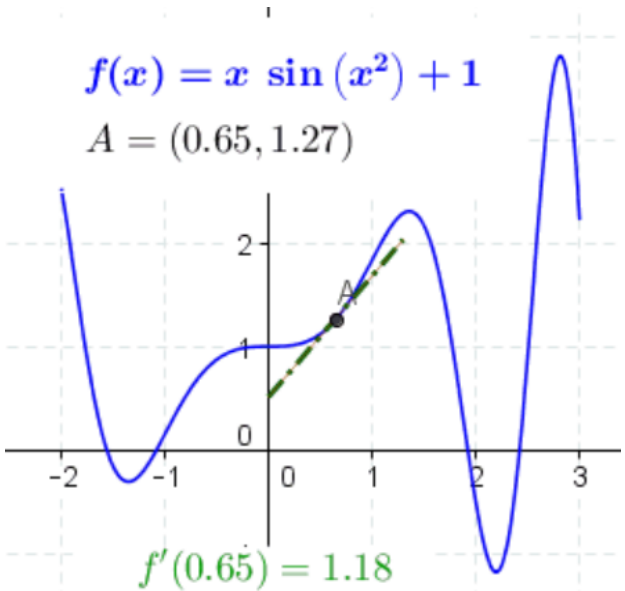
Contents

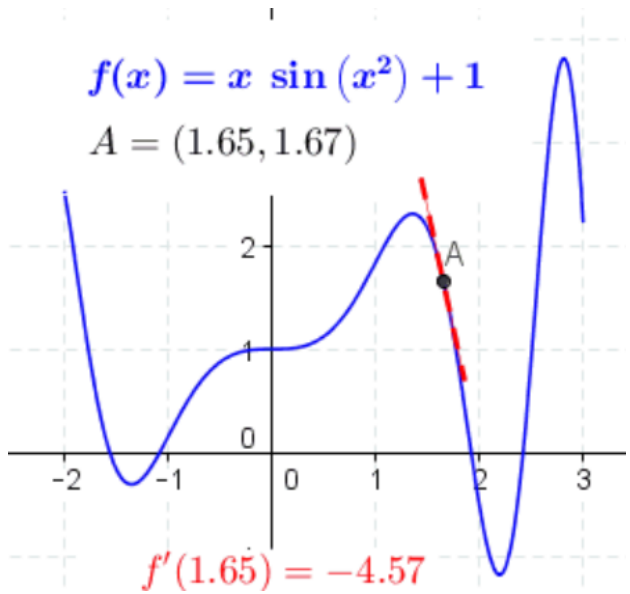
Literature

- Main source
  - Sutton, R. S. and Barto, A. G. Reinforcement Learning: An Introduction. The MIT Press, 2018. Chapters 13 and 16.
  - Mnih, V. et al. Playing Atari with Deep Reinforcement Learning. *NIPS Deep Learning Workshop*, 2013.
- Additional source
  - Russel, S. and Norvig, P. *Artifical Intelligence: A Modern Approach*. Pearson, 2010. Chapters 16, 17 and 21.
  - Mnih, V. et al. Human-level Control through Deep Reinforcement Learning. *Nature 518*, 529–533, 2015.

REINFORCE

- Consider learning a **parameterized policy** $\pi(a|s,\theta)$ to select actions without consulting state or action values.
- Advantages:
  - The policy function may be simpler to learn than the state and action value functions.
  - Possibility to inject prior knowledge through the parameterization chosen.
  - Possibility to model deterministic and stochastic policies. The latter may be interesting in problems with imperfect knowledge, e.g. bluffing in poker. Learning is also easier as the gradient is smoother.
- Choose the parameter values that maximize the following function for **episodic** tasks: $J(\theta) = v_{\pi_\theta}(S_0)$ where S_0 is the initial state. It can also be adapted to continuing tasks.
- Stochastic **gradient ascent**: $\theta^{i+1} \leftarrow \theta^i + \alpha \nabla_{\theta^i} J(\theta^i)$.





REINFORCE

- Theorem:

$$\nabla_\theta J(\theta) = \nabla_\theta v_{\pi_\theta}(S_0) \propto E_{\pi_\theta}\Big[\sum_a \gamma^t q_\pi(S_t, a)\nabla_\theta \pi(a|S_t, \theta)\Big] \text{ at time } t$$

  and thus

$$\nabla_\theta J(\theta) \propto E_{\pi_\theta}\Big[\sum_a \pi(a|S_t,\theta)\gamma^t q_\pi(S_t,a)\frac{\nabla_\theta \pi(a|S_t,\theta)}{\pi(a|S_t,\theta)}\Big] = E_{\pi_\theta}\Big[\gamma^t q_\pi(S_t,A_t)\frac{\nabla_\theta \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)}\Big]$$

$$= E_{\pi_\theta}\Big[\gamma^t G_t \frac{\nabla_\theta \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)}\Big] = E_{\pi_\theta}[\gamma^t G_t \nabla_\theta \ln \pi(A_t|S_t,\theta)].$$

- All this results in the REINFORCE algorithm, which asymptotically converges to a local optimum of $J(\theta)$.

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ $\qquad (G_t)$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta})$

REINFORCE

- REINFORCE has one major disadvantage: It updates the policy only **at the end of each episode** and, thus, it is not suitable for incremental online learning and may show slow convergence and high variance.
- A particularly interesting advantage of REINFORCE is that updating the policy parameters may change the policy for every action-state pair, and not only for the pairs visited, i.e. **generalization**.
- Another interesting feature of REINFORCE is that it can be applied to **continuous state and action spaces** by choosing an appropriate parametric policy model, e.g. $\mathcal{N}(\mu s, \sigma)$:

$$\pi(a|s,\theta) = \pi(a|s,\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\Big(-\frac{(a-\mu s)^2}{2\sigma^2}\Big)$$

which implies that

$$\nabla_\mu \ln \pi(a|s,\mu,\sigma) = \frac{a-\mu s}{\sigma^2} s$$

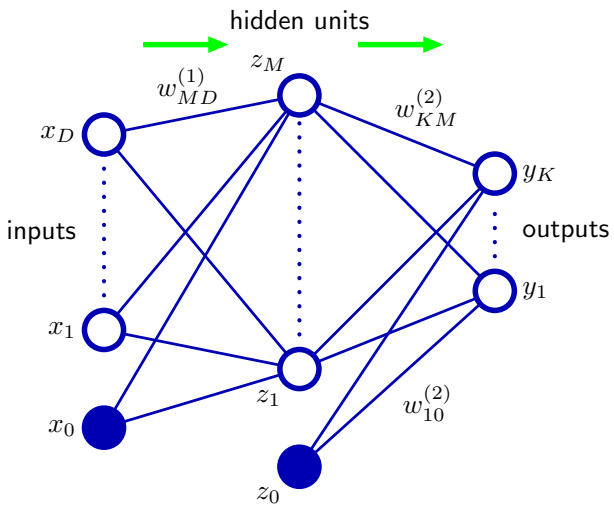
and

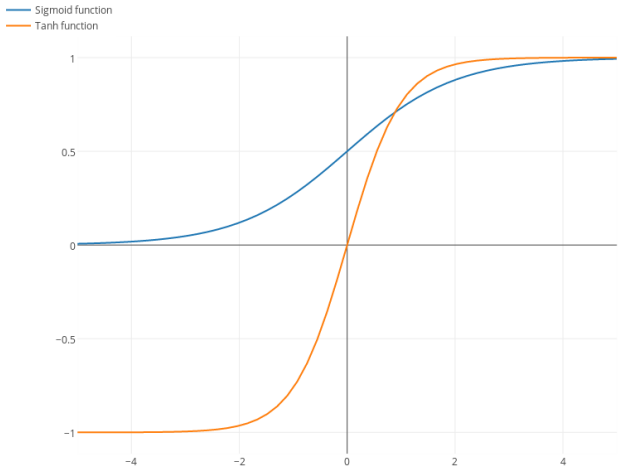
$$\nabla_\sigma \ln \pi(a|s,\mu,\sigma) = \Big(\frac{(a-\mu s)^2}{\sigma^2} - 1\Big) s$$

and, thus, the REINFORCE update has a simple form.

- For discrete state and action spaces, we can easily do better than this as we show next.

Neural Networks





- Consider a multiclass classification problem, i.e. $C \in \{1, 2, \ldots, K\}$.
- Activations: $a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$ for all $j = 1, \ldots, M$.
- Hidden units and activation function: $z_j = h(a_j)$ for all $j = 1, \ldots, M$.
- Output activations: $a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}$ for all $k = 1, \ldots, K$.
- Output : $y_k(\boldsymbol{x}) = a_k$ for all $k = 1, \ldots, K$.
- Two-layer NN: $y_k(\boldsymbol{x}) = \sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)}$.
- Class probabilities: $p(C = k|\boldsymbol{x}) = \frac{\exp\left(y_k(\boldsymbol{x})\right)}{\sum_{k=1}^{K} \exp\left(y_k(\boldsymbol{x})\right)}$.
- All the previous is generalizable to several hidden layers.

Neural Networks

- Consider some training data $\{(\boldsymbol{x}_n, c_n)\}$ of size N. Consider minimizing the cross-entropy or negative log-likelihood loss function:

$$L(\boldsymbol{w}) = \sum_{n=1}^{N} L_n(\boldsymbol{w}) = -\sum_{n=1}^{N} \log p(c_n|\boldsymbol{x}_n, \boldsymbol{w}).$$

- The weight space is highly multimodal and, thus, we have to resort to approximate iterative methods to minimize the previous expression.
- Stochastic gradient descent:

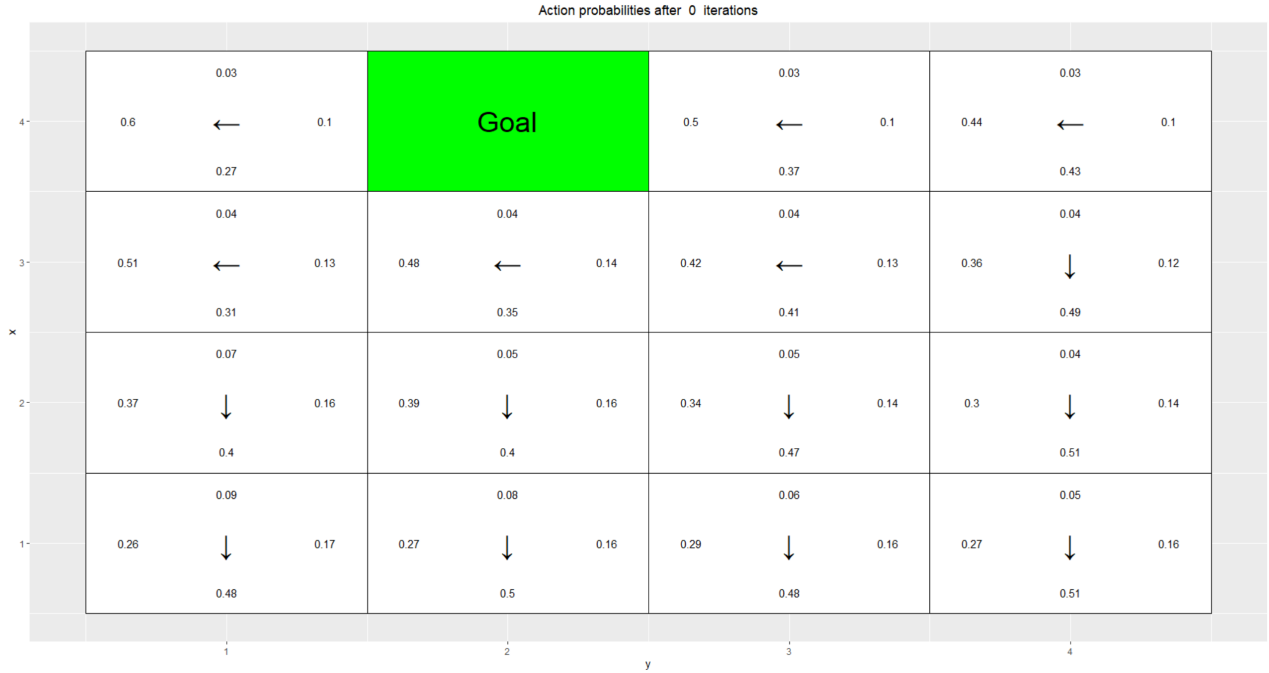
$$\boldsymbol{w}^{i+1} = \boldsymbol{w}^{i} - \alpha \nabla_{\boldsymbol{w}^i} L_n(\boldsymbol{w}^{i}) = \boldsymbol{w}^{i} + \alpha \nabla_{\boldsymbol{w}^i} \log p(c_n|\boldsymbol{x}_n, \boldsymbol{w}^{i})$$

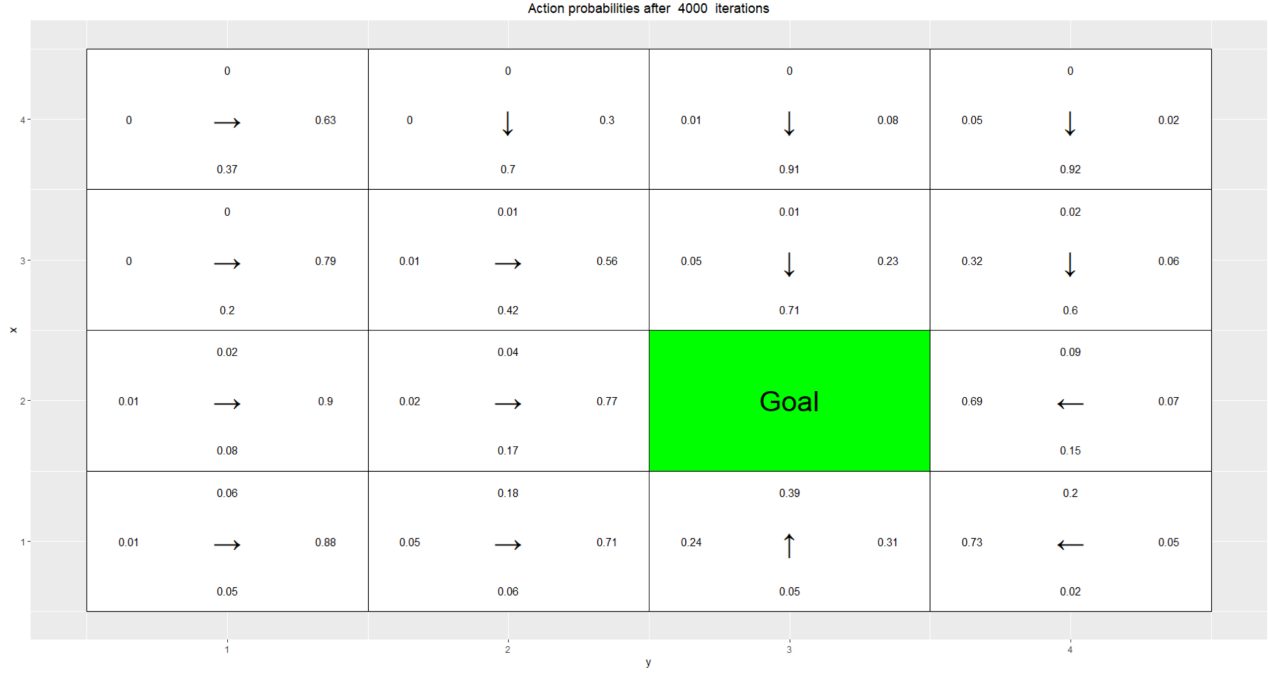
  where n is chosen randomly, and $\nabla_{\boldsymbol{w}^i} L_n(\boldsymbol{w}^{i})$ can be computed efficiently via **backpropagation**.
- Note the similarities between the update above and the REINFORCE update:

$$\theta^{i+1} = \theta^{i} + \alpha \gamma^{t} G \nabla_{\theta^i} \ln \pi(A_t|S_t, \theta^{i}).$$

- Therefore, REINFORCE is straightforward to implement when a NN is used to represent the parameterized policy. We only need to set the sample weights $\gamma^t G$. In other words, the policy can be seen as a classifier whose training involves certain peculiarities: Incremental online, and **delayed labels** via sample weights.

