A Tutorial on Machine Learning for Interactive Pedagogical Systems

Francisco S. Melo[1], Samuel Mascarenhas[2], Ana Paiva[3]

[1]*INESC-ID and Instituto Superior Técnico*
University of Lisbon, Portugal
fmelo@inesc-id.pt
[2]*INESC-ID and Instituto Superior Técnico*
University of Lisbon, Portugal
samuel.mascarenhas@gaips.inesc-id.pt
[3]*INESC-ID and Instituto Superior Técnico*
University of Lisbon, Portugal
ana.paiva@inesc-id.pt

Abstract

This paper provides a short introduction to the field of machine learning for interactive pedagogical systems. Departing from different examples encountered in interactive pedagogical systems—such as intelligent tutoring systems or serious games—we go over several representative families of methods in machine learning, introducing key concepts in this field. We discuss common challenges in machine learning and how current methods address such challenges. Conversely, by anchoring our presentation on actual interactive pedagogical systems, highlight how machine learning can benefit the development of such systems.

Keywords: Interactive Pedagogical Systems, Machine Learning

1 Introduction

It is often said that good teachers are those that also learn from their students. What about applications that are designed with pedagogical aims such intelligent tutoring systems, intelligent collaborative learning environments or serious games? Should such artifacts also learn from their students? And what should they learn? How? Several examples in the literature argue that systems that support learning in humans can greatly benefit from leveraging machine learning (ML) approaches. For instance, pedagogical systems can apply ML to improve their adaptive ability or they can make use of observational data rather than just relying on theoretical guidelines. In fact, almost three decades ago J. Self [64] argued that an intelligent tutoring system (ITS) should be able to work out what a student is doing or is expected to do, and that would be possible with the use of ML techniques. B. Woolf [74] also considers that there are a set of characteristics that artificial tutors ought to enjoy: they should get to know the students' differences and thus adapt to them; they should consider students learning individually or in teams; and should have the knowledge to be taught, combined with the skills to teach it. Thus, the more we are aiming at creating technology for learning that is intelligent and adaptive, the more we should build technologies that are able to learn the content to be taught, the students

background, past knowledge, learning preferences and style, as well as the way to teach it. That is, the use of ML is without a doubt a must in pedagogical applications.

To actually be effective in fostering learning, pedagogical systems have to make predictions and learn from the actions of students in a psychologically plausible way, and as such, adapt the teaching methods, skills and knowledge to them. Yet, to build an accurate student model it is necessary to properly consider what that model is, how to represent it, to acquire it, and how adapt the teaching to it. In general, a student model can be defined as a qualitative representation of the student's behaviour in relation to the existing background knowledge, which comprises the domain theory that is being taught as well as common mistakes and errors that are held by students in that particular domain. The student model can be used by the system to keep track of the progress made in order to avoid lingering on concepts that the student has already mastered and also avoid skipping over material that the student is still struggling with. At the backbone of student modelling techniques one can find approaches like the work proposed more than three decades ago whereby to create a student model one relies on hand-crafted databases of common misconceptions that students have on the pedagogical domain, also referred as bug libraries [17]. However, constructing such libraries is a difficult and time-consuming process. Also, this type of solution becomes ineffective for teaching students with misconceptions that were not anticipated. As such, researchers begun to use machine learning approaches to solve these issues. Beginning with the ACM system [42], an induction approach was used to automatically construct models that contained both correct and buggy knowledge, however it was limited by the difficulty in starting from a complete empty model. Differently, the ASSERT system [9] used an approach based on theory refinement where a model of correct knowledge is automatically refined taking into account examples of errors made by students. A study made with 100 college students on an introductory course on the C^{++} programming language showed that the system was able to significantly improve learning gains and the students who improved the most were also the ones that the system was able to construct significantly better models. But learner modeling goes beyond capturing the learner's skills or mistakes. Several learner modelling approaches are also address the modeling of learners affective states or other attitudes, such as for example, confusion. In a study conducted with a problem-solving based ITS [10], researchers used techniques of ML to detect boredom, confusion and frustration in learners. The results obtained showed that boredom was very persistent in the interaction across learning environments and was associated with poorer learning, whereas frustration was less persistent and less associated with poorer learning.

In addition to the challenge of constructing adequate student models, a pedagogical system must also employ pedagogical strategies that guide which actions to perform while interacting with the learners. For example, if the system is able to detect the learner's level of boredom it can adapt its pedagogical strategies adequately, and even learn which strategies are more motivating for each learner and for each situation. To solve this problem, some tutoring systems apply a traditional rule-based approach that is grounded on existing pedagogical theories [40]. However, such an approach lacks adaptability as different policies might work better with different groups of learners. As such, it is possible to use ML to overcome the limitation of a fixed pedagogical policy, as exemplified by the use of reinforcement learning in Cordillera, a tutoring system that teaches introductory physics using natural language [22]. The approach consisted in first letting students interact with a version of the system that would make its pedagogical decisions in a random manner. The result of this was an exploratory corpus in which pedagogical policies were then induced through the use of reinforcement learning.

Furthermore, as technology becomes more entrenched in our educational settings, the use of gamification techniques is increasingly used [4]. Such techniques tap into the learners' intrinsic motivations, encouraging them to behave in a manner conductive of learning. In

parallel, we are also witnessing an increase in the use of "serious games" (SG) [11], that take advantage of a reward-based interaction, grounded on challenges, levels and sometimes competition. Serious games can be built to provide sequences of learning that appropriately promote knowledge acquisition in a natural and engaging fashion, often without the learner being aware that she or he is learning. Learners learn by playing games. However, it is unrealistic to expect that all knowledge needed in a game as well as all the information about the students, their differences, pedagogical approaches etc., is authored by the system's developers. So, to effectively get to know the student's differences in a serious game and thus adapt to them, the game needs to "learn" them. For example, a young yet quite skilled first grade child interacting with a serious game that supports handwriting should be given a different set of "challenges" and "rewards" from another child that barely knows how to hold a pen and is trying to master the writing of one single simple character. Or a young adult that has learned well a set of algorithms should be provided with an adequate sequence of exercises and challenges that takes into account what she has already mastered. There are several serious games designed that leverage the use of ML. One example is Crystal Island, a well known pedagogical system where the use of a supervised learning approach had a strong impact [47]. It is a narrative-centered learning environment that was designed to teach eight-grade microbiology. It was initially developed in 2011 and since then it has been used as a platform to research the impact of different AI techniques in improving learning outcomes of students. The application of machine learning in Crystal Island involved the creation of a planner that was trained using a corpus of intervention and planning decisions made by human teachers within the virtual environment. A study with 150 students using Crystal Island found that students who were guided by the trained model had significantly higher learning gains than those who were not.

The examples above are but a few illustrative examples of the potential impact that machine learning can have in education. The present paper provides a first tutorial to this broad and active field of research that is ML, aimed at researchers and engineers working on interactive pedagogical systems. We provide an overview of some of the central concepts in machine learning and introduce several representative families of machine learning problems and algorithms.

Each class of machine learning problems is introduced in the context of a real-world scenario involving an interactive pedagogical system; using such scenarios as motivation, we then go over the main ideas behind important algorithmic solutions in machine learning. We conclude the paper by providing pointers to further readings.

Nomenclature

We denote random variables (r.v.s) using upright letters, as in x or y, and instances of r.v.s as slanted letters, as in x or y. We use uppercase letters to denote functions, as in F or G, and calligraphic letters to denote sets, as in $\mathcal{X}$ or $\mathcal{U}$. Vectors are represented as bold lowercase letters. For example, $\mathbf{x}$ denotes a random vector and $\boldsymbol{x}$ an instance thereof. Except where otherwise noted, vectors are taken as *column* vectors. We write x_k and x_k to denote the kth element of the random vector $\mathbf{x}$ and vector $\boldsymbol{x}$, respectively. Matrices are represented using bold uppercase letters, as in $\mathbf{X}$. Estimated/computed entities are denoted with a hat. For example, we could write $\hat{F}$ to denote an estimated function, or $\hat{x}$ to denote an estimated value.

2 What is learning?

This section provides an overview of several core concepts in machine learning. Subsequent sections then go over several families of methods, discussing them in the context of actual applications in pedagogical interactive systems.

◇

Learning has been a central problem in artificial intelligence research from the early days of the field. Back in the 1950s, A. Samuel described machine learning as a *"field of study that gives computers the ability to learn without being explicitly programmed"*. Samuel was behind some of the earliest works in machine learning, having developed a checkers program that autonomously learned to play the game proficiently [63].

In this work, we adopt the definition of learning proposed by Mitchell [48].

Definition 1. An agent is said to *learn* from experience $\mathcal{E}$ with respect to some class of tasks $\mathcal{T}$ and performance measure P, if its performance at tasks in $\mathcal{T}$, as measured by P, improves with experience $\mathcal{E}$.

In other words, a learning problem is defined by specifying: (i) the class of tasks $\mathcal{T}$ to be learned; (ii) the experience $\mathcal{E}$ used to learn; and (iii) the criterion P against which the success of learning is to be assessed. By varying the three parameters $\mathcal{T}$, $\mathcal{E}$ and P, we obtain different types of learning problems.

2.1 Types of learning

In the continuation, we provide a taxonomy of different types of learning problems that differ on *what* they learn (i.e., the class of tasks considered), *how* they learn it (i.e., the data available for learning), and how success is measured with respect to the task (i.e., the performance measure).

The provided taxonomy should not be taken as defining crisp frontiers between the different types of learning, but rather as conveying a high-level perspective on the landscape of learning problems that can help newcomers to frame their own application scenarios with respect to the field of machine learning.

Supervised learning

The first type of learning consists of learning from examples, a problem commonly referred as *supervised learning*.

In supervised learning, the task consists in identifying an unknown relation between elements from two sets, $\mathcal{X}$ and $\mathcal{Y}$. The set $\mathcal{X}$ is usually referred as the *input space*, and $\mathcal{Y}$ is usually referred as the *output space*. It is common to assume that the target relation between inputs and outputs can be captured probabilistically, i.e., there is an unknown joint distribution μ^* over $\mathcal{X} \times \mathcal{Y}$ that perfectly describes such relation.

However, for reasons of tractability, interpretability, mathematical convenience, or other, specific learning approaches may describe the relation between $\mathcal{X}$ and $\mathcal{Y}$ in several different ways, such as

- ... a function $F : \mathcal{X} \to \mathcal{Y}$, mapping inputs to outputs. Such a function is usually known as a *prediction function*;
- ... a probabilistic model taking the form of a conditional probability distribution $p_{\mathrm{y|x}}$ assigning a probability $p_{\mathrm{y|x}}(y \mid x)$ to each output $y \in \mathcal{Y}$ that depends on the input $x \in \mathcal{X}$. Such distribution is often referred as a *discriminative model*;
- ... a probabilistic model taking the form of a joint probability distribution $p_{\mathrm{x,y}}(x, y)$, describing the probability of co-occurrence of the pair $(x, y) \in \mathcal{X} \times \mathcal{Y}$. Such a probabilistic model is often known as a *generative model*.

To learn the relation between $\mathcal{X}$ and $\mathcal{Y}$ in any of the forms discussed above, the algorithm is provided with a set $\mathcal{D}$ of input-output pairs, i.e., a set $\mathcal{D} = \{(x_n, y_n), n = 1, \ldots, N\}$, with $x_n \in \mathcal{X}$ and $y_n \in \mathcal{Y}$, where the pairs (x_n, y_n) illustrate the target relation between $\mathcal{X}$ and $\mathcal{Y}$. In particular, it is usually assumed that the pairs $(x_n, y_n) \in \mathcal{D}$ are independent samples obtained from the underlying distribution μ^*.

Finally, the performance of the learning algorithm is quantified by its ability to *predict* the output given an input. Therefore, if $p_{\mathrm{y|x}}$ denotes the input-output relation estimated by the learning algorithm—henceforth referred briefly as a learned predictor or just a predictor—its performance is usually captured by means of a *loss function* ℓ, where $\ell(x, y, p_{\mathrm{y|x}})$ measures how much (or how little) $p_{\mathrm{y|x}}$ is able to predict the desired output, y, given the input x.[1] Examples of loss functions include

- The *0-1 loss*

$$\ell_{0-1}(x, y, p_{\mathrm{y|x}}) = 1 - p_{\mathrm{y|x}}(y \mid x),$$

  which penalizes equally any predictions that do not match the target output, y. The 0-1 loss is usually applied in settings where $\mathcal{Y}$ is discrete;

- The *quadratic loss*,

$$\ell_2(x, y, p_{\mathrm{y|x}}) = \sum_{\hat{y} \in \mathcal{Y}} \|y - \hat{y}\|_2^2 \, p_{\mathrm{y|x}}(\hat{y} \mid x),$$

  which penalizes each prediction $\hat{y}$ depending on how much it deviates from the target output, y.[2] It is usually applied in settings where $\mathcal{Y}$ is continuous;

- The *negative log-likelihood*,

$$\ell_{\mathrm{NLL}}(x, y, p_{\mathrm{y|x}}) = -\log p(y \mid x),$$

  usually applied with probabilistic models and penalizes models that fail to explain the target output, y.

The overall performance of a learning algorithm corresponds to its *expected loss* or *risk*, and is given by

$$L(p_{\mathrm{y|x}}) = \mathbb{E}_{\mu^*}\left[\ell(\mathrm{x}, \mathrm{y}, p_{\mathrm{y|x}})\right],$$

where the subscript μ^* means that the expectation is taken with respect to the distribution μ^*.

Reinforcement learning

A second type of learning consists of learning by trial-and-error, a problem commonly referred as *reinforcement learning*. The typical reinforcement learning problem corresponds to a sequential decision problem where, at each discrete time step t, the learning algorithm is a decision maker/agent that

[1] The assumption that the learned relation between inputs and outputs takes the form of a conditional probability distribution $p_{\mathrm{y|x}}$ can be made without loss of generality. In fact, a prediction function F can also be represented as a conditional probability distribution $p_{\mathrm{y|x}}$ where, for each input $x \in \mathcal{X}$, $p(y \mid x) = 1$ if $F(x) = y$ and 0 otherwise. Similarly, a generative model $p_{\mathrm{x,y}}$ also implies a conditional probability distribution

$$p_{\mathrm{y|x}}(y \mid x) = \frac{p_{\mathrm{x,y}}(x, y)}{\sum_{y'} p_{\mathrm{x,y}}(x, y')}.$$

[2] We generally use summations throughout the manuscript to avoid unnecessarily cluttering the notation. We note, however, that these should be replaced by the adequate integrals when dealing with continuous variables.

- ... is provided with some piece of information, denoted as a random variable x_t taking values in some set $\mathcal{X}$. The quantity x_t is known as the *state* of the problem at time step t, and $\mathcal{X}$ is known as the *state space*;
- ... based on the state x_t, must select an *action* (possibly at random) from a set $\mathcal{A}$ of possible alternatives. The action selected by the agent at time step t is denoted as the random variable a_t, and $\mathcal{A}$ is known as the *action space*.
- ... upon selecting an action at time step t receives a (possibly random) numerical reward r_t. It is common to assume that there is an unknown time-independent function $r : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ such that

  $$\mathbb{E}\left[\mathrm{r}_t \mid \mathrm{x}_t = x, \mathrm{a}_t = a\right] = r(x, a).$$

  Such function is known as the *immediate reward function*.

Additionally, standard reinforcement learning problems and algorithms rely on a *Markov assumption*, i.e., the state at time step $t + 1$ is fully determined by the state and action at time step t by means of unknown time-independent *transition probabilities*,

$$p(x' \mid x, a) \stackrel{\text{def}}{=} \mathbb{P}\left[\mathrm{x}_{t+1} = x' \mid \mathrm{x}_t = x, \mathrm{a}_t = a\right].$$

A "rule" that prescribes a (possibly random) action to select in each state $x \in \mathcal{X}$ is known as a *policy*. Formally, a policy π can be represented as a conditional probability distribution over actions, where $\pi(a \mid x)$ denotes the probability of selecting action a in state x when following a policy π. The goal of the learning agent is thus to identify the policy that maximizes some long-term criterion formulated in terms of the reward function r.[3] Examples of possible criteria include

- The expected *average per-step reward*, where the "quality" of a policy π is given by

  $$\mathrm{AR}(\pi) = \lim_{T \to \infty} \frac{1}{T} \mathbb{E}_\pi \left[\sum_{t=0}^{T} \mathrm{r}_t\right],$$

  where the subscript π indicates that the expectation is taken with respect to trajectories obtained by following policy π;
- The expected *total discounted reward*, where the "quality" of a policy π is given by

  $$\mathrm{DR}(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t \mathrm{r}_t\right],$$

  where $\gamma \in [0, 1)$ is a *discount factor* translating the importance of rewards arising later have in comparison with rewards arising earlier.

The policy maximizing the prescribed optimality criterion is known as *optimal policy* and denoted as π^*.

Finally, in order to learn the optimal policy, the agent has available a set of sample transitions, i.e., a set $\mathcal{D} = \{(x_t, a_t, r_t, x_{t+1}), t = 0, \ldots, T\}$, obtained while following some policy π. Reinforcement learning usually takes place *online*, i.e., as the data in $\mathcal{D}$ is collected, although some alternative versions exist in which learning takes place *offline*, i.e., after the data is collected.

[3] Some works instead formulate reinforcement learning as the minimization of a long-term optimality criterion based on a *cost function* c [13]. The two formulations are equivalent.

Unsupervised learning

Finally, the third type of learning is known as *unsupervised learning*. Unlike supervised learning and reinforcement learning, in which the task is to determine a relation/mapping between two sets (input-output, state-action) and is driven by supervisory information (in the form of examples or reward signal), in unsupervised learning the learning algorithm is provided with a data set and the goal is, roughly speaking, to identify some underlying structure in such data.

From a very abstract perspective, given a set of instances from some underlying instance space $\mathcal{X}$, i.e., a set $\mathcal{D} = \{x_n, n = 1, \ldots, N\}$, the task in unsupervised learning consists of determining an *abstract description* for the data in $\mathcal{D}$. Such description can take different forms and serve different purposes, but the key point is that it builds on the regularities found in the data. Examples include

- *Clustering,* where the goal is to aggregate the data in $\mathcal{D}$ into K subsets—known as *clusters*—such that data are as similar as possible within each cluster, and as distinct as possible across different clusters. A cluster can thus be seen as an "abstraction" of the data in it. For example, Fig. 1(a) showcases an example of *k-means clustering*, a popular clustering technique. Data points belonging to different clusters are represented with different colors. The 3 large colored markers can be interpreted as "summarizing" the data in the corresponding cluster.

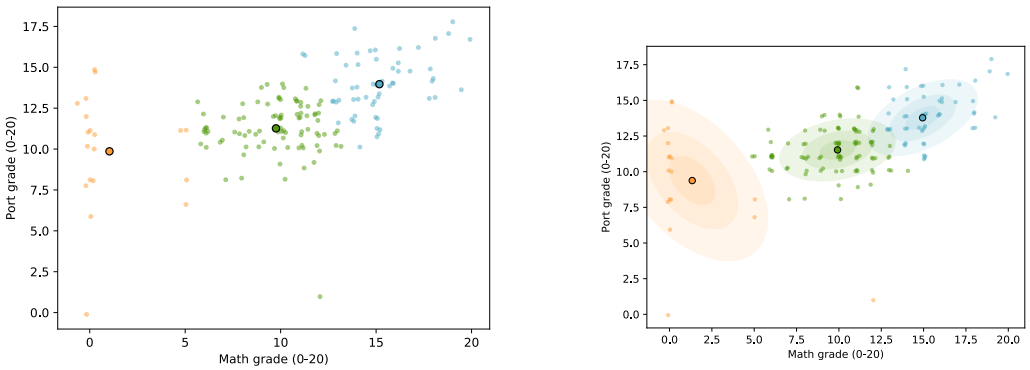
- *Density estimation,* where the goal is to identify a probability distribution p that properly describes the observed data. In a sense, the distribution p can be understood as a *generative model* of the data in $\mathcal{D}$ in the sense that it enables additional synthetic data to be produced that is similar to the one in $\mathcal{D}$. For example, Fig. 1(b) showcases an example of a *Gaussian mixture model*, where the distribution p takes the form of a mixture of Gaussian distributions. The contour lines represent the learned density, and the data points are colored according to which of three Gaussians best explains it.

- *Feature extraction and dimensionality reduction,* where the goal is to find an alternative representation for the data that is usually more compact. As an example, in *principal component analysis* (PCA), the data is represented as the linear combination of a set of uncorrelated features in such a way that the smallest number of features can explain the maximum variance in the data. For instance, the principal component for the data in Fig. 1 explains 85% of the variance in the data; if we run, for example, k-means clustering on the transformed 1-dimensional data, we obtain the exact same division in clusters observed in Fig. 1(a), indicating that, indeed, the single principal component captures most of the relevant information in the data.

Unsupervised learning techniques can often be used as a pre-processing step to other learning problems—such as supervised or reinforcement learning. In this paper, we focus mostly on supervised and reinforcement learning.

2.2 Challenges in machine learning

Each learning problem presents specific challenges that one should be aware of when developing/deploying the learning algorithms. We now provide an overview of some of the main challenges encountered in supervised and reinforcement learning.



(a) K-means clustering. (b) Gaussian mixture model.