Example: Grid Worlds





Neural Networks

- Now, consider a unidimensional regression problem. Consider also some training data $\{(\boldsymbol{x}_n, c_n)\}$ of size N. Consider minimizing the mean squared-error loss function:

$$L(\boldsymbol{w}) = \sum_{n=1}^{N} L_n(\boldsymbol{w}) = \sum_{n=1}^{N} \frac{1}{2}(c_n - y(\boldsymbol{x}_n))^2.$$

- The above implies the following stochastic gradient descent update:

$$\boldsymbol{w}^{i+1} = \boldsymbol{w}^i - \alpha \nabla_{\boldsymbol{w}^i} L_n(\boldsymbol{w}^i) = \boldsymbol{w}^i + \alpha(c_n - y(\boldsymbol{x}_n)) \nabla_{\boldsymbol{w}^i} y(\boldsymbol{x}_n)$$

  where n is chosen randomly, and $\nabla_{\boldsymbol{w}^i} y(\boldsymbol{x}_n)$ can be computed efficiently via **backpropagation**.

Deep Q-Learning

- If the transition model were so that s and a are always followed by s' and r, then $q_*(s,a) = r + \gamma \max_{a'} q_*(s',a')$ by the Bellman optimality equation and, thus, $0 = r + \gamma \max_{a'} q_*(s',a') - q_*(s,a)$. We can try to enforce this constraint by executing π one step from s and a and, then, updating the estimate of $q_*(s,a)$ as

$$q_*(s,a) \leftarrow q_*(s,a) + \alpha(r + \gamma \max_{a'} q_*(s',a') - q_*(s,a)).$$

  where $\alpha > 0$ is the learning rate.
- The reasoning above also applies to stochastic transition models, since the number of times that s and a are followed by s' and r in the sampled episodes is proportional to the transition probability.

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal,\cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$
  until S is terminal

Deep Q-Learning

- Q-learning has a major disadvantage: Representing the action value function $q(s, a)$ as a look-up table is not feasible for **large state and/or action spaces**, due to storage space and time to convergence.
- A solution is to represent it as a **parameterized function** $q(s, a|\theta)$. This brings advantages such as fewer parameters than table entries and thus easier to learn and reach convergence, less storage space, **generalization to unvisited state-action pairs**, etc.
- It also comes with the challenge of selecting an appropriate parameterized function: It has to be expressive and allow for incremental online training. A NN is typically used.

Deep Q-Learning

- For the i-th iteration (i.e., episode step), the NN is trained by minimizing the mean squared-error loss function:

$$L(\theta^i) = E_{s,a\sim\rho(s,a)}[(y^i - q(s,a|\theta^i))^2]$$

where $\rho(s,a)$ is the behaviour distribution induced by ϵ-greedy, and

$$y^i = E_{s'\sim p(s'|s,a)}[r + \gamma \max_{a'} q(s',a'|\theta^{i-1})|s,a].$$

- The gradient is given by

$$\nabla_{\theta^i} L(\theta^i) = -E_{s,a\sim\rho(s,a);s'\sim p(s'|s,a)}[(r+\gamma \max_{a'} q(s',a'|\theta^{i-1}) - q(s,a|\theta^i))\nabla_{\theta^i} q(s,a|\theta^i)]$$

and stochastic gradient descent is typically considered:

$$\theta^{i+1} = \theta^i + \alpha(r + \gamma \max_{a'} q(s',a'|\theta^{i-1}) - q(s,a|\theta^i))\nabla_{\theta^i} q(s,a|\theta^i).$$

- Note the similarities between the update above and the NN update:

$$\boldsymbol{w}^{i+1} = \boldsymbol{w}^i + \alpha(c_n - y(\boldsymbol{x}_n))\nabla_{\boldsymbol{w}^i} y(\boldsymbol{x}_n)$$

which implies that deep Q-learning is straightforward to implement when a NN is used to represent the parameterized state value function. In other words, the action value function can be seen as a regressor whose training involves certain peculiarities: Incremental online, and **delayed targets** via bootstrapping on the model from the previous iteration.

Deep Q-Learning

Deep Q-learning with experience replay for estimating $\pi \approx \pi_*$

Algorithm parameters: small $\epsilon > 0$
Initialize $Q(S, A|\theta)$ with random weights, and initialize $\mathcal{D}$ with N random experiences

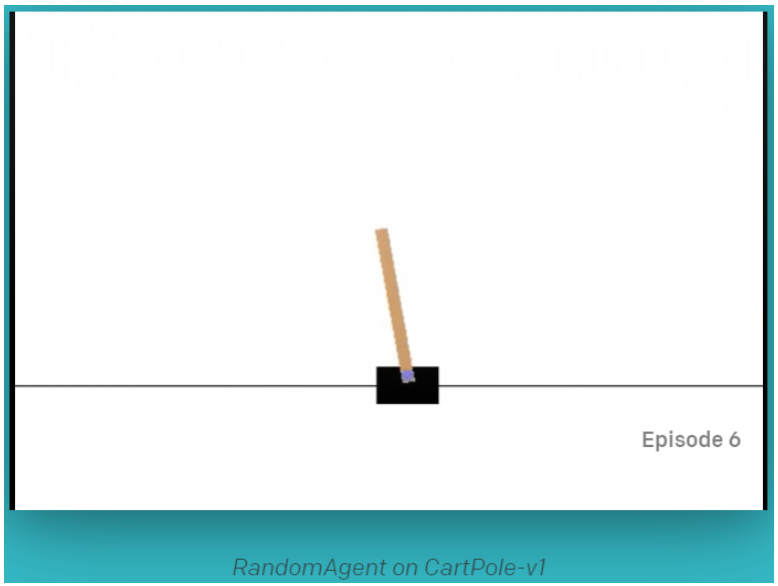
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observed R, S'
    Store transition (S, A, R, S') in $\mathcal{D}$
    Sample random minibatch of transitions (S_j, A_j, R_j, S'_j) from $\mathcal{D}$

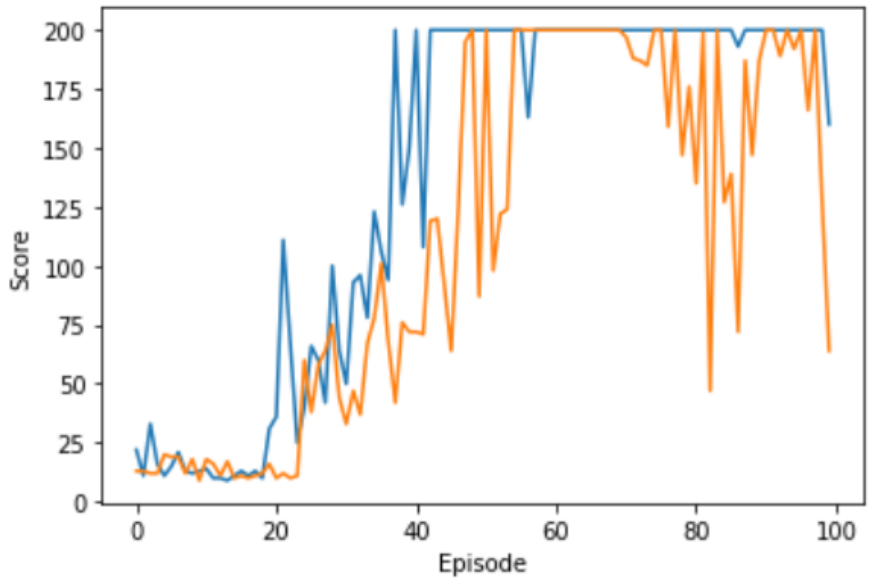
$$Y_j \leftarrow \begin{cases} R_j & \text{for terminal } S'_j \\ R_j + \gamma \max_a Q(S'_j, a|\theta) & \text{for non-terminal } S'_j \end{cases}$$

    Perform a gradient descent step on $(Y_j - Q(S_j, A_j|\theta))^2$
    $S \leftarrow S'$
  until S' is terminal

- Learning a parameterized state value function may resemble supervised learning but the learning data is **not i.i.d.** due to highly correlated states and changing agent behaviour. Experience replay alleviates these problems by smoothing the training distribution over many past behaviours.
- All action values at once: $S \rightarrow NN \rightarrow (Q(S, A_1), \ldots, Q(S, A_k))$.

Example: CartPole





- Open AI's gym
- PilcoLearner

Summary

- REINFORCE
- Neural Networks
- Example: Grid Worlds
- Deep Q-Learning
- Example: CartPole
- Interested in more ? Check out AlphaGo - The Movie.

Thank you

732A96/TDDE15 Advanced Machine Learning

Gaussian Process Regression and Classification

Jose M. Peña
IDA, Linköping University, Sweden

Lectures 9: Gaussian Process Regression

Contents

Literature

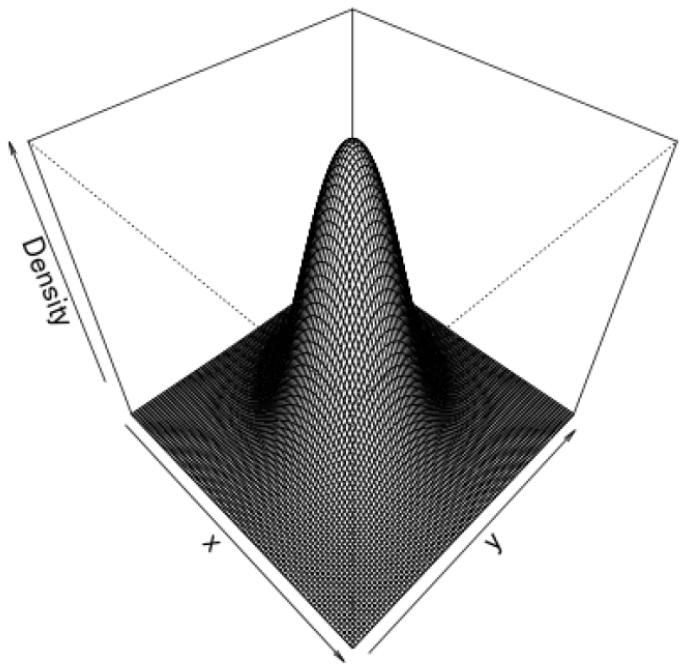
- Main source
  - Rasmussen, C. E. and Williams, K. I. *Gaussian Processes for Machine Learning*. MIT Press, 2006. Chapters 2.1-2.5.
- Additional source
  - Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006. Chapters 6.4.1-6.4.2.

Gaussian Distribution

- Density function of the Gaussian (a.k.a normal) distribution for a D-dimensional random variable $\boldsymbol{X}$:

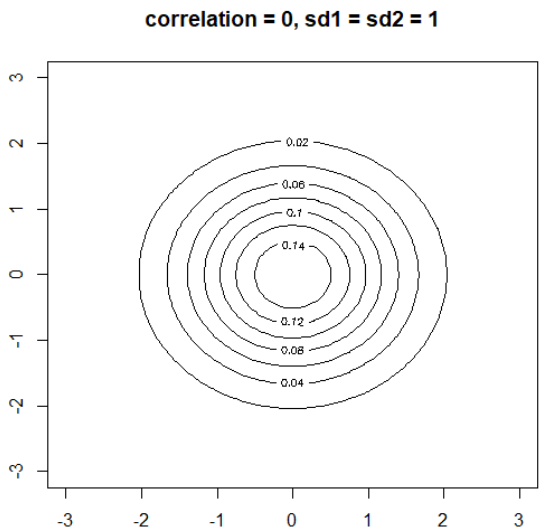
$$\mathcal{N}(\boldsymbol{x}; \mu, \Sigma) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma|^{1/2}} \exp\left\{ -\frac{1}{2}(\boldsymbol{x} - \mu)^T \Sigma^{-1} (\boldsymbol{x} - \mu) \right\}$$

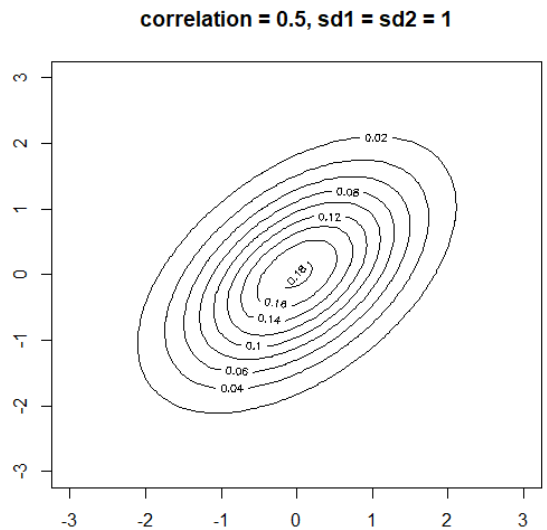
- Recall that $E[\boldsymbol{X}] = \mu$ and $cov(\boldsymbol{X}) = \Sigma$.

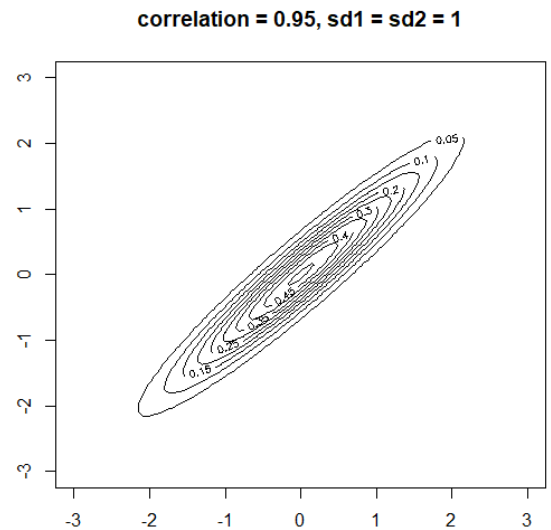


Gaussian Distribution

- Example: $\mathcal{N}(x_1, x_2; \mu, \Sigma)$ with $\Sigma = \begin{bmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{bmatrix}$.