Figure 1: *Two different ways of aggregating student data into 3 subsets. The data is a subset of the dataset of Cortez and Silva [25], and comprises the grades of* 166 *high-school students in Portugal. Each data point corresponds to a student and consists of a pair* (g_M, g_P)*, where* g_M *is the student's final grade in Math and* g_P *is the student's final grade in Portuguese. For ease of visualization and to minimize overlapping, the actual data was perturbed for plotting.*

Supervised learning

As discussed in Section 2.1, supervised learning seeks to determine a predictor $p_{\mathrm{y}|\mathrm{x}}$ that minimizes expected loss, given by

$$L(p_{\mathrm{y}|\mathrm{x}}) = \mathbb{E}_{\mu^*}\left[\ell(\mathrm{x}, \mathrm{y}, p_{\mathrm{y}|\mathrm{x}})\right]. \tag{1}$$

Unfortunately, since the underlying distribution μ^* is unknown, it is not possible to directly compute (1) for any given predictor $p_{\mathrm{y}|\mathrm{x}}$. Instead, $L(p_{\mathrm{y}|\mathrm{x}})$ is approximated as

$$\hat{L}(p_{\mathrm{y}|\mathrm{x}}) = \frac{1}{N}\sum_{n=1}^{N} \ell(x_n, y_n, p_{\mathrm{y}|\mathrm{x}}), \tag{2}$$

and most algorithms seek to identify the predictor that minimizes (2) instead of (1). The value $\hat{L}(p_{\mathrm{y}|\mathrm{x}})$ is known as *empirical risk*, and selecting the predictor $p_{\mathrm{y}|\mathrm{x}}$ that minimizes $\hat{L}(p_{\mathrm{y}|\mathrm{x}})$ is known as *empirical risk minimization* (ERM).

The ERM framework lies at the core of supervised learning. However, it presents specific challenges that should be considered carefully when designing/deploying learning algorithms:

- *"No-free-lunch" theorems.* The designation "no-free-lunch theorems" refers to a number of theoretical results by Wolpert [73]. In a nutshell, no-free-lunch theorems state that bias-free learning is not possible. The key observation is that if we assume nothing about the target input-output relation, then the data in $\mathcal{D}$ provides no information regarding such relation for inputs not in $\mathcal{D}$. Successful learning must thus rely on some assumptions regarding the target input-output relation. Such assumptions are usually known as the *inductive bias*.

- *The bias-complexity trade-off.* The inductive bias usually consists of restricting the learning algorithm to consider only predictors that belong to some specific family $\mathcal{P}$. Ideally, we would like $\mathcal{P}$ to be as representative as possible, i.e., for any distribution μ^* and loss function ℓ, there is a predictor $p_{\mathrm{y}|\mathrm{x}} \in \mathcal{P}$ for which $L(p_{\mathrm{y}|\mathrm{x}})$ is small. In this case, we say that $\mathcal{P}$ has a small bias. However, the more representative $\mathcal{P}$ is, the more complex it is to reliably identify the $p_{\mathrm{y}|\mathrm{x}} \in \mathcal{P}$ that minimizes the expected loss from the

given data. The selection of $\mathcal{P}$ thus involves a trade-off between bias and complexity, extensively studied in the field of *algorithmic learning theory* [12, 38].

- *The fitting-stability trade-off.* An important observation implicit in the no-free-lunch theorems is that a predictor may perform very well in the observed inputs, but generalize very poorly in non-observed inputs. A learning algorithm that chooses a predictor with small empirical risk but large true risk over another with larger empirical risk but smaller true risk is said to be *over-fitting*. Over-fitting thus means that the algorithm is over-sensitive to the observed data: a small change in $\mathcal{D}$ means that the algorithm will prompt the algorithm to select a completely different predictor. Methods to mitigate over-fitting—such as regularization—typically work by rendering the algorithm more stable with respect to changes in the observed data, typically at the cost of larger empirical risk [12].

All issues above revolve around a key challenge in supervised learning: that of robustly assessing the true performance of a predictor.

Reinforcement learning

As seen in Section 2.1, the goal of a reinforcement learning agent is to identify the *optimal policy*, defined as a decision rule that maximizes some function of the total reward collected by the agent. The fundamental challenges in reinforcement learning arise from the fact that the data used for learning—i.e., the set of transitions $\mathcal{D} = \{(x_t, a_t, r_t, x_{t+1}), t = 0, \ldots, T\}$—depends on the actions selected by the agent when such data is collected. We herein highlight two such challenges:

- *The exploration-exploitation trade-off.* Each transition (x_t, a_t, r_t, x_{t+1}) depends on an action a_t selected by the agent. Such action conditions both the reward r_t received by the agent at time step t, and the transition from state x_t to state x_{t+1} experienced by the agent. Unsurprisingly, successful learning will require that the agent experiments as many different actions as possible in as many different states as possible—a process known as *exploration*.

  However, exploration may steer the agent away from the most rewarding states. Additionally, even if the agent is in states where the right actions may yield high rewards, it will often be selecting sub-optimal actions while exploring, and thus miss such high rewards. Therefore, from a perspective of agent performance, exploration should be kept to a minimum, allowing the agent to use the knowledge it has gathered to maximize the received rewards—a process known as *exploitation*.

- *Temporal credit assignment.* The fact that the actions of the agent condition subsequent states means that the impact of some actions in terms of reward may not be apparent. To illustrate this observation, consider the problem depicted in Fig. 2. This problem features only 3 states, 0, A and B, and two actions, a and b. In state 0, both actions yield a reward of 0 but each leads to a different state. In states A and B both actions are equivalent.

  Suppose that the learning agent starts in state $\mathrm{x}_0 = 0$ at time step $t = 0$. The noteworthy aspect is that the action at time step $t = 0$ is the determinant factor as to whether the agent will receive a positive reward at time step $t = 1$ or not. Such difficulty—of assigning the credit for the reward at time step $t = 1$ to the action taken at time step $t = 0$—is an instance of the so-called *temporal credit assignment problem*.

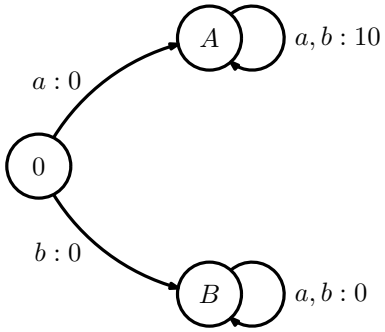


Figure 2: *Small 3-state decision problem. The circles correspond to the states, and the arrows to possible transitions between states. In this example, all transitions are deterministic. Each transition is labeled with a pair $a : r$, where a is an action and r is a reward. For example, in state 0 action a causes a transition to state A and a reward of 0, while action b causes a transition to state B and a reward of 0.*

2.3 Outline of the paper

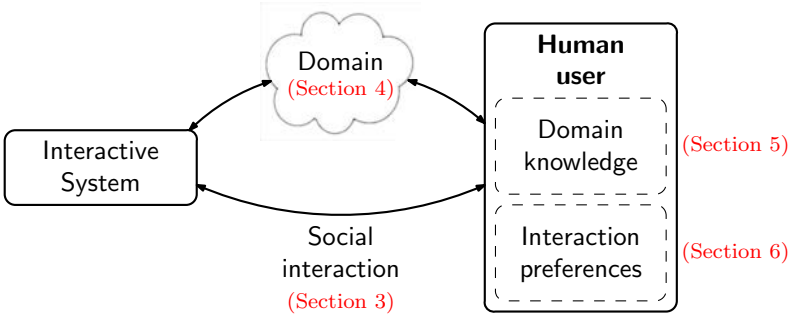
In the remainder of the paper, we illustrate different types of learning problems in the context of specific applications of interactive pedagogical systems. The overall layout of the paper is summarized in Fig. 3. We build our presentation around real-world problems that motivate the learning approaches described throughout the paper. The application scenarios roughly correspond to key elements around which the interaction between a human user and an interactive pedagogical system can be structured. We use such applications as a pretext to briefly introduce several learning algorithms, discussing their underlying principles. In particular,

- Section 3 introduces the problem of learning social behavior from human examples, in the context of human-robot interaction. In particular, given (i) a discrete set of social actions that a robot can perform when interacting with a person; and (ii) a set of examples of the use of such behaviors obtained from a human operator; we describe how the examples from the user can be used to learn adequate social behavior for the robot. We formulate this problem as a supervised learning problem with a discrete output space, and introduce two distinct approaches that can be used in this context: decision trees and k-nearest neighbors.

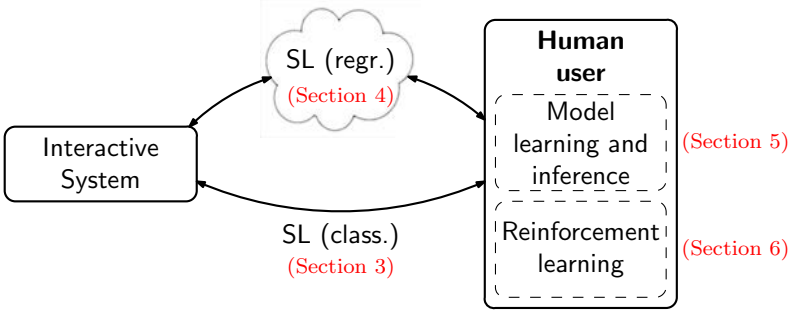
- Section 4 introduces the problem of learning complex motions (such as writing motions) from human examples, once again in the context of human-robot interaction. In particular, given a set of motion examples from humans that showcase how different letters can be handwritten, we describe how a robot may learn to perform such motions from the provided examples. We formulate this problem as a supervised learning problem with a continuous output space, and introduce two distinct approaches that can be used in this context: linear regression and neural networks.

- Section 5 introduces a distinct problem from previous sections—namely that of inferring human knowledge from incomplete data, in the context of intelligent tutoring systems. In particular, given a curriculum to be learned, we discuss how we can use data to learn a model of the student's learning process, and how such model can then be used to make inferences regarding the student's knowledge from answers to quizzes. We frame this problem as a problem of Bayesian inference and discuss how such probabilistic framework can be applied in this scenario.

- Finally, Section 6 introduces the problem of adaptation to the user in the context of human-robot interaction. In particular, given a set of distinct interaction modalities available to a robot, we describe how to optimize the interaction modality to a specific user from its feedback. We formulate this problem as a reinforcement learning problem and discuss two different approaches that can be used in this context: multi-armed bandit approaches, and Q-learning.



(a) We structure the interaction between a human user and a pedagogical interactive system in terms of the social interaction—and the user's preferences with respect to such interaction—and the domain (i.e., the task)—and the user's knowledge regarding such domain. The real-world examples used in this paper address these different elements.



(b) The different real world examples are used to motivate the introduction of different learning techniques/problems/paradigms. Classification is introduced in the context of learning social interaction; regression is introduced in the context of learning the domain; model learning is introduced in the context of learning a model of the user's knowledge; finally, reinforcement learning is introduced in the context of learning the user's preferences.

Figure 3: *Layout of the paper in terms of interaction with the human user and in terms of machine learning.*

We conclude the paper in Section 7 by providing pointers to complementary topics and further readings.

3 Learning social behavior

Consider the following scenario, explored in the context of the European Project EMOTE.[4] A robot and two students interact socially in the context of a pedagogical game. The game is a turn-based 3-player cooperative game and its goal is to create a sustainable city. The city has a number of resources available (electric power, money) that the players must manage in order to further develop the city, while maintaining healthy levels of economy, environment, and population's well-being. Each of the three players has a different role: the Mayor (in charge of the population's well-being), the Environmentalist (in charge of the city's environmental impact), and the Economist (in charge of the city's economic health).

The robot is fully autonomous, both in terms of game-play and in terms of social interaction. It acts as a peer of the human students (see Fig. 4): just like them, it is one of the players in the game and follows the adequate strategy for its role in the game. At the same

[4] http://www.emote-project.eu

Figure 4: *Interactive scenario where two students and a robot play a cooperative game on sustainability in the context of project EMOTE [66]. Photo adapted from [65].*

time, the robot also seeks to promote discussion regarding issues of sustainability, pushing the educational agenda of the project and further enhancing the pedagogical value of the game.

In order to be effective in promoting discussion, however, the robot must behave in a way that is socially acceptable within the game context. In spite of the recent advances in human-robot interaction (HRI) that facilitate the creation of social behaviors in robots [34], sustained social human-robot interaction—particular in multi-user settings—is still extremely difficult to attain. It is not surprising, thus, that many real-world HRI scenarios rely on some form of *Wizard-of-Oz paradigm*, where a human operator—hidden from the user—monitors the interaction and tele-operates the robot.

Recent works, however, propose that the operator inputs actually provide valuable information for the design of the robot's social behavior [39, 65]. In particular, the social actions selected by the operator in different situations can be interpreted as *examples* of the desired social behavior and can thus be used for *learning*. This was also the approach followed in EMOTE, where machine learning was used to generalize the social behavior driven by the human operator (the Wizard) in the construction a fully autonomous social robot.

To properly formalize the learning problem, let $\mathcal{X}$ denote the set of all possible situations in which the robot may intervene. A situation $x \in \mathcal{X}$ thus includes as varied information as the state of the game, the previous plays, the conversation held during the current game, previous conversations, the emotional state of the players, and more. The robot, however, is only able to perceive the interaction through its sensors. In other words, the robot does not observe x directly but, instead, an array of features $\boldsymbol{\phi}(x)$, where $\phi_k(x)$ is the reading of the robot's kth sensor.[5]

Additionally, let $\mathcal{Y}$ denote the set of *social actions* available to the robot. These actions comprise the different modalities of interaction that the robot can perform—gaze, gestures, and speech acts—and were constructed by observing human interaction in the exact same task. The set of possible social actions is *discrete*.

The data used in learning was collected through a *restricted perception Wizard-of-Oz* (RP-WoZ) paradigm [65], in which the wizard operator can only observe the same features that will drive the robot's behavior.[6] During the RP-WoZ interaction, a set $\mathcal{D} = \{(\boldsymbol{\phi}(x_n), y_n), n = 1, \ldots, N\}$

[5]Here we consider "sensor" in a very broad sense—essentially, any input channel of the system.

[6]The restriction of the operator's perception contributes to mitigate the *correspondence problem*, i.e., the perceptual-motor mismatch between the human operator and the robot. We refer to Sequeira et al. [65] for details.

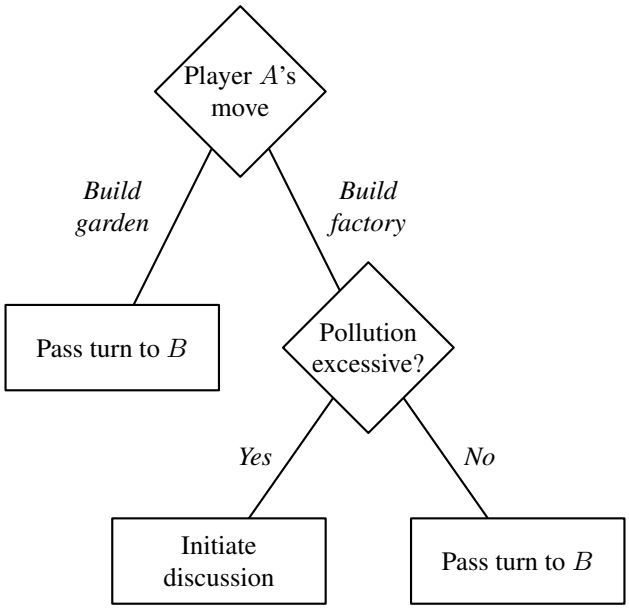


Figure 5: *Example of a very simple decision tree. The leaf nodes (rectangular-shaped boxes) correspond to possible social actions by the robot. The intermediate nodes (diamond-shaped boxes) correspond to* tests*. The decision tree expresses the rule: "If Player A builds a garden, then pass turn to player B. Otherwise, if Player A builds a factory and the pollution is excessive, initiate discussion. Otherwise, pass turn to Player B."*

of sensory-action pairs was collected, where each pair $(\phi(x_n), y_n)$ corresponds to the observed features at situation x_n, $\phi(x_n)$, and the action y_n prescribed by the operator. The relation between each situation $x \in \mathcal{X}$ and the correct social action $y \in \mathcal{Y}$ can now be learned from the examples in $\mathcal{D}$.

3.1 Learning rules

The learning problem introduced above is a supervised learning problem, in which a relation between inputs and outputs must be learned from a number of examples. In the EMOTE domain, the output set is discrete and, as such, the learning problem is usually called a *classification problem*; likewise, the resulting predictor is usually called a *classifier*.

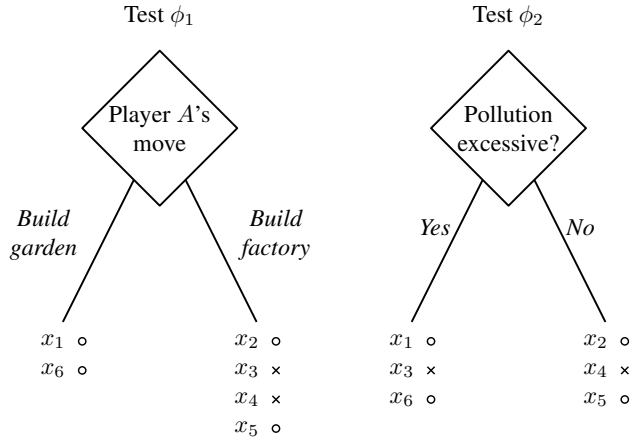
We thus start by discussing an approach that learns a discriminant function F in the form of "if…then…" rules. The conditions tested in the "if" part are usually known as *tests*, and a common approach to building one such classifier is to identify the minimum number of tests that yields an accurate prediction. Such a classifier can be represented in a tree-like form, as depicted in Fig. 5, and, for that reason, is usually called a *decision tree*.

Building the smallest decision tree that perfectly classifies the given data is known to be a NP-hard problem [35], so decision tree learning algorithms usually adopt a greedy approach when selecting the tests in each node. Such algorithms build the tree iteratively, starting from the root node. At each node, given a set of data points, the algorithm considers how each possible test (not yet used) splits the data points, and selects the test inducing the most homogeneous split.

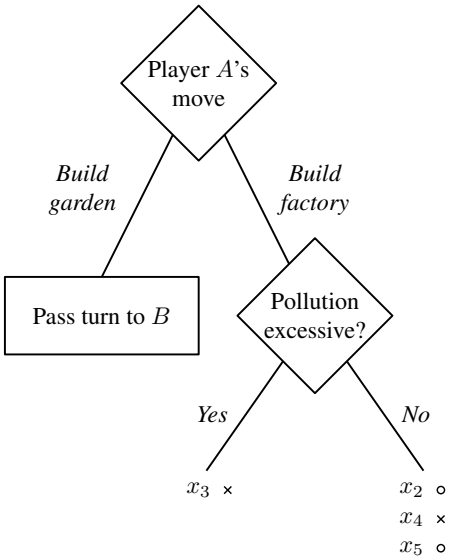
Figure 6 illustrates one simple data set that yields the decision tree of Fig. 5. In this case, the heuristic used to measure the homogeneity of the split is known as *information gain*, and measures the average entropy after the split induced by a test with respect to the entropy

	ϕ_1 (Play A)	ϕ_2 (Excess. pollution)	y (robot action)
x_1	Garden	Yes	Pass to B ($\circ$)
x_2	Factory	No	Pass to B ($\circ$)
x_3	Factory	Yes	Discuss ($\times$)
x_4	Factory	No	Discuss ($\times$)
x_5	Factory	No	Pass to B ($\circ$)
x_6	Garden	Yes	Pass to B ($\circ$)

(a) Data set comprising 6 situations (x_1 through x_6), each described by two features: whether the last play of Player A was to build a garden or a factory and whether there is excessive pollution or not. The last column corresponds to the action performed by the Wizard operator.



(b) Two possible tests in root node and induced splits of the dataset. In this case, the split induced by testing feature ϕ_1 is more homogeneous than that induced by testing feature ϕ_2, so the former is selected as the test for the root node.



(c) Given the split in the root node, the data in the left branch is perfectly classified by the "Positive rei split after the second test is considered. The non-homogeneous leaf node adopts the action of the majority.

Figure 6: *Example run of a decision tree learning algorithm yielding the tree in Fig. 5 as a result.*

before the split.[7] Other possible heuristics to measure the homogeneity of a split, such as Gini impurity or variance reduction. Decision tree learning was first introduced by Quinlan [57], and is a family of methods that are widely used, due to their simplicity and the great interpretability of the resulting models.

3.2 Learning by similarity

Decision tree learning explicitly computes a prediction function that, given any input $x \in \mathcal{X}$, provides a well-defined output $y \in \mathcal{Y}$. Such prediction function takes the form of a decision tree or, equivalently, a set of "if…then…" rules. In its search for the smallest decision tree, decision tree learning algorithms implicitly assume that a small set of such rules can perfectly capture the target input-output relation.

An alternative is to consider that "similar inputs yield similar outputs." This idea is perhaps best captured by the *k-nearest neighbors* (KNN) algorithm. The KNN algorithm does not explicitly compute an input-output relation from the provided examples. Instead, it memorizes the data in $\mathcal{D}$. Then, given a new input $x \in \mathcal{X}$, it finds the k examples in $\mathcal{D}$ that are most similar to x (the k nearest neighbors). The output for x is then computed as the majority/average of the outputs associated with such neighbors. To measure the similarity between points in input space, KNN relies on a *metric*—i.e., a function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ that measures the distance between any two inputs.

To illustrate the application of the KNN algorithm, let us consider once again the example data in Fig. 6, and let us suppose that the distance between two inputs x and x' is given by

$$d(x, x') = 0.7\mathbb{1}[\phi_1(x) \neq \phi_1(x')] + 0.3\mathbb{1}[\phi_2(x) \neq \phi_2(x')],$$

where $\mathbb{1}[p]$ takes the value 1 if p is true and 0 otherwise. Then, for any input x such that $\phi_1(x) =$ "Garden", 3NN will yield, as output, the action "Pass turn to B", since two of its nearest neighbors will be x_1 and x_6 and, in both cases, the corresponding output is "Pass turn to B". Similarly, for any input x such that $\phi_1(x) =$ "Factory" and $\phi_2(x) =$ "No", 3NN will prescribe the action "Pass turn to B". As for the remaining cases, the output of the 3NN will depend on how ties are broken in selecting the k nearest neighbors.