Gaussian Distribution

- Recall that if

$$p(x) = \mathcal{N}(x; \mu, \Lambda^{-1})$$
$$p(y|x) = \mathcal{N}(y; Ax + B, L^{-1})$$

then

$$p(x, y) = \mathcal{N}(x, y; (\mu, A\mu + B), R^{-1})$$

where

$$R^{-1} = \begin{pmatrix} \Lambda^{-1} & \Lambda^{-1}A^T \\ A\Lambda^{-1} & L^{-1} + A\Lambda^{-1}A^T \end{pmatrix}.$$

- Recall also that if $p(x) = \mathcal{N}(x; \mu, \Sigma)$ and $\Lambda = \Sigma^{-1}$ and

$$x = (x_a, x_b)^T \qquad \mu = (\mu_a, \mu_b)^T$$
$$\Sigma = \begin{pmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{pmatrix} \qquad \Lambda = \begin{pmatrix} \Lambda_{aa} & \Lambda_{ab} \\ \Lambda_{ba} & \Lambda_{bb} \end{pmatrix}$$

then

$$p(x_a) = \mathcal{N}(x_a; \mu_a, \Sigma_{aa})$$
$$p(x_a|x_b) = \mathcal{N}(x_a; \mu_{a|b}, \Lambda_{aa}^{-1}) \quad \text{where } \mu_{a|b} = \mu_a - \Lambda_{aa}^{-1}\Lambda_{ab}(x_b - \mu_b) \text{ or}$$
$$p(x_a|x_b) = \mathcal{N}(x_a; \mu_{a|b}, \Sigma_{a|b}) \quad \text{where } \mu_{a|b} = \mu_a + \Sigma_{ab}\Sigma_{bb}^{-1}(x_b - \mu_b)$$
$$\text{and } \Sigma_{a|b} = \Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba}.$$

Linear Regression

- Training data: $\mathcal{D} = \{(\boldsymbol{x}_i, y_i) | i = 1, \ldots, n\} = (X, \boldsymbol{y})$.
- Deterministic function: $f(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{w}$.
- Additive noisy observations: $y = f(\boldsymbol{x}) + \epsilon$.
- Gaussian noise: $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$.
- Likelihood function:
  $p(\boldsymbol{y}|X, \boldsymbol{w}) = \mathcal{N}(X^T \boldsymbol{w}, \sigma_n^2 I) \propto \exp\left\{ -\frac{1}{2\sigma_n^2} \|\boldsymbol{y} - X^T \boldsymbol{w}\|^2 \right\}$.
- To obtain $\boldsymbol{w}^{ML}$,
  - take the derivative of the log lik function wrt $\boldsymbol{w}$, and
  - set it to zero, and
  - solve to obtain $\boldsymbol{w}^{ML} = (XX^T)^{-1} X\boldsymbol{y}$.
- Minimizing the least squared error (i.e., $\frac{1}{2} \sum_{i=1}^{n} (y_i - \boldsymbol{x}_i^T \boldsymbol{w})^2$) gives the same result. This justifies the use of LSE.

Bayesian Linear Regression

- Prior distribution: $\boldsymbol{w} \sim \mathcal{N}(0, \Sigma_p)$, e.g. ridge regression $\Sigma_p = \alpha^{-1} I$.
- Posterior distribution:

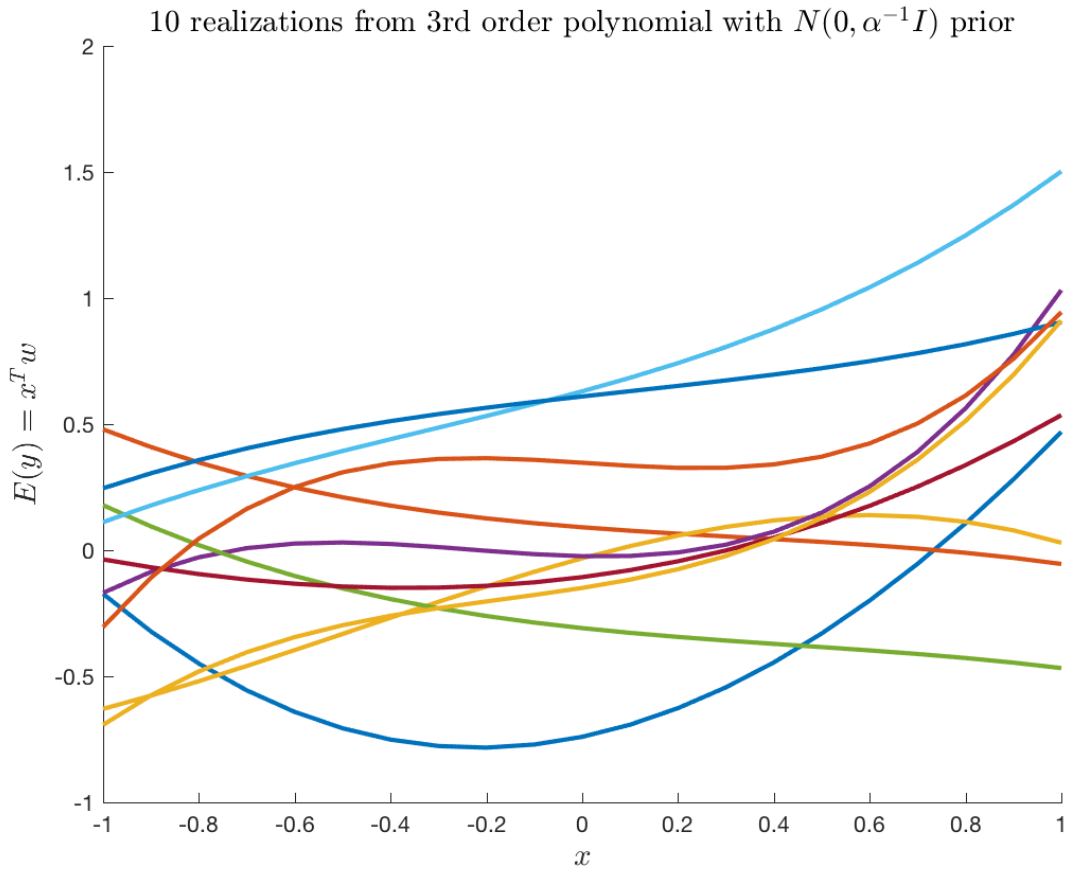
$$\log p(\boldsymbol{w}|X, \boldsymbol{y}) \propto \log p(\boldsymbol{y}|X, \boldsymbol{w}) + \log p(\boldsymbol{w}) \propto -\frac{1}{2\sigma_n^2}||\boldsymbol{y} - X^T \boldsymbol{w}||^2 - \frac{1}{2}\boldsymbol{w}^T \Sigma_p^{-1} \boldsymbol{w}.$$

- So, $\boldsymbol{w}^{MAP}$ can be seen as a penalized/regularized ML estimate.
- Specifically, $p(\boldsymbol{w}|X, \boldsymbol{y}) = \mathcal{N}(\bar{\boldsymbol{w}} = \frac{1}{\sigma_n^2} A^{-1} X \boldsymbol{y}, A^{-1})$ where $A = \sigma_n^{-2} X X^T + \Sigma_p^{-1}$, and thus $\boldsymbol{w}^{MAP} = \bar{\boldsymbol{w}}$.
- A full Bayesian approach does not use $\boldsymbol{w}^{MAP}$ but the predictive distribution:

$$p(f_*|\boldsymbol{x}_*, X, \boldsymbol{y}) = \int p(f_*|\boldsymbol{x}_*, \boldsymbol{w}) p(\boldsymbol{w}|X, \boldsymbol{y}) d\boldsymbol{w} = \mathcal{N}(\frac{1}{\sigma_n^2} \boldsymbol{x}_* A^{-1} X \boldsymbol{y}, \boldsymbol{x}_*^T A^{-1} \boldsymbol{x}_*).$$

Bayesian Linear Regression

- A prior on $\boldsymbol{w}$ is a prior on f.

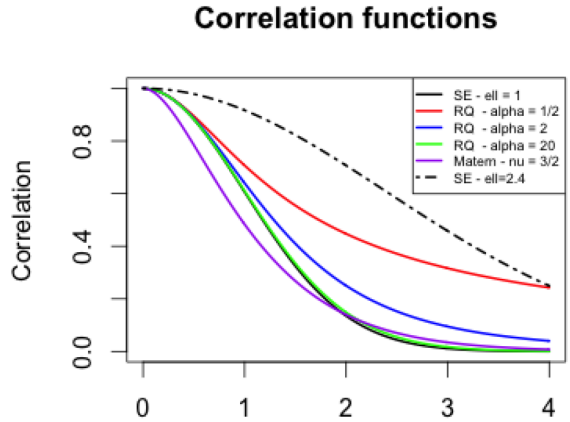


Gaussian Processes Regression

- A GP defines a prior distribution **over functions directly**, instead of indirectly through weights as before. Therefore, a GP operates on the space of functions rather than on the space of weights. Operating in either space is equivalent. A GP defines a prior over functions by defining a prior over a **finite** number of input points.
- Formally, a GP is a collection of random variables, any finite number of which have a joint Gaussian distribution. Hence, a GP is defined as
  - $f(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}'))$ where
  - $m(\boldsymbol{x}) = E[f(\boldsymbol{x})]$ is the mean function (assumed to be zero hereinafter), and
  - $k(\boldsymbol{x}, \boldsymbol{x}') = E[(f(\boldsymbol{x}) - m(\boldsymbol{x}))(f(\boldsymbol{x}') - m(\boldsymbol{x}'))]$ is the covariance function, e.g. squared exponential:

$$k(\boldsymbol{x}, \boldsymbol{x}') = cov(f(\boldsymbol{x}), f(\boldsymbol{x}')) = \sigma_f^2 \exp\left\{ - \frac{||\boldsymbol{x} - \boldsymbol{x}'||^2}{2\ell^2} \right\}$$

i.e. highly correlated **function values** for close **input points**. Intuitively, σ_f^2 is the overall variance of the function, and ℓ is the distance we have to move in the input space for the function to vary significantly.



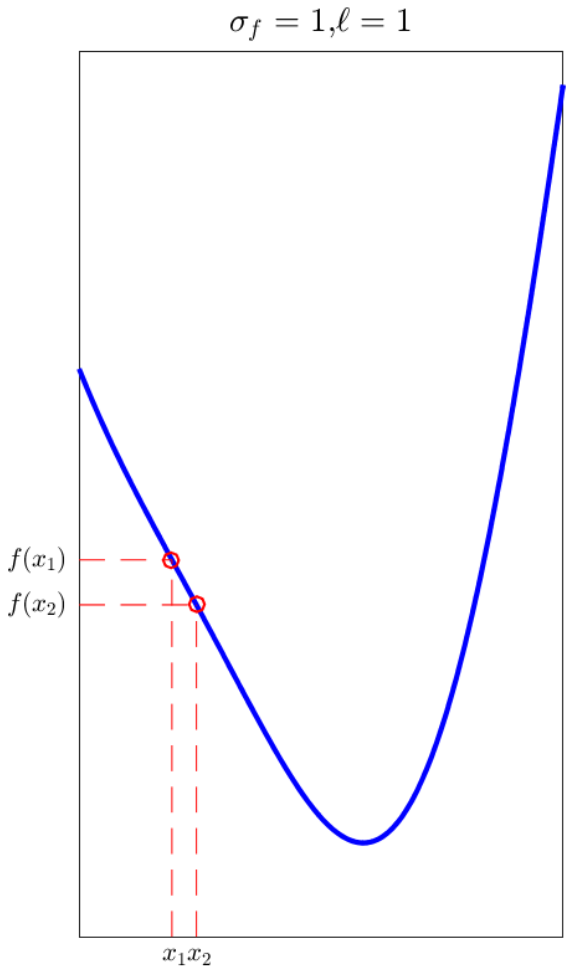
Gaussian Processes Regression

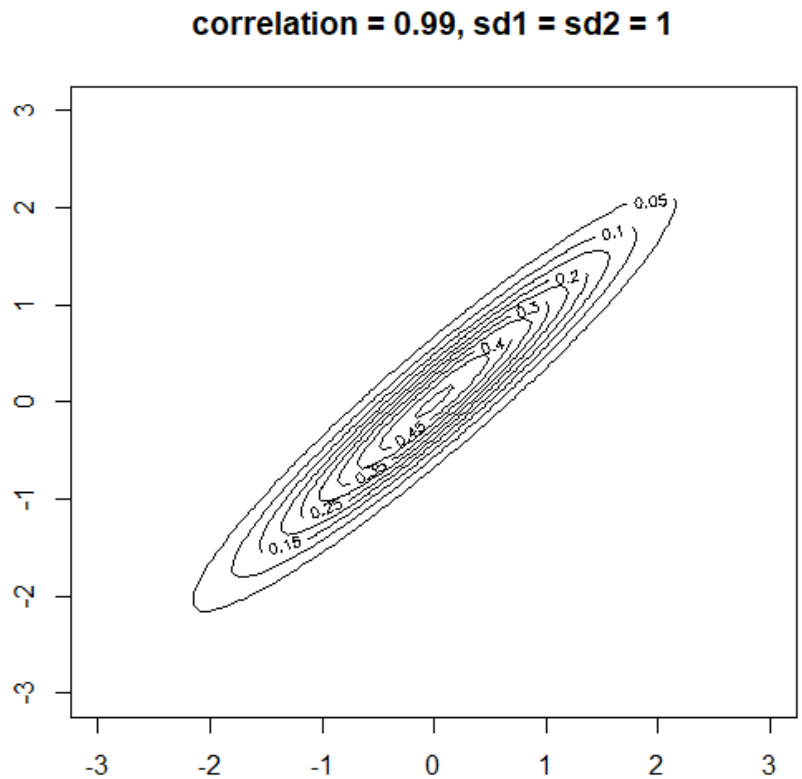
- Formally, a GP is a collection of random variables, any finite number of which have a joint Gaussian distribution. Hence, a GP is defined as
  - $f(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}'))$ where
  - $m(\boldsymbol{x}) = E[f(\boldsymbol{x})]$ is the mean function (assumed to be zero hereinafter), and
  - $k(\boldsymbol{x}, \boldsymbol{x}') = E[(f(\boldsymbol{x}) - m(\boldsymbol{x}))(f(\boldsymbol{x}') - m(\boldsymbol{x}'))]$ is the covariance function, e.g. squared exponential:

$$k(\boldsymbol{x}, \boldsymbol{x}') = cov(f(\boldsymbol{x}), f(\boldsymbol{x}')) = \sigma_f^2 \exp\left\{ - \frac{||\boldsymbol{x} - \boldsymbol{x}'||^2}{2\ell^2} \right\}.$$