The KNN algorithm belongs to the class of *instance based* learning algorithms. It is widely used, mostly due to its simplicity and good results. It is worth mentioning that the performance of KNN is dependent on the metric used and on the value of k, as low values of k often lead to over-fitting. A common variant of KNN is *weighted KNN* where, given an input x, the different neighbors contribute to the computation of the output, depending on their distance to x.

4 Learning the domain

We now consider a different human-robot interaction scenario, explored in the context of the Co-Writer project.[8] The CoWriter Project explores the paradigm of "learning by teaching" by having a child teach a robot handwriting skills (see Fig. 7). Adjusting the handwriting motion of the robot, for example by exaggerating certain elements in the writing of a particular letter, it is possible to address specific difficulties that a certain child has; the child, playing the role

[7] If p_1 is the proportion of $\circ$ and p_2 the proportion of $\times$ in a set, the corresponding entropy is given by

$$H = -p_1 \log(p_1) - p_2 \log(p_2).$$

[8] `https://chili.epfl.ch/cowriter`

Figure 7: *Interactive scenario where children teach a robot the necessary motion to hand-write different letters. Photo adapted from [21].*

of the teacher, is lead to reflect on the "difficulties" of the robot and, through this process, on its own hand-writing learning process [37, 46].

Although the robotic platform used in most CoWriter studies does not have the necessary dexterity to actually write—for which reason handwriting is simulated with the help of a tablet—the project also investigates how handwriting motions can be executed by a more dexterous robotic platform. However, since most handwriting involves fine and somewhat intricate movements, the most natural approach to programming a robot to write is by using demonstrations of human handwriting motions as examples from which the desired motions can be learned.

In learning from demonstration, the robot is provided with one or more trajectories that exemplify the desired motion [7]. A trajectory can be represented as a time-indexed sequence $\{\boldsymbol{\theta}_t, t = 0, \ldots, T\}$, where each $\boldsymbol{\theta}_t$ is a vector with the position of the joints of the robot arm at time step t or, alternatively, as a sequence $\{\boldsymbol{z}_t, t = 0, \ldots, T\}$, where each $\boldsymbol{z}_t$ is a vector with the 3-D position and orientation of the end effector (see Fig. 8 for an illustration).

In order to represent an abstracted trajectory, several recent works propose the use of *motion primitives* [36, 52]. A motion primitive typically represents each component (i.e., each joint angle or 3-D coordinate) independently, using a "abstract time" (also known as phase) to coordinate the movement across all components.[9] We represent the phase as a variable $x \in [0, 1]$ and the position of the robot arm as an abstract vector $\boldsymbol{y}$, which can correspond either to joint angles or 3-D positions. The set of all trajectories thus corresponds to a data set $\mathcal{D} = \{(x_n, \boldsymbol{y}_n), n = 1, \ldots, N\}$, where each $\boldsymbol{y}_n$ is a position of the robot arm somewhere during a trajectory, and x_n is the corresponding phase. The motion to be learned corresponds to the relation between the input x and the output $\boldsymbol{y}$ implicit in the examples in $\mathcal{D}$. An appealing aspect of the adopted representation is that each component of the output can be learned independently. In the continuation, we thus focus on the learning of a single

[9] The trajectories provided by the human demonstrators are usually not time-aligned. The use of an "abstract" time representation facilitates the alignment across different trajectories.

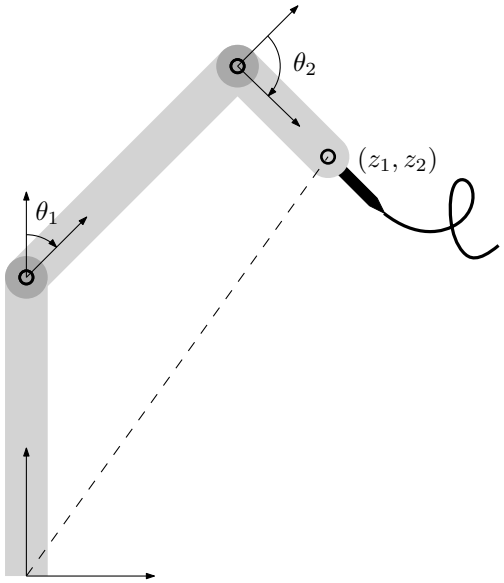


Figure 8: *Illustration of a 2-D robot arm. The configuration of the arm can be represented either as a pair (θ_1, θ_2) containing the angles of each joint, or as a pair (z_1, z_2) containing the (2-D) position of the end effector.*

component y_j.

4.1 Learning simple representations

Much like in Section 3, the problem just described corresponds to a supervised learning problem, in which a relation between inputs and outputs must be learned from a number of examples. However, in this case, the output is continuous and, as such, the learning problem is usually called a *regression problem*.

The two approaches discussed in Section 3 can readily be adapted to address regression problems. The counterpart to decision trees for regression problems are commonly known as *regression trees*, but the underlying principles behind the corresponding learning algorithms are essentially the same as those discussed in Section 3.1 [58].

Similarly, it is also possible to use KNN to address regression problems. In the case of regression problems, the prediction for input x is the mean of the output associated with the k nearest neighbors.

However, we now discuss an alternative family of approaches, in which the output takes a particularly simple form. Let $\phi_k, k = 1, \ldots, K$, denote a set of features. In *linear regression* the output is computed as a linear combination of such features, i.e.,

$$y_j = F_j(x; \boldsymbol{w}) \stackrel{\text{def}}{=} \boldsymbol{w}^\top \boldsymbol{\phi}(x),$$

where F_j is a predictor function for the jth component of the output, $\boldsymbol{w}$ is the parameter vector that determines how much each feature contributes to the computation of the output and $^\top$ is the transpose operator. Then, given the dataset $\mathcal{D}$, we can now compute the parameter vector that minimizes, for example, the quadratic loss. Following our discussion on Section 2, we have

$$\hat{L}(F_j) = \frac{1}{N} \sum_{n=1}^{N} \ell_2(x_n, y_{j,n}, F_j) = \frac{1}{N} \sum_{n=1}^{N} (y_{j,n} - \boldsymbol{w}^\top \boldsymbol{\phi}(x_n))^2. \tag{3}$$

The risk (3) is convex on $\boldsymbol{w}$ and the minimum can actually be computed analytically, yielding

$$\boldsymbol{w} = \left[\sum_{n=1}^{N} \boldsymbol{\phi}(x_n) \boldsymbol{\phi}^\top(x_n) \right]^{-1} \sum_{n=1}^{N} y_{j,n} \boldsymbol{\phi}(x_n). \tag{4}$$

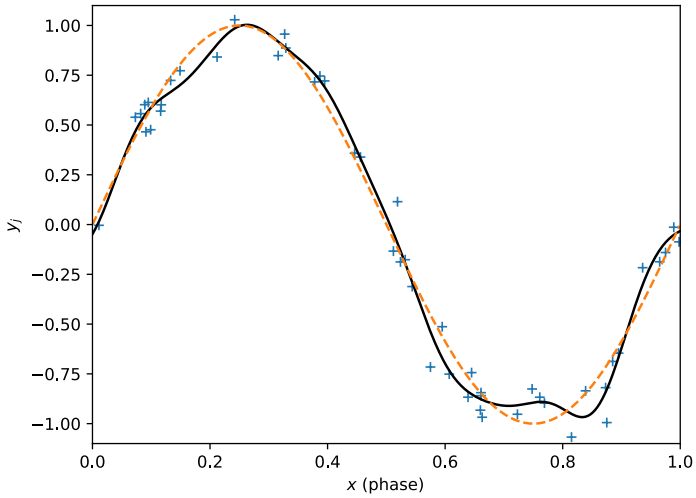
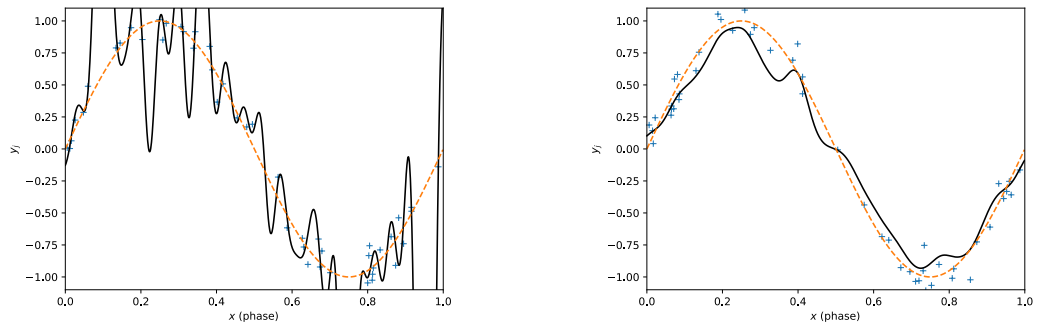


Figure 9: *Example of a linear regression. The dashed line corresponds to the original (desired) trajectory. The points marked with $+$ corresponds to the examples provided by the demonstrator, corresponding to samples from perturbed trajectories. Finally, the solid line corresponds to the trajectory learned by linear regression with $K = 15$ features.*



(a) Example of overfitting with $K = 50$ features.

(b) Effect of regularization with $K = 50$ features.

Figure 10: *Example of overfitting and regularization with linear regression for $K = 50$ features.*

The expression (4) has an appealing geometric interpretation. In fact, it consists of the orthogonal projection of the observed outputs in the linear space spanned by the features ϕ_k.

To illustrate the use of linear regression in the CoWriter scenario, consider the example depicted in Fig. 9. The target trajectory corresponds to the dashed line, but the human demonstrator provides several demonstrations that correspond to perturbed versions of such target trajectory. The learner actually observes these trajectories as a collection of (x, y) pairs, depicted with $+$ in the plot. Using a total of $K = 15$ features corresponding to Gaussian kernels uniformly distributed across $\mathcal{X}$, we then obtain the trajectory depicted as a solid line.

Linear regression, although simple, is prone to overfitting, in particular with richer sets of features. For example, if we consider a total of $K = 50$ features in the CoWriter example from Fig. 9, the error with respect to $\mathcal{D}$ is significantly smaller, but the resulting trajectory is much further from the target trajectory, as seen in Fig. 10(a). To mitigate the effect of

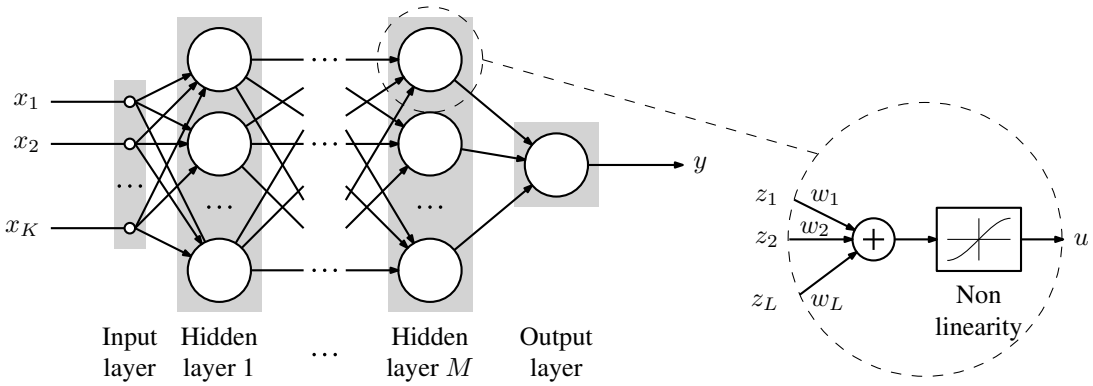


Figure 11: *Layout of a neural network. The network includes an input layer with one unit per input, a number of hidden layers, comprising a variable number of units, and an output layer, containing one layer per output. Each hidden unit is connected, through its input, to the output of all units in the preceding layer and, through its output, to the input of all units in the subsequent layer.*

overfitting in linear regression, one possibility is to use *regularization*, in which the risk in (3) is augmented with an additional term that penalizes large parameter vectors.

Figure 10(b) illustrates the use of regularization in the example from Fig. 10(a). Note how the effect of overfitting is greatly reduced. Regularization is a common approach to address overfitting, as briefly discussed in Section 2.2, since it contributes to improves the stability of learning algorithms.

4.2 Neural networks

Linear regression, as the name suggests, computes the output as a linear combination of the features describing the input. However, such linear approximations are not always convenient—either because the output is a highly nonlinear function of the input (or its features) or because there is not a clear set of features to be used to compute the output.

Neural networks provide a natural alternative to the linear regression approach just described. A neural network is a layered network of *units*, where the output of each unit in a layer is connected to all units in previous and subsequent layers. Figure 11 illustrates the general layout of a neural network. The output of a unit is a non-linear function of a linear combination of its inputs, i.e.,

$$u = G(\boldsymbol{w}^\top \boldsymbol{z}),$$

where $\boldsymbol{z}$ are the inputs to the unit, $\boldsymbol{w}$ is a vector of weights, G is a nonlinear function and u is the output of the unit.

Neural networks are *universal approximators* [26, 33], i.e., any function can be approximated arbitrarily well using a sufficiently large network.[10] Moreover, they can be trained using local search methods, i.e., the network weights can be adjusted using standard stochastic gradient descent and variations thereof. In fact, the gradient of the empirical risk with respect to the network parameters can be efficiently computed by taking advantage of the fact that a neural network is a computational graph, using a technique dubbed *backpropagation* [31].

To illustrate the application of a neural network approach to the CoWriter example from Fig. 9, we deployed a 3-layer neural network with 100, 50 and 25 units in the three hidden layers, and trained the network with the data in Fig. 9 to obtain the result in Fig. 12.

[10]Note, however, that this does not mean that such approximation can be computed from data.

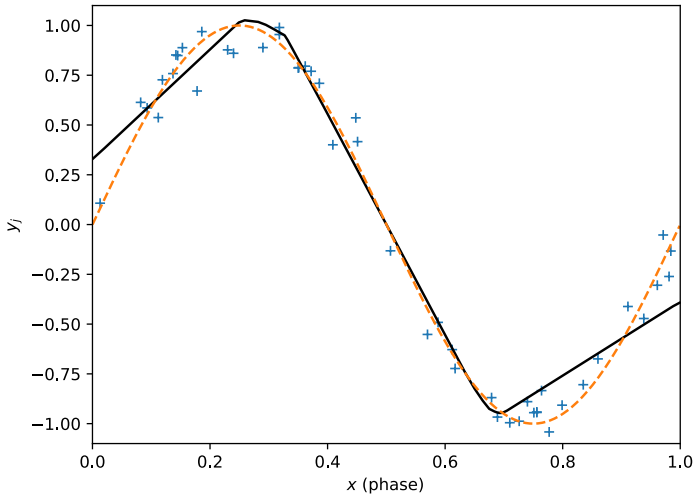


Figure 12: *Example of regression using a 3-layer neural network on the data from Fig. 9.*

Neural networks are behind many of the recent successes in machine learning, mostly due to the abundance of data and to the development of hardware that is particularly suited to the computations performed during neural network training. Such advances allow the use of networks of many layers (*deep networks*) and are behind the deep learning revolution of recent years [44]. In the context of interactive pedagogical systems, the impact of deep learning is still moderate, probably due to the significant cost of data acquisition [55]. The few examples focus mostly on domain learning—for example, perceptual processing [60], for which the advantages of deep learning are well established.

◇

We conclude this section by noting that both the linear approach discussed in Section 4.1 and the neural network model discussed in Subsection 4.2 can also be used in classification problems, by simply combining these models with a *logistic function*, given by

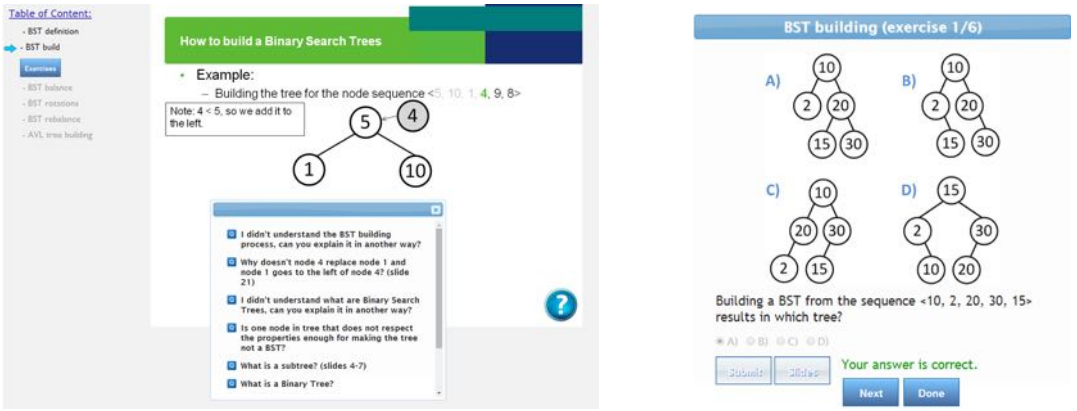
$$\sigma(u) = \frac{1}{1 + e^{-u}}.$$

For the linear model, this yields the *logistic regression classifier*; for the neural network model, this amounts to using the logistic function as the non-linearity in the output unit.

5 Inferring the user's domain knowledge

We now introduce a learning problem that, to some extent, differs from those discussed so far. To introduce such problem, we visit the concept of *intelligent tutoring systems* (ITS). ITSs seek to provide a personalized teaching experience to students, and several success stories can be found in the literature, showing that ITSs can, indeed, contribute to a better learning experience and performance [32, 61, 69].

As our running example in this section, we use the ITS system discussed in the work of Mota et al. [49]. The system was designed to assist students in their study of *Adelson-Velskii and Landis,Äô trees* (AVL trees) [3], a data structure currently taught in the Algorithms and Data Structures first-year course from the Computer Science degree at Instituto Superior



(a) Example of study material and help menu.

(b) Example of an exercise.

Figure 13: *Interface of the ITS system from [49].*

Técnico, University of Lisbon. This is a topic that students struggle with, which was part of the motivation behind the development of the ITS. The system consists of a web page in which students can access study material and exercises on 6 subtopics (see Fig. 13 for an example). The student can also access a help menu with material designed by the course instructor.

The goal of the ITS is to support students in their individual study in a personalized manner. In particular, the system should suggest the student what material she should visit next, in order to optimize her study performance. Such suggestions depend on the student's level of understanding of the different topics, and requires the system to be able to assess such level reliably, since it is not directly observable to the ITS. On the other hands, the performance of the student in the different exercises provides a good indication of how comfortable she is with each topic.

In order to build an adequate model that allows the system to reason on the student's level of understanding on the different topics, a number of students were allowed to use the system during their study. The interaction of each student with the system was registered in detail, providing—for each student—a "trace" that includes the materials visited by that particular student, the exercises completed by the student, etc. Such data can be used to *learn* a model of the student's learning process which, in turn, can be used at interaction time to *infer* the student's level of understanding of the different topics.

The different elements that the student is expected to learn are referred as *knowledge units* (KU). Let us suppose that the student is expected to learn P KUs by the end of the learning process. Each KU can be described as a binary random variable u_p, where $\mathrm{u}_p = 1$ if the student has acquired KU p, and 0 otherwise. The variables $\mathrm{u}_p, p = 1, \ldots, P$, are not directly observable and, for that reason, are called *latent variables*. Each KU builds on the material from preceding KUs. For that reason, and to simplify our presentation, we assume that the student/system does not move to KU $p+1$ prior to learning KU p. As a consequence, we can consider only the learning process of a single KU p.

For each KU p there is also a set of exercises that assess the student's understanding of that particular KU. The success or failure of a student in the exercises of KU p can be represented as a binary random vector $\mathbf{z}_p$, where the qth component $\mathrm{z}_{p,q}$ indicates whether the student succeeded on the qth exercise of KU p. Unlike the random variables u_p, the random vectors $\mathbf{z}_p$ are visible, i.e., the system "observes" how the student fared in each of the exercises for KU p.

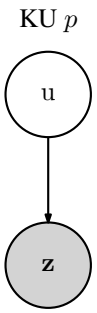


Figure 14: *Bayesian network representing the dependence between the random variable* u *and the random vector* $\mathbf{z}$*. The white cell indicates that* u *is a latent variable, while the shaded cell indicates that* $\mathbf{z}$ *is an observed variable. Additionally, the decomposition highlights the fact that the joint distribution of the several random variables can be decomposed as in* (5)*.*

5.1 Assessing the student's knowledge

The success or failure of a student in the exercises for each KU depend on the student's understanding of that KU. Formally, such dependence can be written as

$$\mathbb{P}\left[\mathrm{u} = u, \mathbf{z} = \boldsymbol{z}\right] = \mathbb{P}\left[\mathrm{u} = u\right]\mathbb{P}\left[\mathbf{z} = \boldsymbol{z} \mid \mathrm{u} = u\right], \tag{5}$$

where we dropped the subscript p to avoid excessively clutering the notation. The equality in (5) can also be graphically represented as the *Bayes diagram* of Fig. 14. This is, perhaps, the simplest representation of the relation between the student's knowledge and the observable exercise outcome. Then, upon observing $\boldsymbol{z}$, it is possible to make inferences about the latent variable u using Bayes theorem:

$$\mathbb{P}\left[\mathrm{u} = u \mid \mathbf{z} = \boldsymbol{z}\right] = \frac{\mathbb{P}\left[\mathrm{u} = u\right]\mathbb{P}\left[\mathbf{z} = \boldsymbol{z} \mid \mathrm{u} = u\right]}{\mathbb{P}\left[\mathbf{z} = \boldsymbol{z}\right]} \tag{6}$$

Let p_{u} denote the *prior distribution* regarding the student's knowledge on KU p. In other words,

$$p_{\mathrm{u}}(u) \stackrel{\text{def}}{=} \mathbb{P}\left[\mathrm{u} = u\right]$$

and translates the system's belief that the students knowledge on KU p is u before observing the outcome of the exercises. Similarly, let $p_{\mathrm{z}|\mathrm{u}}$ denote the *likelihood* of the observations given the student's knowledge. In other words,

$$p_{\mathrm{z}|\mathrm{u}}(\boldsymbol{z} \mid u) \stackrel{\text{def}}{=} \mathbb{P}\left[\mathbf{z} = \boldsymbol{z} \mid \mathrm{u} = u\right]$$

and translates the likelihood that the exercise outcome is $\boldsymbol{z}$ when the student's knowledge of KU p is u. Then, it is possible to rewrite (6) as

$$\mathbb{P}\left[\mathrm{u} = u \mid \mathbf{z} = \boldsymbol{z}\right] = \frac{p_{\mathrm{z}|\mathrm{u}}(\boldsymbol{z} \mid u)p_{\mathrm{u}}(u)}{\sum_{u'} p_{\mathrm{z}|\mathrm{u}}(\boldsymbol{z} \mid u')p_{\mathrm{u}}(u')},$$

and inferences about u can be drawn from the test outcomes $\boldsymbol{z}$ if the prior distribution p_{u} and the likelihood $p_{\mathrm{z}|\mathrm{u}}$ are known.