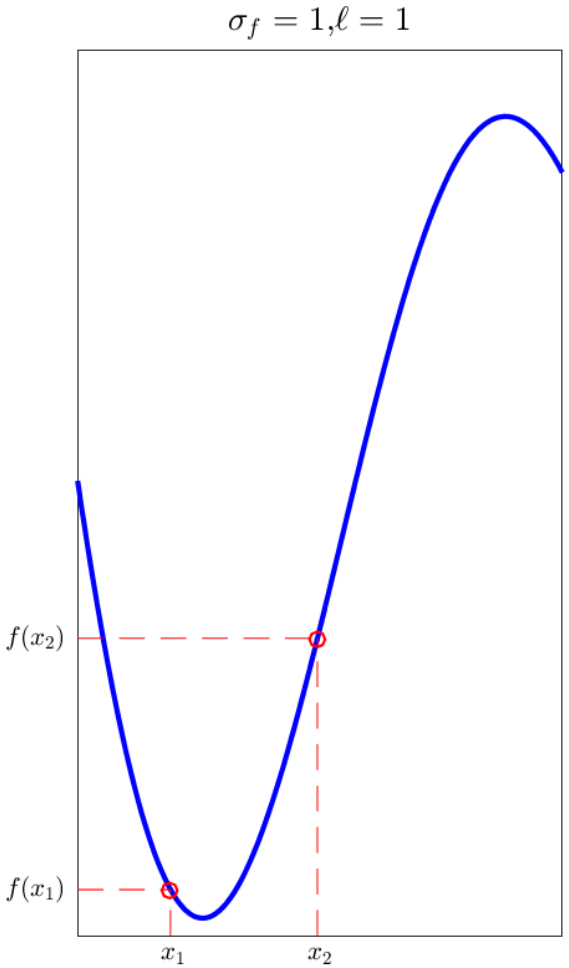
- Note that each random variable or dimension in a GP is a **function value** at an input point. Hence, a GP specifies a probability distribution over functions at any **finite** number of input points.
- Note that the covariance between the **function values** is written as a function of the **inputs points**. Note that the covariances are non-negative.
- Note that the zero mean assumption affects the location but not the smoothness of the prior. Note also that the posterior mean is not assumed to be zero. However, it may get close to zero in regions with few input points when ℓ is small, because the posterior is then similar to the prior. Of course, one can always center or standarize the data.
- We can sample the function space by sampling the GP at any number of chosen input points X_*. To do so, we sample a multivariate Gaussian distribution with the corresponding covariance matrix, i.e. $\boldsymbol{f}_* | X_* \sim \mathcal{N}(0, K(X_*, X_*))$.
- Demo of `GaussianProcesses.R`.

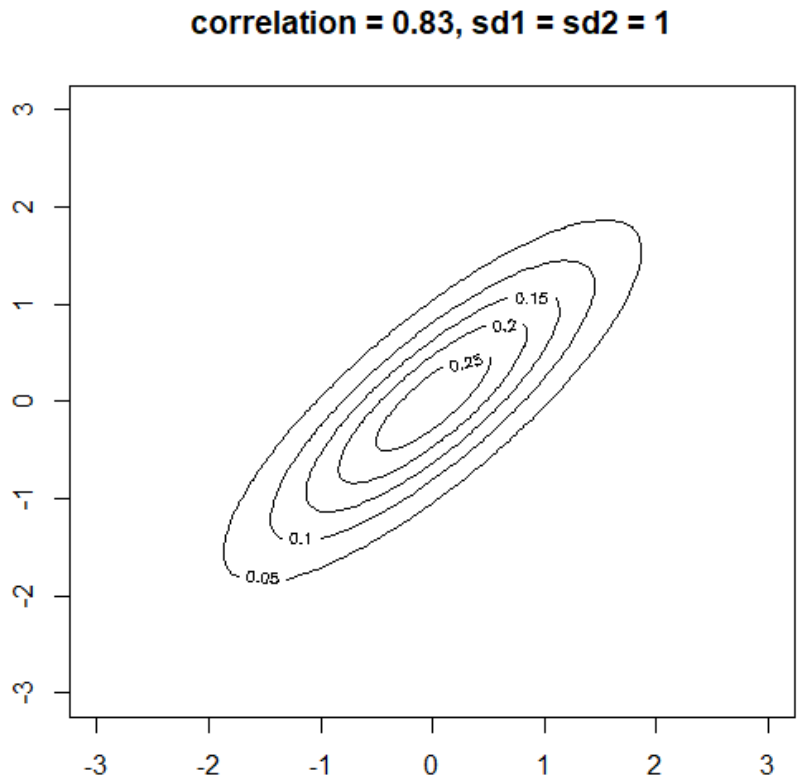
Squared Exponential Covariance: Smooth Function, Close Points



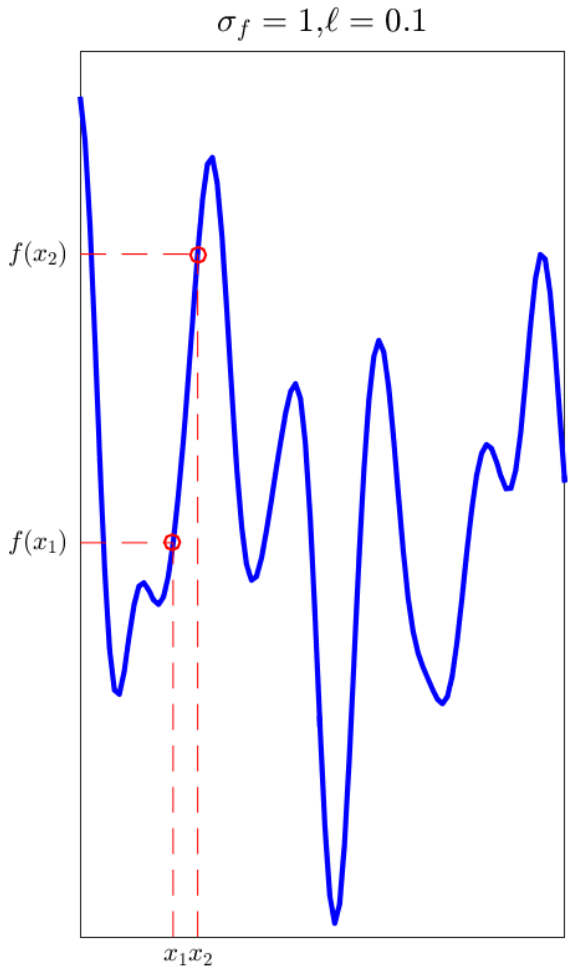


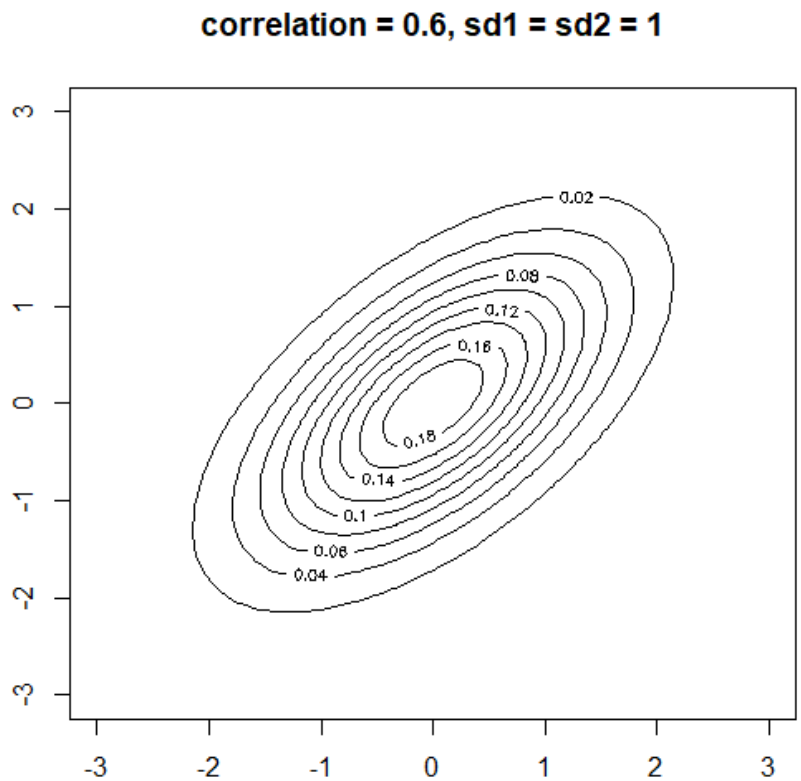
Squared Exponential Covariance: Smooth Function, Distant Points



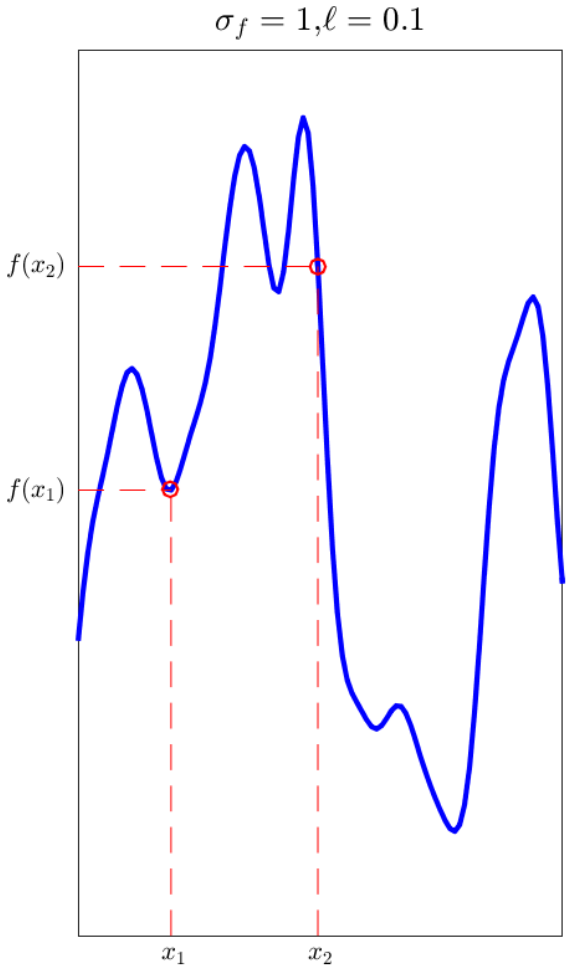


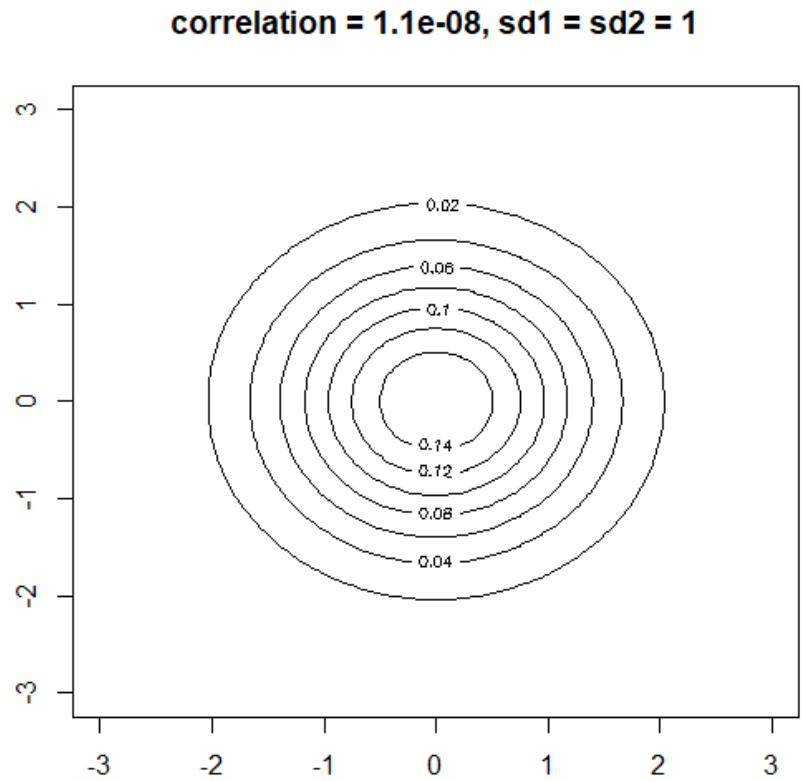
Squared Exponential Covariance: Jagged Function, Close Points





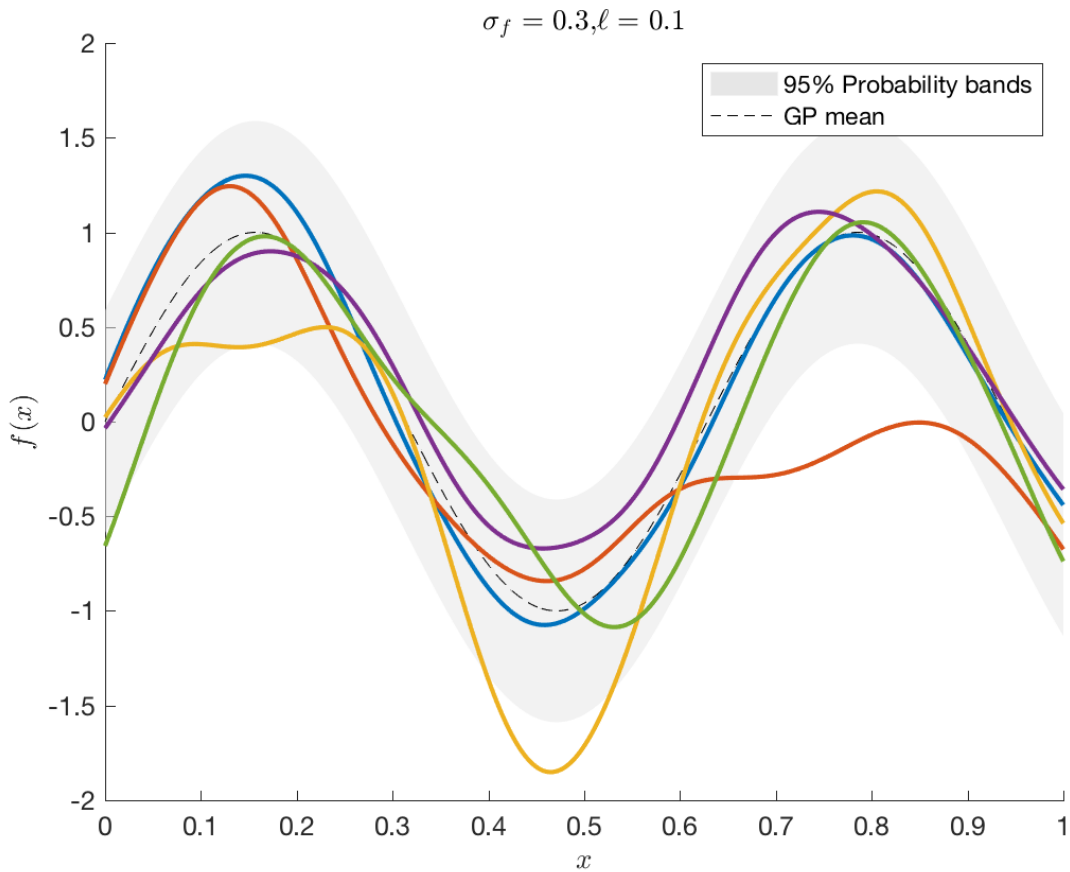
Squared Exponential Covariance: Jagged Function, Distant Points





Gaussian Process Sampling

- To sample a GP at points $X_* = \{x_1, \ldots, x_n\}$, we sample a multivariate Gaussian distribution $\mathcal{N}(0, K(X_*, X_*))$.