The model in Fig. 14, in its simplicity, disregards the fact that learning is a dynamic process, in which the student's prior knowledge, study and exercising influence its knowledge of the different KUs. To capture some of these dependencies, we can instead consider the dynamic model represented in Fig. 15, a model known as a *dynamical Bayesian network*. The diagram represents the variable dependencies across consecutive time steps.

The latent random variables u_t correspond to the student's knowledge on KU p at time step t—i.e., after interacting with the teaching materials of KU p for t time steps. Similarly,

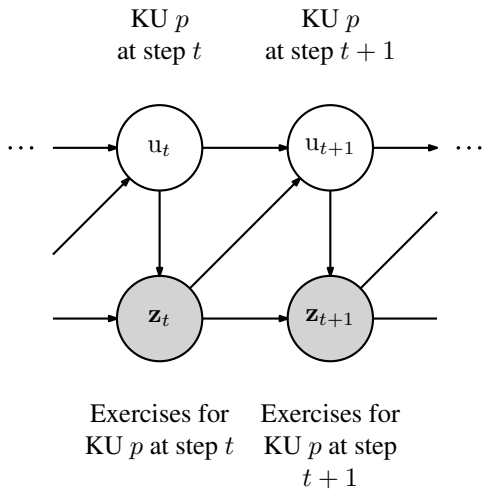


Figure 15: *Dynamic Bayesian network representing dynamics of the learning process. As before, white cells correspond to latent variables, while shaded cells indicate observed variables.*

the observed random vectors $\mathbf{z}_t$ correspond to the outcome in the exercises of KU p at time step t. The arrows indicate conditional dependencies. The diagram in Fig. 15 can be translated as

$$\begin{aligned}&\mathbb{P}\left[\mathbf{u}_{0:T} = \boldsymbol{u}_{0:T}, \mathbf{z}_{0:T} = \boldsymbol{z}_{0:T}\right] \\ &= \mathbb{P}\left[\mathrm{u}_0 = u_0\right] \mathbb{P}\left[\mathbf{z}_0 = \boldsymbol{z}_0 \mid \mathrm{u}_0 = u_0\right] \\ &\quad \cdot \prod_{t=1}^{T} \mathbb{P}\left[\mathrm{u}_t = u_t \mid \mathrm{u}_{t-1} = u_{t-1}, \mathbf{z}_{t-1} = \boldsymbol{z}_{t-1}\right] \mathbb{P}\left[\mathbf{z}_t = \boldsymbol{z}_t \mid \mathrm{u}_t = u_t, \mathbf{z}_{t-1} = \boldsymbol{z}_{t-1}\right].\end{aligned} \tag{7}$$

The model represented in Fig. 15, however, complex it may seem, has several appealing properties. First of all, it highlights the structure of dependencies (and independencies) among the many variables $\mathrm{u}_t, \mathbf{z}_t$, for $t = 0, \ldots, T$ which, in turn, facilitates the interpretation of the model.

Another important advantage of the representation in Fig. 15 is that the joint distribution $\mathbb{P}\left[\mathbf{u}_{0:T} = \boldsymbol{u}_{0:T}, \mathbf{z}_{0:T} = \boldsymbol{z}_{0:T}\right]$ can be written in a factorized way, as seen in (7). This means that, instead of representing the learning model for the student as a (potentially enormous) joint probability table, we can instead represent it using few, significantly smaller probability tables. In particular, it suffices to learn the distributions

$$p_{\mathrm{u}}(u) \stackrel{\text{def}}{=} \mathbb{P}\left[\mathrm{u}_0 = u\right], \tag{8a}$$

$$p_{\mathrm{z}|\mathrm{u}}(\boldsymbol{z} \mid u) \stackrel{\text{def}}{=} \mathbb{P}\left[\mathbf{z}_0 = \boldsymbol{z} \mid \mathrm{u}_0 = u\right], \tag{8b}$$

$$p_{\mathrm{u}|\mathrm{u},\mathrm{z}}(u' \mid u, \boldsymbol{z}) \stackrel{\text{def}}{=} \mathbb{P}\left[\mathrm{u}_t = u' \mid \mathrm{u}_{t-1} = u, \mathbf{z}_{t-1} = \boldsymbol{z}\right], \tag{8c}$$

$$p_{\mathrm{z}|\mathrm{u},\mathrm{z}}(\boldsymbol{z}' \mid u, \boldsymbol{z}) \stackrel{\text{def}}{=} \mathbb{P}\left[\mathbf{z}_t = \boldsymbol{z}' \mid \mathrm{u}_t = u, \mathbf{z}_{t-1} = \boldsymbol{z}\right]. \tag{8d}$$

Finally, inference about any of the random variables $\mathrm{u}_{p,t}$ (or sequences thereof) can be performed using an approach known as *belief propagation* [54]. The main idea is that the information about observed variables is propagated through the network, taking advantage of the factorization in (7).

	Exercise 1	Exercise 2	KU learned
x_1	Failure	Failure	No
x_2	Success	Failure	No
x_3	Success	Success	Yes
x_4	Success	Failure	Yes
x_5	Failure	Success	Yes
x_6	Failure	Failure	No

u	$p_{\mathrm{u}}(u)$
Yes	0.5
No	0.5

z_1	u	$p_{\mathrm{z}_1\mid u}(z_1 \mid u)$
Failure	Yes	0.33
Success	Yes	0.67
Failure	No	0.67
Success	No	0.33

z_2	u	$p_{\mathrm{z}_2\mid u}(z_2 \mid u)$
Failure	Yes	0.33
Success	Yes	0.67
Failure	No	1.0
Success	No	0.0

$$\mathbb{P}[\mathrm{u} = \text{“No”} \mid \mathrm{z}_1 = \text{“Failure”}, \mathrm{z}_2 = \text{“Success”}] = \rho \times 0.5 \times 0.67 \times 0 = 0$$

$$\mathbb{P}[\mathrm{u} = \text{“Yes”} \mid \mathrm{z}_1 = \text{“Failure”}, \mathrm{z}_2 = \text{“Success”}] = \rho \times 0.11 = 1.0$$

Figure 16: *Illustration of the Naive Bayes approach in the ITS scenario. The learned distributions can then be used to compute the probability that the student has learned KU p, given that it failed the first exercise but succeeded the second (ρ is a normalization constant).*

5.2 Learning models

As discussed in Section 5.1, in order to perform inference on a student's knowledge about a KU p, it is necessary to learn the corresponding models. In this section we discuss how such models can be computed, considering increasingly complex situations.

Let us start with the simple model in Fig. 14, assuming that the data collected from the preliminary interactions of users with the system corresponds to a set $\mathcal{D} = \{(u_n, \boldsymbol{z}_n), n = 1, \ldots, N\}$. Using a maximum likelihood approach, we immediately get

$$p_{\mathrm{u}}(u) = \frac{N_u}{N}, \qquad p_{\mathrm{z}|\mathrm{u}}(\boldsymbol{z} \mid u) = \frac{N_{u,\boldsymbol{z}}}{N_u},$$

where N_u is the number of times that u appears in a pair in $\mathcal{D}$, and $N_{u,\boldsymbol{z}}$ is the number of times that the pair $(u, \boldsymbol{z})$ appears in $\mathcal{D}$.

One difficulty that such approach presents is that, depending on the number of exercises in KU p, the number of possible pairs $(\boldsymbol{z}, u)$ may be quite large, and many pairs $(\boldsymbol{z}, u)$ may never occur in $\mathcal{D}$. The *Naive Bayes* approach seeks to address precisely such difficulty. Naive Bayes assumes that the components of the random vector $\mathbf{z}$ are independent given u. In other words,

$$\mathbb{P}\left[\mathbf{z} = \boldsymbol{z} \mid \mathrm{u} = u\right] = \prod_{m=1}^{M} \mathbb{P}\left[\mathrm{z}_m = z_m \mid \mathrm{u} = u\right]. \tag{9}$$

The factorization in (9) implies that, instead of learning the conditional probability $p_{\mathrm{z}|\mathrm{u}}$ from the data, it is possible to alternatively learn M much smaller distributions $p_{\mathrm{z}_k|\mathrm{u}}$, rendering the learning process significantly more data-efficient. Figure 16 illustrates the construction of the model and how it can be used to perform inference.

Let us now suppose that, instead of the model in Fig. 14, we have the model in Fig. 15, and the dataset takes the form $\mathcal{D} = \{(u_{0,n}, z_{0,n}, u_{1,n}, \boldsymbol{z}_{1,n}, \ldots, u_{T,n}, \boldsymbol{z}_{T,n}), n = 1, \ldots, N\}$.

In other words, the dataset comprises several actual learning trajectories by different students on KU p. Then, it is also possible to use a maximum likelihood approach to get

$$p_{\mathrm{u}}(u) = \frac{N_{u,0}}{N}, \qquad p_{\mathrm{z|u}}(\boldsymbol{z} \mid u) = \frac{N_{u,\boldsymbol{z},0}}{N_{u,0}},$$
$$p_{\mathrm{u|u,z}}(u' \mid u, \boldsymbol{z}) = \frac{N_{u,\boldsymbol{z},u'}}{N_{u,\boldsymbol{z}}}, \qquad p_{\mathrm{z|u,z}}(\boldsymbol{z}' \mid u, \boldsymbol{z}) = \frac{N_{\boldsymbol{z},u,\boldsymbol{z}'}}{N_{\boldsymbol{z},u}},$$

where

- $N_{u,0}$ is the number of trajectories starting with u;
- $N_{u,\boldsymbol{z},0}$ is the number of trajectories starting with $(u, \boldsymbol{z})$;
- $N_{u,\boldsymbol{z}}$ is the number of times that $\boldsymbol{z}$ succeeds u;
- $N_{\boldsymbol{z},u}$ is the number of times that u succeeds $\boldsymbol{z}$;
- $N_{u,\boldsymbol{z},u'}$ is the number of times that u' succeeds a pair $(u, \boldsymbol{z})$;
- $N_{\boldsymbol{z},u,\boldsymbol{z}'}$ is the number of times that $\boldsymbol{z}'$ succeeds a pair $(\boldsymbol{z}, u)$.