Gaussian Process Regression

- With no training data, sample from $\boldsymbol{f}_*|X_* \sim \mathcal{N}(0, K(X_*, X_*))$.
- With **noise-free** training data $\mathcal{D} = \{(\boldsymbol{x}_i, f_i)|i = 1, \ldots, n\} = (X, \boldsymbol{f})$, build

$$\begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{f}_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K(X, X) & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix}\right)$$

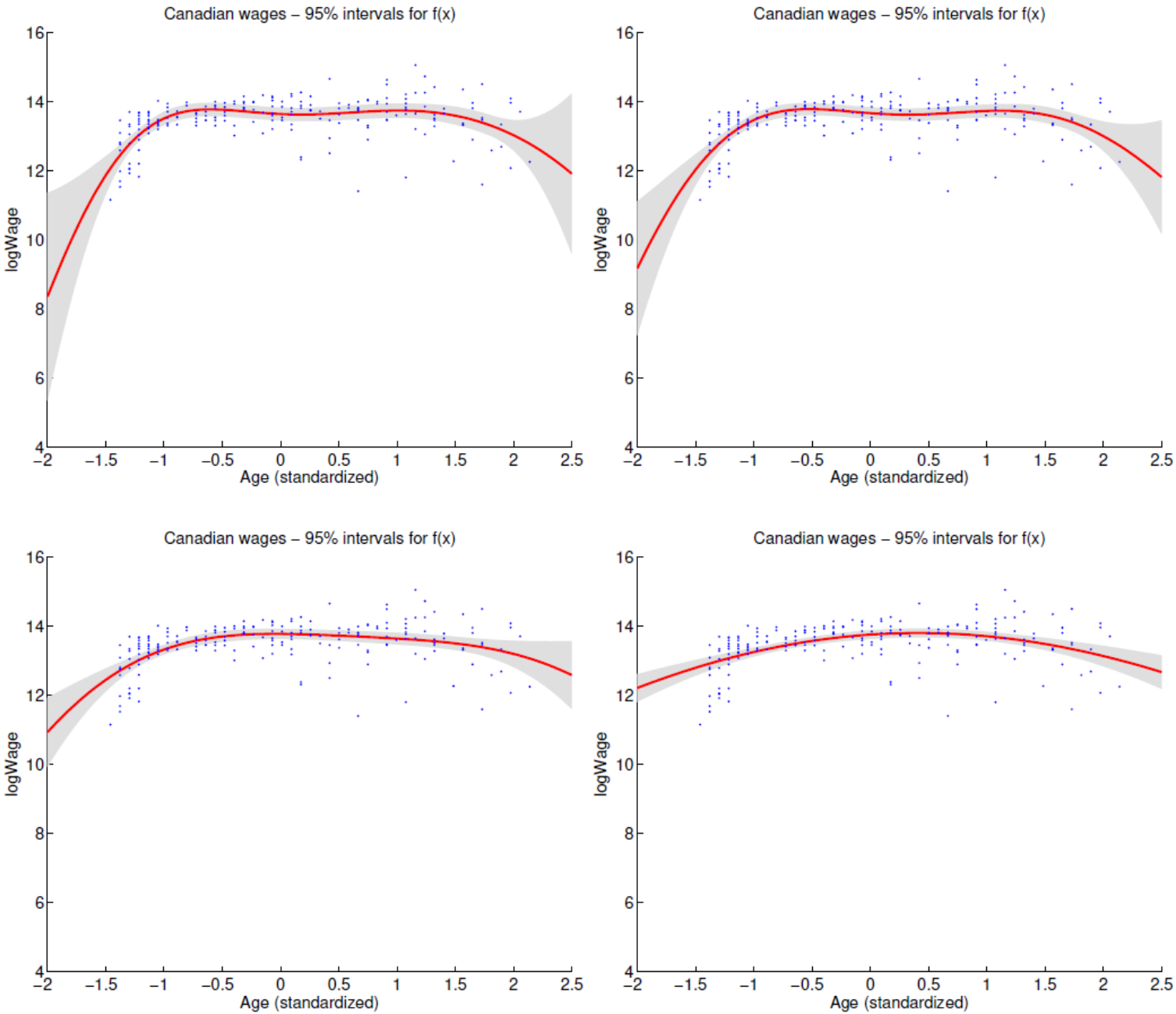
  and sample from $\boldsymbol{f}_*|X_*, X, \boldsymbol{f} \sim$
  $\mathcal{N}(K(X_*, X)K(X, X)^{-1}\boldsymbol{f}, K(X_*, X_*) - K(X_*, X)K(X, X)^{-1}K(X, X_*))$.
- With **noisy** training data $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)|i = 1, \ldots, n\} = (X, \boldsymbol{y})$, build[1]

$$\begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{f}_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K(X, X) + \sigma_n^2 I & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix}\right)$$

  and sample from $\boldsymbol{f}_*|X_*, X, \boldsymbol{y} \sim \mathcal{N}(K(X_*, X)[K(X, X) +$
  $\sigma_n^2 I]^{-1}\boldsymbol{y}, K(X_*, X_*) - K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-1}K(X, X_*))$.
- Demo of `KernLabDemo.R`.

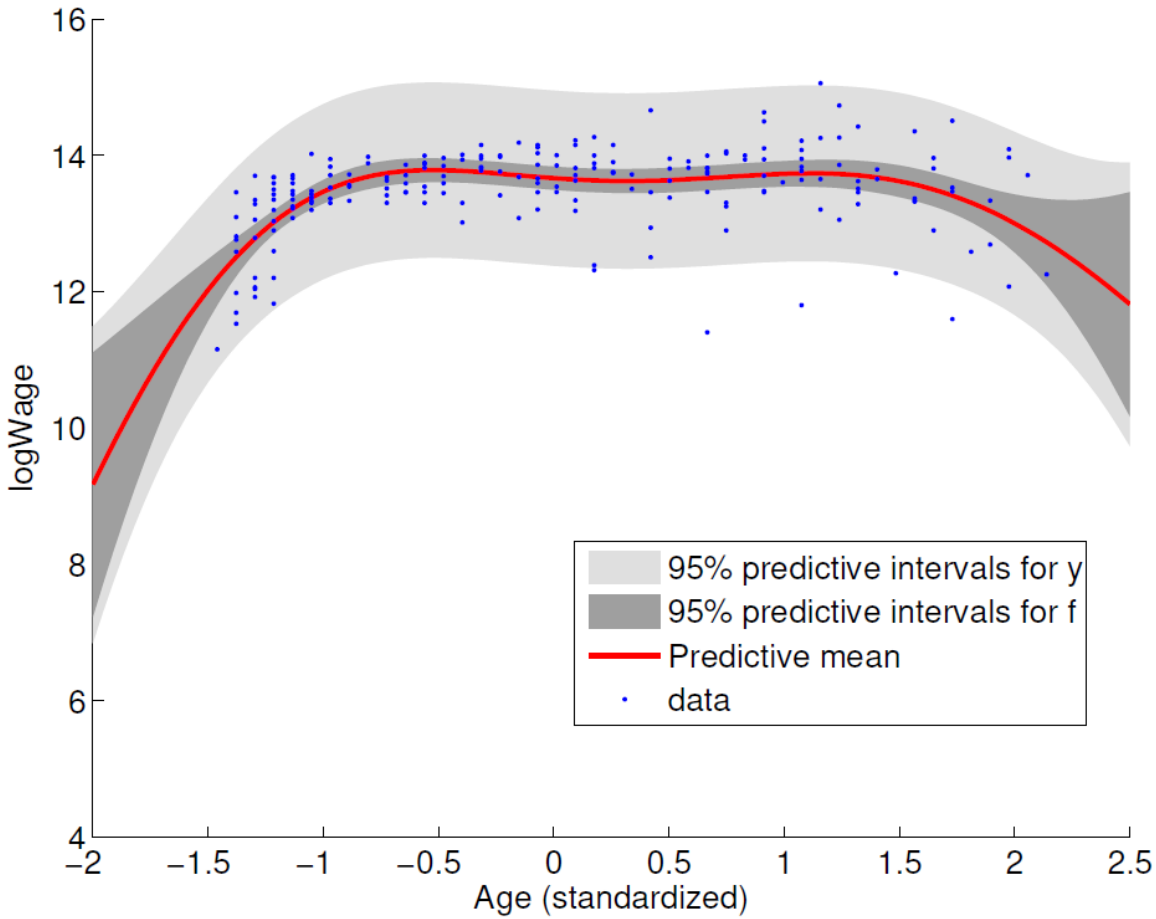
[1] $cov(y, y') = cov(f(x) + \epsilon, f(x') + \epsilon') =$
$cov(f(x), f(x')) + cov(f(x), \epsilon') + cov(\epsilon, f(x')) + cov(\epsilon, \epsilon') = cov(f(x), f(x')) + \sigma_n^2 \delta_{xx'}$.

Gaussian Process Regression: Canadian Wages ($\ell = 0.2, 0.5, 1, 2$)



Gaussian Process Regression: Canadian Wages ($\ell = 0.5$)

- Predictive interval for $\boldsymbol{f}_*$: mean($\boldsymbol{f}_*$) $\pm$ 1.96 sqrt(var($\boldsymbol{f}_*$)).
- Predictive interval for $\boldsymbol{y}_*$: mean($\boldsymbol{f}_*$) $\pm$ 1.96 sqrt(var($\boldsymbol{f}_*$) + σ_n^2).



Contents

Thank you

732A96/TDDE15 Advanced Machine Learning

Gaussian Process Regression and Classification

Jose M. Peña
IDA, Linköping University, Sweden

Lectures 10: Kernels, Hyperparameter Learning and More

Contents

Literature

- Main source
  - Rasmussen, C. E. and Williams, K. I. *Gaussian Processes for Machine Learning*. MIT Press, 2006. Chapters 2.3, 5.1-5.4.1.
- Additional source
  - Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006. Chapters 6.4.3-6.4.4.

Three Common Covariance Functions

- Let $r = ||\boldsymbol{x} - \boldsymbol{x}'||$.
- Squared exponential (SE):

$$k_{SE}(r) = \sigma_f^2 \exp\left\{ -\frac{r^2}{2\ell^2} \right\}$$

  where $\sigma_f^2 > 0, \ell > 0$. Very smooth.
- Rational quadratic (RQ):

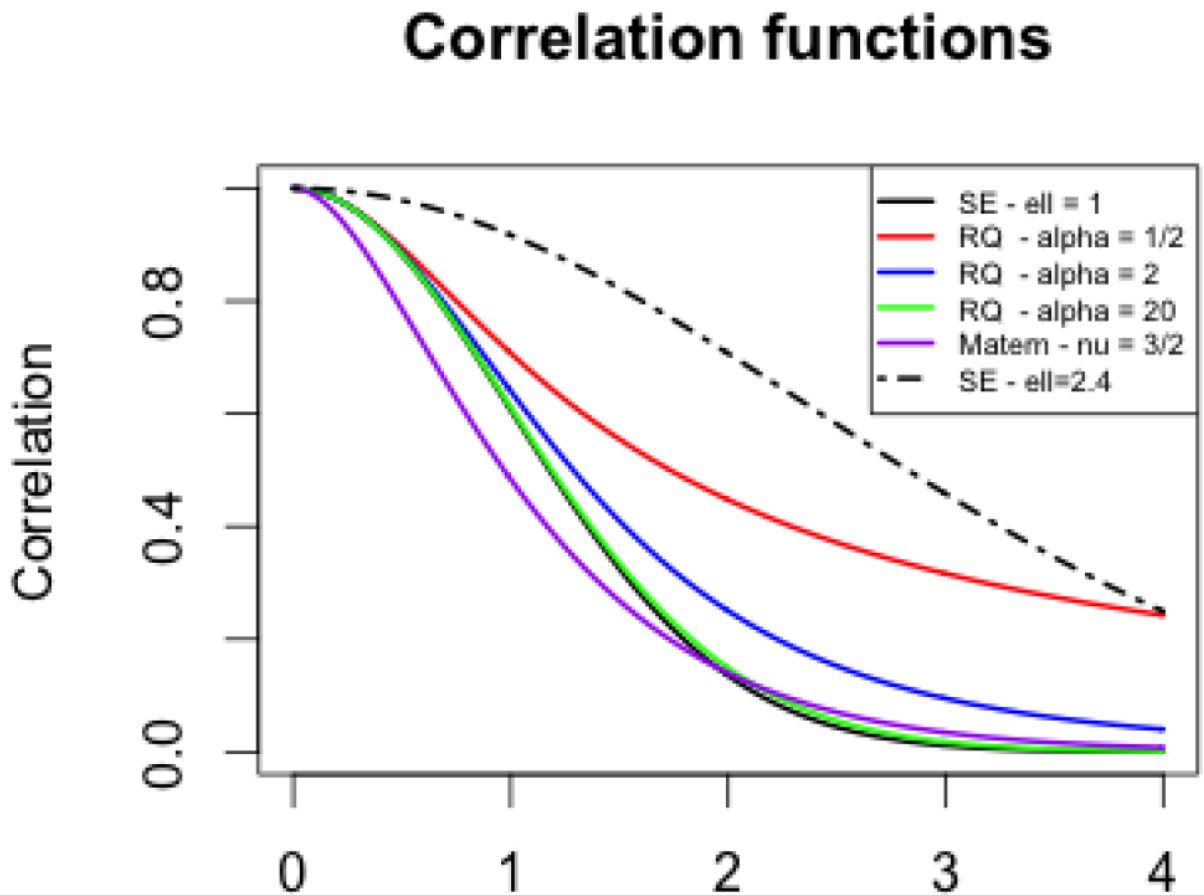
$$k_{RQ}(r) = \sigma_f^2 \left(1 + \frac{r^2}{2\alpha\ell^2} \right)^{-\alpha}$$

  $\sigma_f^2 > 0, \ell > 0, \alpha > 0$. k_{RQ} is an infinite sum of k_{SE} with different ℓ. As $\alpha \to \infty$, $k_{RQ}(r) \to k_{SE}(r)$.
- Matérn:

$$k_{Matern} = \sigma_f^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu} r}{\ell} \right)^{\nu} K_\nu \left(\frac{\sqrt{2\nu} r}{\ell} \right)$$

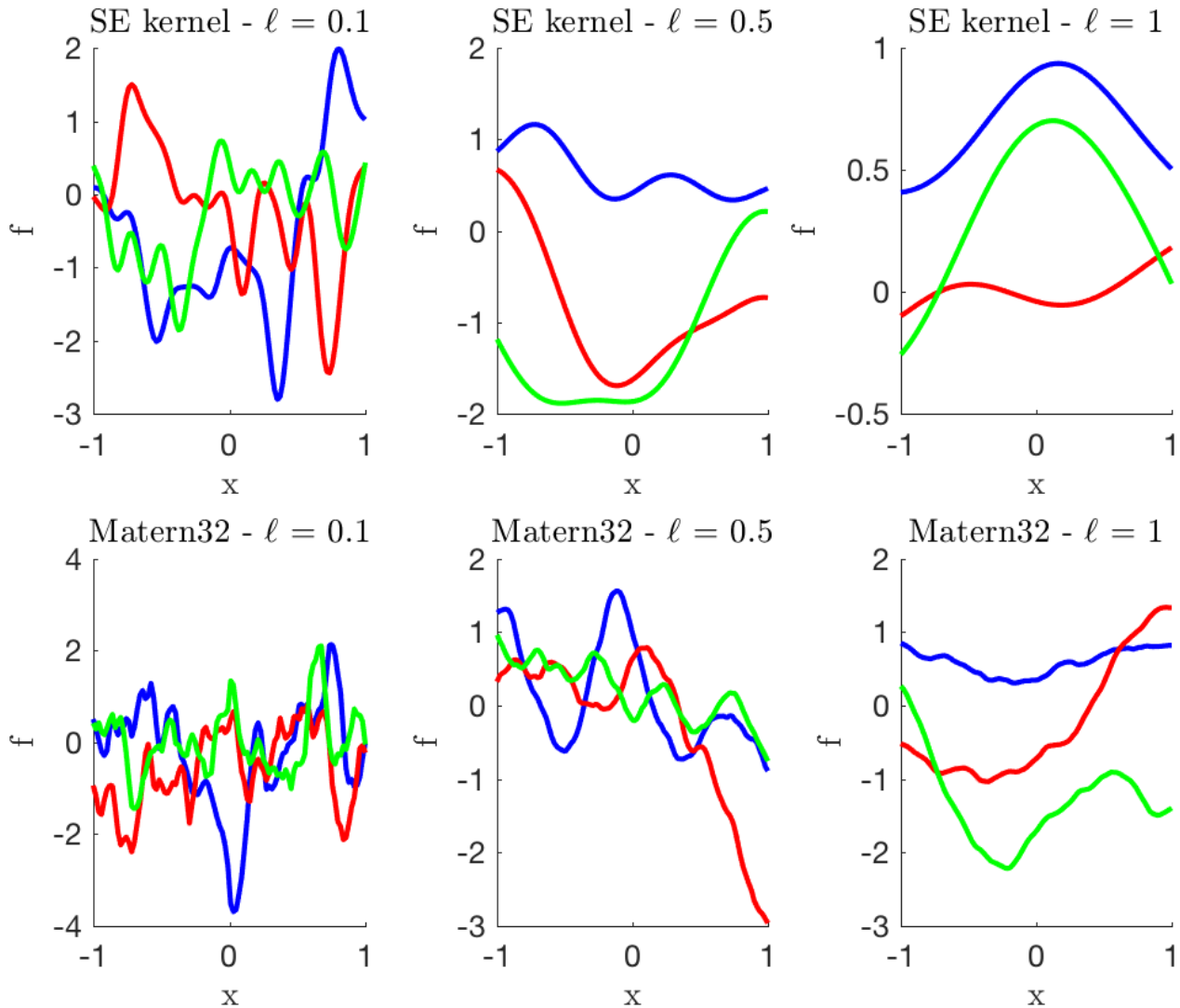
  where $\sigma_f^2 > 0, \ell > 0, \nu > 0$, and K_ν is the modified Bessel function. As $\nu \to \infty$, $k_{Matern}(r) \to k_{SE}(r)$.
- Demo of `GaussianProcesses.R` and `KernLabDemo.R`.

Three Common Covariance Functions



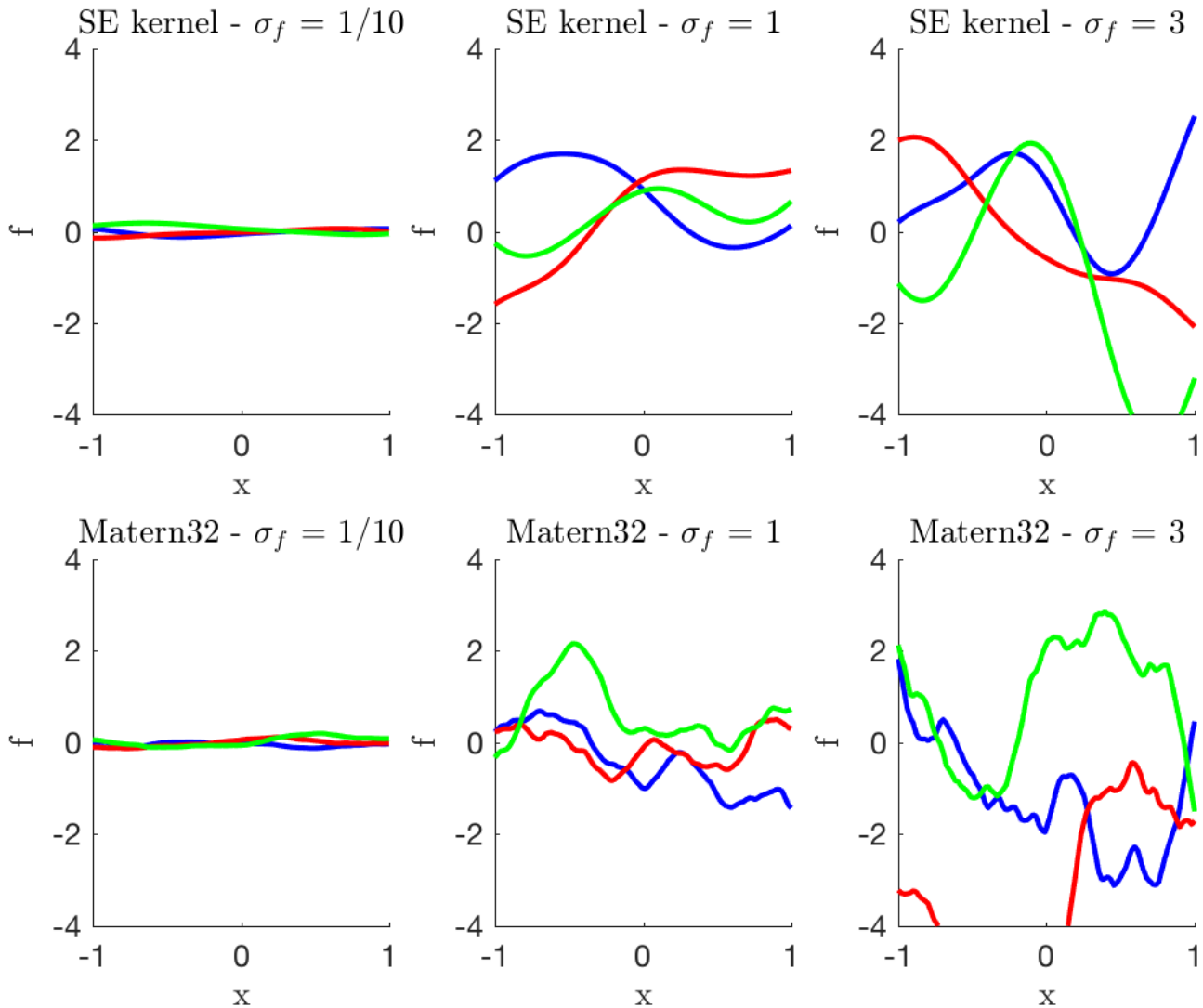
Three Common Covariance Functions

- The length scale ℓ determines the smoothness.



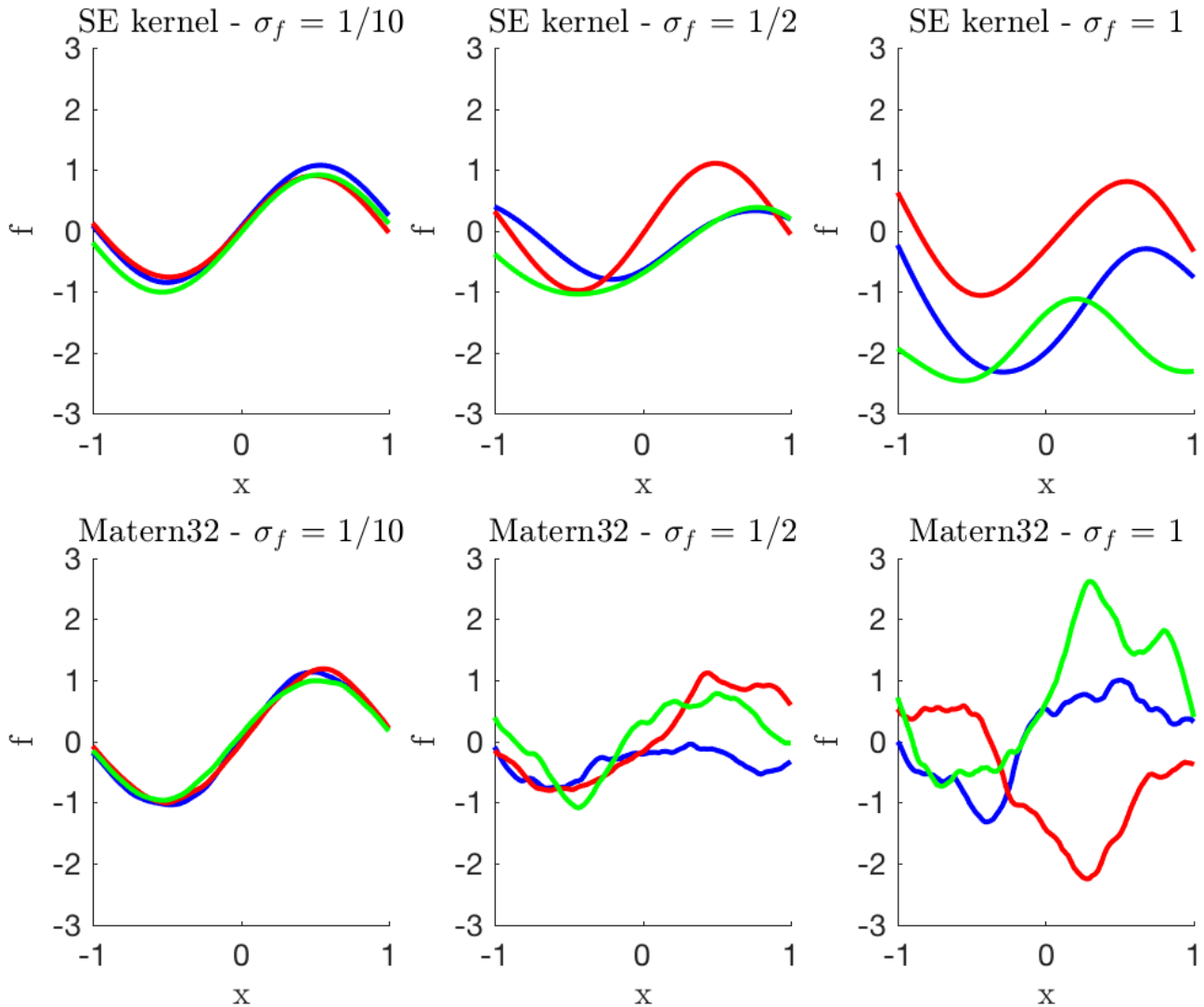
Three Common Covariance Functions

- The scale factor σ_f determines the variance.



Three Common Covariance Functions

- The mean can be arbitrary, e.g. $\sin(3x)$.



Learning the Hyperparameters of the Covariance Function

- Let θ denote the hyperparameters of the covariance function, i.e. $\theta = (\sigma_f, \ell)$ for k_{SE}, $\theta = (\sigma_f, \ell, \alpha)$ for k_{RQ}, and $\theta = (\sigma_f, \ell, \nu)$ for k_{Matern}.
- Choose the hyperparameters that maximize the marginal likelihood:

$$\log p(\boldsymbol{y}|X,\theta) = -\frac{1}{2}\boldsymbol{y}^T(K(X,X)+\sigma_n^2 I)^{-1}\boldsymbol{y} - \frac{1}{2}\log|K(X,X)+\sigma_n^2 I| - \frac{n}{2}\log 2\pi$$

  which follows from

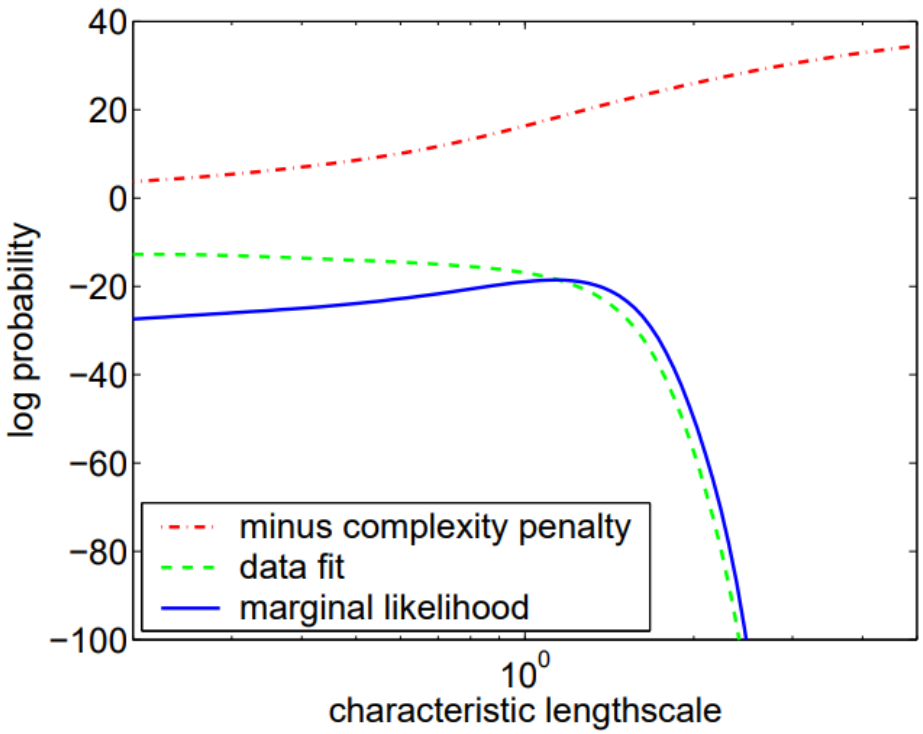
$$\begin{bmatrix}\boldsymbol{y}\\ \boldsymbol{f}_*\end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix}K(X,X)+\sigma_n^2 I & K(X,X_*)\\ K(X_*,X) & K(X_*,X_*)\end{bmatrix}\right).$$

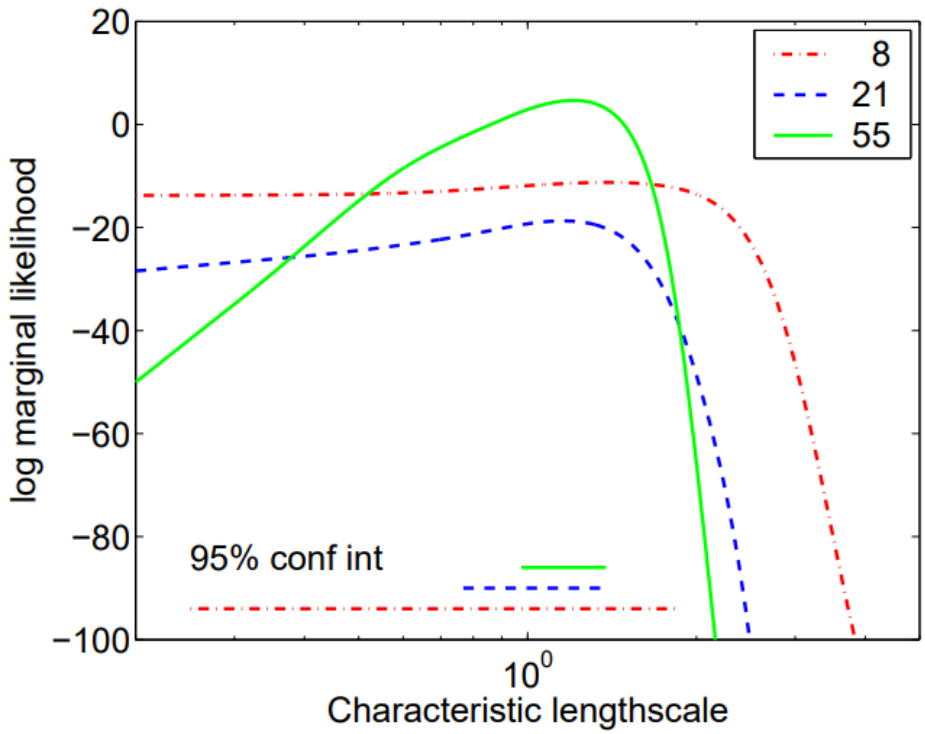
- In general, this is a non-convex optimization problem, and gradient methods are typically used. For most common covariance functions, the derivative of $K(X,X)$ wrt θ is easy to compute.
- For a Bayesian approach, choose the hyperparameters that maximize the posterior distribution $p(\theta|\boldsymbol{y},X) \propto p(\boldsymbol{y}|X,\theta)p(\theta)$. It typically requires MCMC sampling or Laplace approximation.
- The methods above can also be used to select among covariance functions, i.e. simply include them as hyperparameters. Cross-validation is also an option.

Learning the Hyperparameters of the Covariance Function

$$\log p(\boldsymbol{y}|X,\theta) = -\frac{1}{2}\boldsymbol{y}^T(K(X,X)+\sigma_n^2 I)^{-1}\boldsymbol{y} - \frac{1}{2}\log|K(X,X)+\sigma_n^2 I| - \frac{n}{2}\log 2\pi$$

$$= \text{data fit - model complexity - normalization constant.}$$





Lab: Algorithm 2.1 in Rasmussen and Williams

input: X (inputs), $\mathbf{y}$ (targets), k (covariance function), σ_n^2 (noise level), $\mathbf{x}_*$ (test input)

2: $L := \text{cholesky}(K + \sigma_n^2 I)$

$\boldsymbol{\alpha} := L^\top \backslash (L \backslash \mathbf{y})$

4: $\bar{f}_* := \mathbf{k}_*^\top \boldsymbol{\alpha}$ } predictive mean eq. (2.25)

$\mathbf{v} := L \backslash \mathbf{k}_*$

6: $\mathbb{V}[f_*] := k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{v}^\top \mathbf{v}$ } predictive variance eq. (2.26)

$\log p(\mathbf{y}|X) := -\frac{1}{2}\mathbf{y}^\top \boldsymbol{\alpha} - \sum_i \log L_{ii} - \frac{n}{2}\log 2\pi$ eq. (2.30)

8: **return**: $\bar{f}_*$ (mean), $\mathbb{V}[f_*]$ (variance), $\log p(\mathbf{y}|X)$ (log marginal likelihood)

- $K = K(X, X)$.
- $\boldsymbol{k}_* = K(X, X_*)$.
- $L = cholesky(A) \Rightarrow A = LL^T \Rightarrow A^{-1} = (L^T)^{-1}L^{-1} = (L^{-1})^T L^{-1}$ and $|A| = det(A) = det(L)det(L^T) = (\prod_i L_{ii})^2$.
- $L \backslash \boldsymbol{y} = solve(L, \boldsymbol{y}) = L^{-1}\boldsymbol{y}$.
- The algorithm uses Cholesky decomposition instead of matrix inversion because it is faster and numerically more stable.
- It returns the predictive distribution for noise-free test data, i.e. $\boldsymbol{f}_*$. Add σ_n^2 to the predictive variances to obtain the distribution for noisy test data, i.e. $\boldsymbol{y}_*$
- It is presented for a single test case but it also works for several test cases.

Contents

- Three Common Covariance Functions
- Learning the Hyperparameters of the Covariance Function
- Lab: Algorithm 2.1 in Rasmussen and Williams

Thank you

732A96/TDDE15 Advanced Machine Learning

Gaussian Process Regression and Classification

Jose M. Peña
IDA, Linköping University, Sweden

Lectures 11: Gaussian Process Classification

Contents

Literature

- Main source
  - Rasmussen, C. E. and Williams, K. I. *Gaussian Processes for Machine Learning*. MIT Press, 2006. Chapters 3.1-3.4.1 and 3.7.
- Additional source
  - Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006. Chapters 6.4.5-6.4.6.

Linear Logistic Regression

- Consider a binary classification problem $y \in \{-1, +1\}$. Then,

$$p(y = +1|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|y = +1)p(y = +1)}{p(\boldsymbol{x}|y = +1)p(y = +1) + p(\boldsymbol{x}|y = -1)p(y = -1)} = \sigma(s(\boldsymbol{x}))$$

  where $s(\boldsymbol{x}) = \log \frac{p(\boldsymbol{x}|y=+1)p(y=+1)}{p(\boldsymbol{x}|y=-1)p(y=-1)} = \log \frac{p(y=+1|\boldsymbol{x})}{p(y=-1|\boldsymbol{x})}$ is the log odds ratio, and $\sigma(a) = \frac{1}{1+exp(-a)}$ is the logistic sigmoid function.

- We assume that $p(\boldsymbol{x}|y)$ is a member of the exponential family (e.g., Gaussian, multinomial), which implies that $s(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{w}$. The model $p(y = +1|\boldsymbol{x}, \boldsymbol{w}) = \sigma(\boldsymbol{x}^T\boldsymbol{w})$ is called logistic regression.
- Given some training data $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)|i = 1, \ldots, n\} = (X, \boldsymbol{y})$, we determine the parameters $\boldsymbol{w}$ by maximizing the log lik function:

$$\log p(\boldsymbol{y}|X, \boldsymbol{w}) = \sum_{i=1}^{n} \log \sigma(y_i(\boldsymbol{x}_i^T\boldsymbol{w}))$$

  since $\sigma(-a) = 1 - \sigma(a)$.

- No closed form solution exists, but the log lik function is concave and thus easy to maximize via gradient ascent.
- Beware of overfitting for linearly separable datasets: Log lik maximization causes $|\boldsymbol{w}|$ to tend to infinity, i.e. the sigmoid function becomes a Heaviside step function.

Bayesian Linear Logistic Regression

- Prior distribution: $\boldsymbol{w} \sim \mathcal{N}(0, \Sigma_p)$, e.g. ridge regression $\Sigma_p = \alpha^{-1} I$.
- Posterior distribution:

$$\log p(\boldsymbol{w}|X, \boldsymbol{y}) \propto \sum_{i=1}^{n} \log \sigma(y_i(\boldsymbol{x}_i^T \boldsymbol{w})) - \frac{1}{2} \boldsymbol{w}^T \Sigma_p^{-1} \boldsymbol{w}.$$

- No closed form solution exists, but the penalty term is quadratic on $\boldsymbol{w}$ and thus the log posterior is concave and thus easy to maximize via gradient ascent or related methods.
- A full Bayesian approach uses the predictive distribution:

$$p(y_* = +1|\boldsymbol{x}_*, X, \boldsymbol{y}) = \int \sigma(\boldsymbol{x}_*^T \boldsymbol{w}) p(\boldsymbol{w}|X, \boldsymbol{y}) d\boldsymbol{w}.$$

- No closed form expression for the predictive distribution exists.
- The above carries over to multi-class classification problems by using the multiple logistic function, a.k.a. softmax.

Gaussian Process Classification

- Consider a **binary** classification problem $y \in \{-1, +1\}$.
- Given a test case $\boldsymbol{x}_*$, use a GP for regression to predict a real number f_* that is then "squashed" through the logistic function to produce a class label $y_* = \sigma(f_*)$.
- However, the training data only include class labels $\boldsymbol{y}$ and, thus, $\boldsymbol{f}$ are **latent** variables.
- In other words, prediction occurs in two steps:
  - Compute the distribution of the latent variable f_*:

$$p(f_*|\boldsymbol{x}_*, X, \boldsymbol{y}) = \int p(f_*|\boldsymbol{x}_*, X, \boldsymbol{f}) p(\boldsymbol{f}|X, \boldsymbol{y}) d\boldsymbol{f}.$$

  - Compute the prediction y_*, since the latent variable f_* is uninteresting:

$$p(y_* = +1|\boldsymbol{x}_*, X, \boldsymbol{y}) = \int \sigma(f_*) p(f_*|\boldsymbol{x}_*, X, \boldsymbol{y}) df_*.$$

  - No closed form solutions exist for these integrals. Solutions: Laplace approximation and/or MC sampling.

Gaussian Process Classification

- Computing the distribution of the latent variable can be rewritten as

$$p(f_*|\boldsymbol{x}_*, X, \boldsymbol{y}) = \int p(f_*, \boldsymbol{f}|\boldsymbol{x}_*, X, \boldsymbol{y})d\boldsymbol{f} = \int p(f_*|\boldsymbol{x}_*, X, \boldsymbol{f})p(\boldsymbol{f}|X, \boldsymbol{y})d\boldsymbol{f}$$

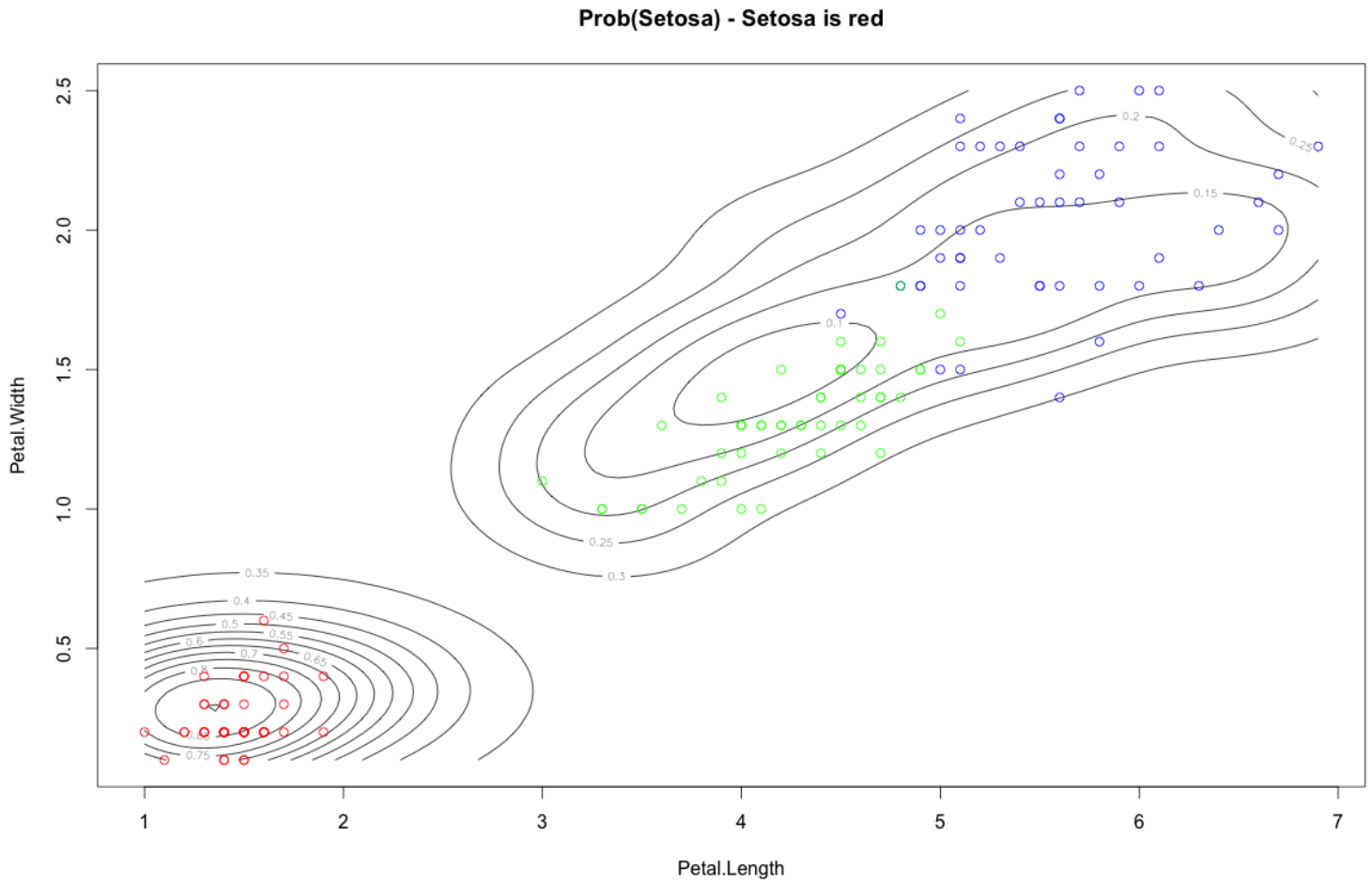
  where
  - the first term is $\mathcal{N}(K(\boldsymbol{x}_*, X)K(X, X)^{-1}\boldsymbol{f}, K(\boldsymbol{x}_*, \boldsymbol{x}_*) - K(\boldsymbol{x}_*, X)K(X, X)^{-1}K(X, \boldsymbol{x}_*))$ since it is a GP for regression, and
  - the second term is approximated by $\mathcal{N}(\hat{\boldsymbol{f}}, A^{-1})$ where $\hat{\boldsymbol{f}} = \arg\max_{\boldsymbol{f}} p(\boldsymbol{f}|X, \boldsymbol{y})$ and $A = -\nabla\nabla \log p(\boldsymbol{f}|X, \boldsymbol{y})|_{\boldsymbol{f}=\hat{\boldsymbol{f}}}$.
- Moreover,

$$p(\boldsymbol{f}|X, \boldsymbol{y}) = p(\boldsymbol{f}, \boldsymbol{y}|X)/p(\boldsymbol{y}|X) \propto p(\boldsymbol{y}|\boldsymbol{f})p(\boldsymbol{f}|X)$$

  i.e. logistic function times GP prior. Typically, numerical methods are used to maximize it.
- Moreover, $A = -\nabla\nabla \log p(\boldsymbol{f}|X, \boldsymbol{y})|_{\boldsymbol{f}=\hat{\boldsymbol{f}}} = -W - K(X, X)^{-1}$ where W is a diagonal matrix with elements $\sigma(\hat{f}_i)(1 - \sigma(\hat{f}_i))$.
- Then, $p(f_*|\boldsymbol{x}_*, X, \boldsymbol{y}) = \mathcal{N}(K(X, \boldsymbol{x}_*)^T K(X, X)^{-1}\hat{\boldsymbol{f}}, K(\boldsymbol{x}_*, \boldsymbol{x}_*) - K(X, \boldsymbol{x}_*)^T(K(X, X) + W^{-1})^{-1}K(X, \boldsymbol{x}_*))$.
- Finally, note that the (approximate) prediction requires one-dimensional numerical integration, or MC sampling.
- The prediction (expected sigmoid) differs from the sigmoid of the expectation ($\sigma(K(X, \boldsymbol{x}_*)^T K(X, X)^{-1}\hat{\boldsymbol{f}})$). Luckily, either both or none are greater than 0.5. So, the latter suffices to find the most probable class.

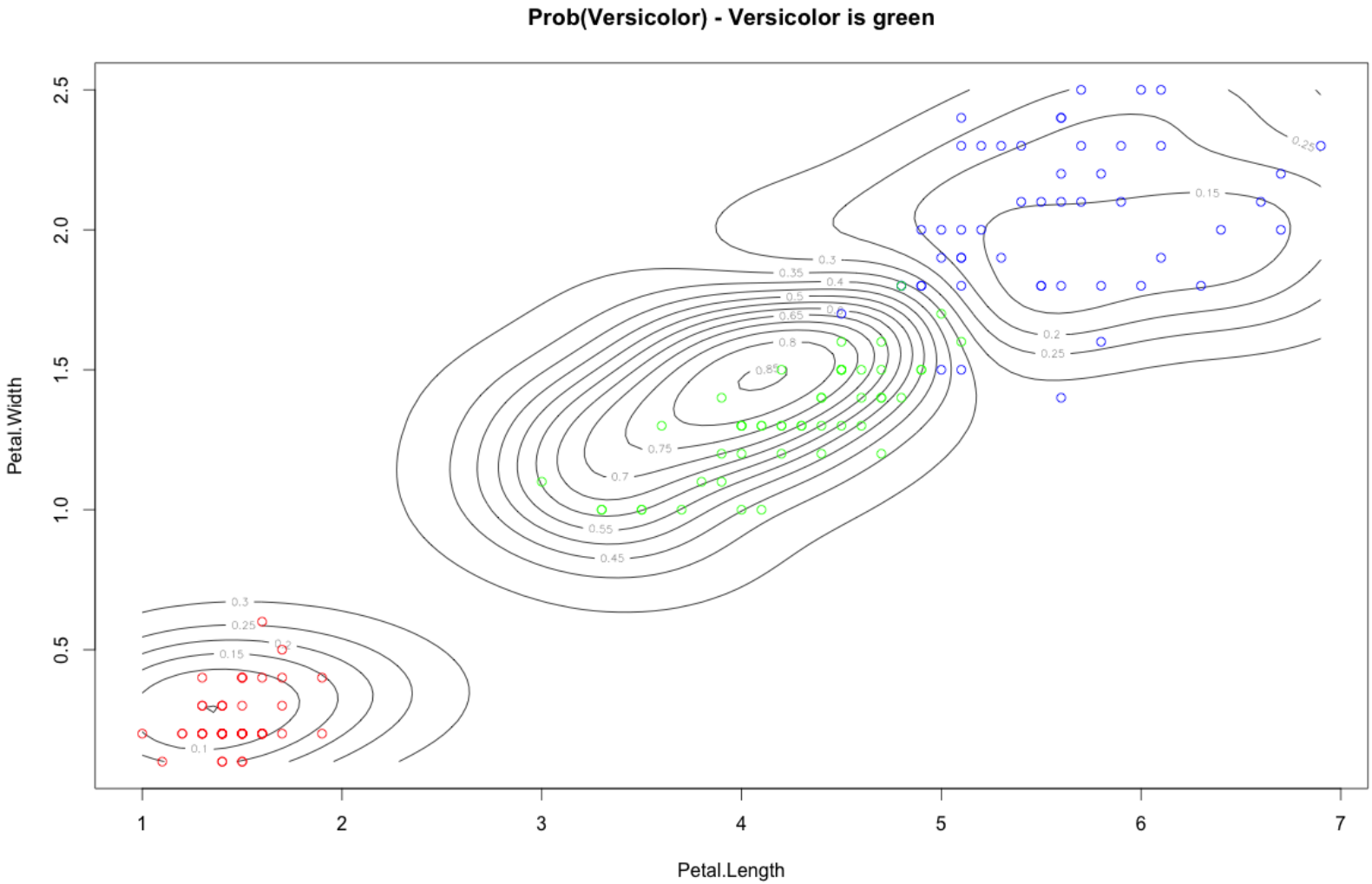
Gaussian Process Classification: Iris Data

- Multi-class classification is solved similarly, i.e. latent variable + softmax + Laplace's approximation + numerical integration or MC sampling.
- Demo of `KernLabDemo.R`.
- SE kernel with automatic ℓ estimation and $\sigma_f = 1$.
- *Species* ~ *Petal.Length* + *Petal.Width*.
- $p(Setosa|Petal.Length, Petal.Width)$:



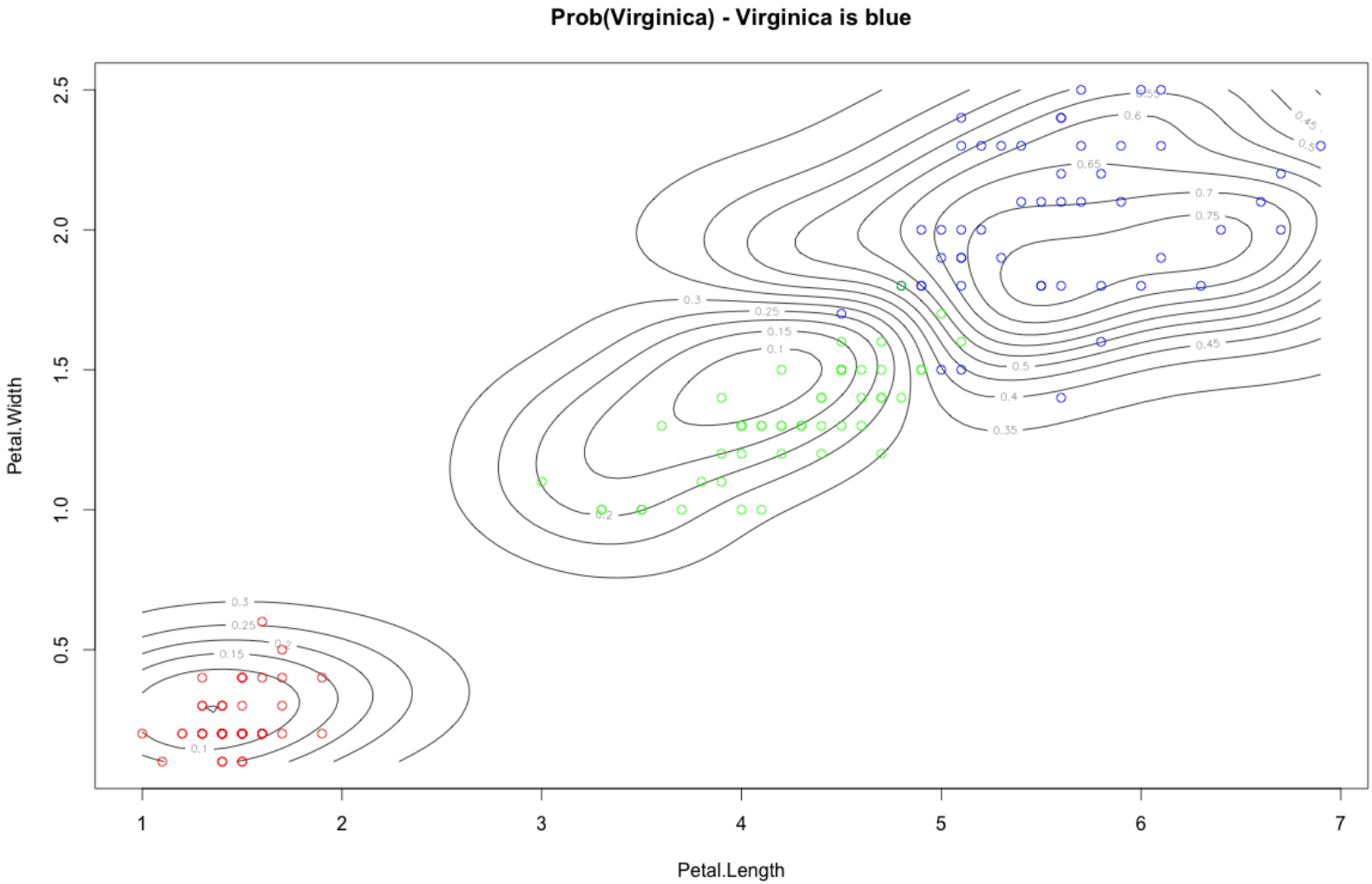
Gaussian Process Classification: Iris Data

- SE kernel with automatic ℓ estimation and $\sigma_f = 1$.
- $Species \sim Petal.Length + Petal.Width$.
- $p(Versicolor|Petal.Length, Petal.Width)$:



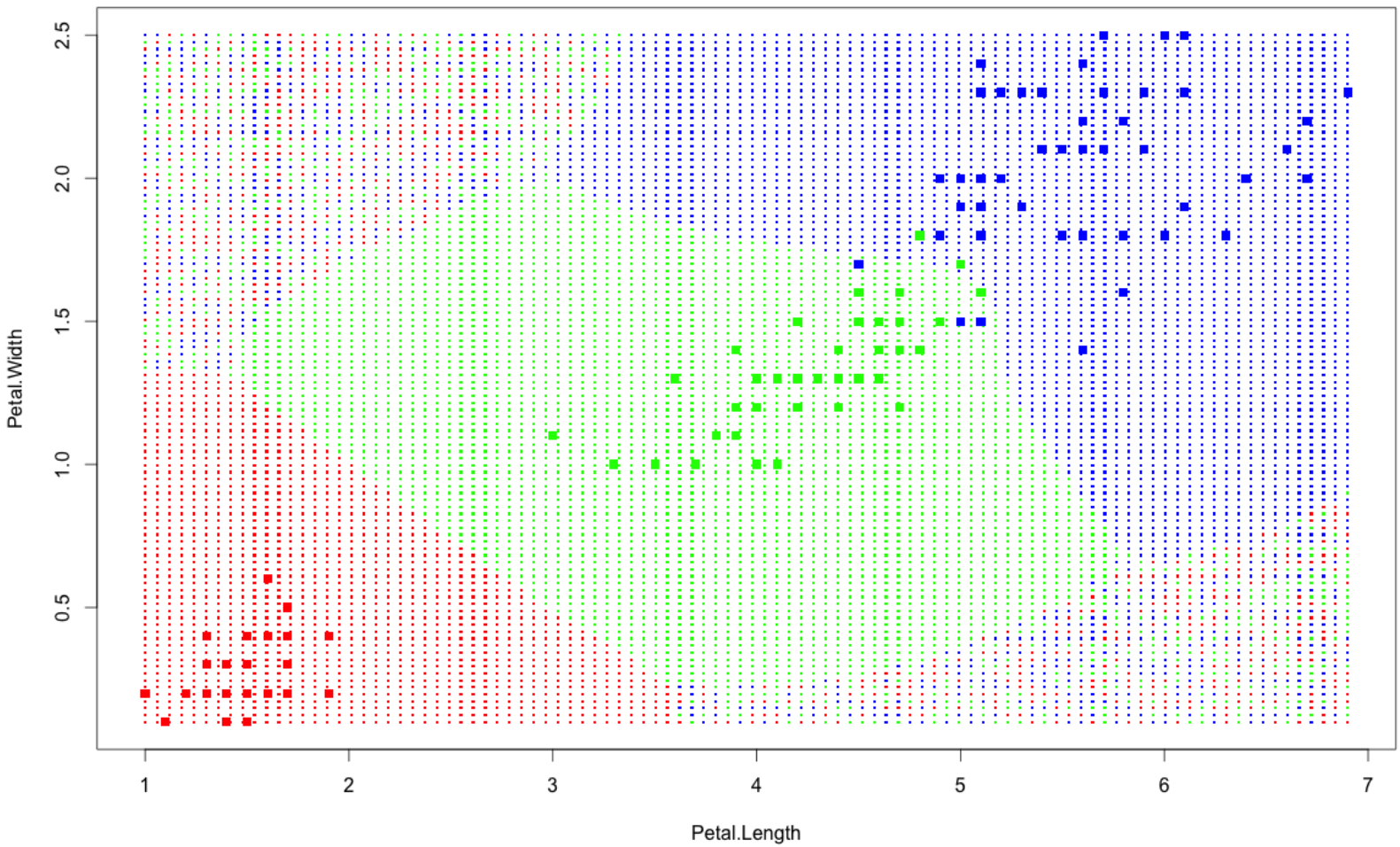
Gaussian Process Classification: Iris Data

- SE kernel with automatic ℓ estimation and $\sigma_f = 1$.
- $Species \sim Petal.Length + Petal.Width$.
- $p(Virginica|Petal.Length, Petal.Width)$:



Gaussian Process Classification: Iris Data

- SE kernel with automatic ℓ estimation and $\sigma_f = 1$.
- *Species* ~ *Petal.Length* + *Petal.Width*.
- Decision boundary:



Contents

Thank you