Unsurprisingly, building a model like the one in Fig. 15 requires an amount of data significantly larger than that required to learn the simpler model in Fig. 14, especially if using Naive Bayes. Complexity—both when learning and when doing inference—is the price to pay for a significantly richer model.

Finally, we conclude with the more complex—and more realistic—scenario in which the dataset takes the form $\mathcal{D} = \{(\boldsymbol{z}_{0,n}, \boldsymbol{z}_{1,n}, \ldots, \boldsymbol{z}_{T,n}), n = 1, \ldots, N\}$. In other words, the dataset contains only the values for the observed variables, and it is up to the learning algorithm to estimate the distributions in (8) in the absence of information regarding the latent variables.

In very abstract terms, we want to compute the vector of parameters of the joint distribution $p_{x,\boldsymbol{z}}$ that maximizes the likelihood of the observed data, i.e., we want to compute $\boldsymbol{\theta}^*$ such that

$$\boldsymbol{\theta}^* = \operatorname*{argmax}_{\boldsymbol{\theta}} \mathbb{P}\left[\mathbf{z}_{0:T} = \boldsymbol{z}_{0:T}; \boldsymbol{\theta}\right] = \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{\boldsymbol{u}_{0:T}} \mathbb{P}\left[\mathbf{u}_{0:T} = \boldsymbol{u}_{0:T}, \mathbf{z}_{0:T} = \boldsymbol{z}_{0:T}; \boldsymbol{\theta}\right].$$

The most common approach to address the optimization problem above is perhaps the *expectation maximization (EM) algorithm*. To provide an intuition over the inner workings of EM, we start by noting that each vector $\boldsymbol{\theta}$ defines a joint distribution $\mathbb{P}\left[\mathbf{u}_{0:T} = \boldsymbol{u}_{0:T}, \mathbf{z}_{0:T} = \boldsymbol{z}_{0:T}; \boldsymbol{\theta}\right]$ over $\mathbf{u}_{0:T}$ and $\mathbf{z}_{0:T}$. Then, given a sequence $\boldsymbol{z}_{0:T}$, we can compute the distribution

$$p_{\mathbf{u}|\mathbf{z}}(\boldsymbol{u}_{0:T} \mid \boldsymbol{z}_{0:T}) \stackrel{\text{def}}{=} \mathbb{P}\left[\mathbf{u}_{0:T} = \boldsymbol{u}_{0:T} \mid \mathbf{z}_{0:T} = \boldsymbol{z}_{0:T}; \boldsymbol{\theta}\right]. \tag{10}$$

In a nutshell, the EM algorithm departs from an initial setting for the parameters, $\boldsymbol{\theta}_0$, and iteratively optimizes the parameters of the model by repeating the following two steps:

- *The E-step:* At iteration m, given the current estimate $\boldsymbol{\theta}_m$ for the model parameters, we compute the distribution $p_{\mathbf{u}|\mathbf{z}}$ using (10) and use it to compute the expected log-likelihood of the observed data as a function of the model parameters, i.e.,

$$Q(\boldsymbol{\theta}) = \sum_{\boldsymbol{u}_{0:T}} \log \mathbb{P}\left[\mathbf{z}_{0:T} = \boldsymbol{z}_{0:T} \mid \mathbf{u}_{0:T} = \boldsymbol{u}_{0:T}; \boldsymbol{\theta}\right] p_{\mathbf{u}|\mathbf{z}}(\boldsymbol{u}_{0:T} \mid \boldsymbol{z}_{0:T}).$$

- *The M-step:* Given the expected log-likelihood of the observed data computed in the E-step, we now compute a new estimate $\boldsymbol{\theta}_{m+1}$ as

$$\boldsymbol{\theta}_{m+1} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, Q(\boldsymbol{\theta}).$$

The EM algorithm can roughly be interpreted as follows: in the E-step, it computes the "expected trajectory" for the latent variables, given the current estimate of the model parameters. Then, in the M-step, it computes the parameters that maximize the likelihood of the observed data given the "expected trajectory" of the latent variables. EM was originally proposed by Dempster et al. [27] and is widely used in machine learning, largely due to its solid convergence guarantees. An important note is that, in the practical situations where it is difficult to explicitly work with the distribution in (10), EM can be combined with sampling methods.

◇

The particular scenario discussed herein—namely that of learning a generative model for the data in the presence of unobserved data—is closer to unsupervised learning, since we do not have explicit examples illustrating an input-output relation to be learned. However, the algorithmic ideas discussed are general and can be applied whenever a generative model of the data is to be learned, in the context of supervised learning or other.

Both models in Figures 14 and 15 fall under the broad category of *graphical models*. Graphical models provide a concise and interpretable representation for complex probabilistic models, and a vast range of tools exist for learning and inference [41].

6 Assessing the user's preferences

To conclude our machine learning tutorial, we introduce the following scenario, explored in the context of the European Project LIREC.[11] LIREC investigates the creation of sustained long-term interactions between humans and robots. One scenario in which long-term interactions were investigated involved a robot that acted as a chess tutor (see Fig. 17). Teaching takes place by having the robot play chess games against the child who is learning to play. The robotic platform is, however, relatively limited, which renders the social interaction relatively simpler than that considered in the EMOTE scenario discussed in Section 3. Nevertheless, the robot provides different types of feedback on the gameplay, including strategy tips, positive reinforcement, and emotional empathy [45].

It is worth noting that the ability of the robot to exhibit empathic behavior towards the child is particularly relevant in the process of establishing sustained long-term interactions. The ability to recognize and respond to a student's affective state was found to facilitate the creation and development of social relationships [6] and lead to more efficient teaching [24]. The creation of empathic behavior in the context of the LIREC scenario just described relies on two key components: (i) an assessment of the affective state of the child; and (ii) the ability to respond emotionally and adapt its behavior to that affective state.

Regarding (i), the robot uses both context information (the current state of the game, the evolution of the game, the most recent plays, etc.) and behavior information to assess the affective state of the child. The behavior information is captured through the robot's sensors, and include the child's gaze direction, facial expressions, gestures, etc.

As for (ii), the robot responds emotionally to moves played by the child, displaying a facial expression that matches the move just performed. Additionally, when the robot realizes

[11] http://lirec.eu/project

Figure 17: *Interactive scenario where a child learns how to play chess by playing against an iCat robot. Photo adapted from [19].*

that the child is experiencing a negative feeling—most likely caused by a bad evolution of the game—the robot exhibits one of the following empathic strategies [45]:

1. Provide encouraging comments (e.g., "Don't be sad, you can still recover").
2. Provide feedback on the child's most recent play, eventually allowing her to repeat;
3. Suggest a play for the child's next turn;
4. Intentionally play a bad move.

All these strategies proved successful in improving the user's affective state in different circumstances. However, not all users respond equally well to all the above strategies, and we would like the robot to be able to adapt its selection to the user's preferences.

Let $\mathrm{x}_t \in \mathcal{X}$ denote the interaction state after play t. Such state includes, among other things, the current game context and the child's affective state as perceived by the robot. Let $\mathcal{A}$ denote the set of empathic strategies available to the robot. Upon performing an empathic actions a_t, the robot typically influences the child's affective state. Such influence is sometimes positive—when the child appreciates the robot's action—but may be neutral or negative—for example, some children feel some of the empathic actions as "demeaning" to them as players.

We write x_{t+1} to denote the interaction state of the child after the robot's empathic action, and let $\mathrm{r}_t \in \{-1, 0, 1\}$ denote the valence of the child's response: $\mathrm{r}_t = 1$ indicates that the child's response was positive; $\mathrm{r}_t = 0$ indicates a neutral reaction and $\mathrm{r}_t = -1$ indicates a negative response. Using the information collected during the interaction—i.e., the set $\mathcal{D} = \{(x_t, a_t, r_t, x_{t+1}), t = 1, \ldots\}$, we can adapt the robot's action selection strategy to the child's preference, effectively learning how to empathically respond to each particular user.

6.1 Learning simple decisions

In addressing the learning problem faced by the empathic robot, let us first consider a very simple approach and assume that the best action for a given child is constant across the time. In other words, the interaction state is immaterial for the selection of the empathic action, and the robot needs only to identify the best empathic action for the present child.

In order to do so, the robot needs to experiment the different actions available, assessing the child's response to each action. However, it should minimize the amount of experimentation: as soon as the best action is identified, the robot should follow it, in order to achieve the best possible interaction. Deciding what is the right amount of "experimentation" corresponds to the exploration vs exploitation trade-off discussed in Section 2.2, in the context of reinforcement learning.

In fact, under the assumption that the best action does not depend on the interaction state, the resulting reinforcement learning problem is known as a *stochastic multi-armed bandit*, first introduced in the work of Robbins [62]. In a stochastic multi-armed bandit problem, a decision-maker is provided with a finite set $\mathcal{A}$ of possible actions. Each action has an associated expected reward $r(a)$ that the decision-maker does not know. The decision-maker must then experiment the different actions and observe the received reward, using these noisy rewards to estimate the best action (the one with the highest expected reward).

One common approach to address the stochastic multi-armed bandit problem is the *upper confidence bound (UCB) algorithm* [8]. In UCB, the decision-maker starts by playing each action once, keeping track of the average reward associated with each action $a \in \mathcal{A}$. At every subsequent step N, the decision-maker computes a high-probability confidence interval for each value $r(a)$ of the form

$$[\hat{r}_N(a) - \text{margin}_N, \hat{r}_N(a) + \text{margin}_N(a)],$$

where $\hat{r}_N(a)$ denotes the average reward associated with action a after N steps. Then, according to UCB, the decision-maker should select the action a^* such that

$$a^* = \underset{a \in \mathcal{A}}{\text{argmax}} \left[\hat{r}_N(a) + \text{margin}_N(a)\right].$$

UCB, as many multi-armed bandit algorithms, relies on the principle of "optimism in the face of uncertainty": the decision-maker will not select the action with the current highest reward, but the action with the largest potential of yielding the highest reward. UCB is a *no regret algorithm*, meaning that, in the limit, the average reward per step converges to that of the optimal action. Figure 18 illustrates the application of UCB in the LIREC scenario.

6.2 Learning by with context information

The multi-armed bandit formalism is, perhaps, the simplest instantiation of a reinforcement learning problem, showcasing the exploration vs exploitation trade-off in its barest form. However, it fails to take into consideration several aspects that are central in many real-world scenarios—including the LIREC scenario previously discussed. In particular, stochastic multi-armed bandits disregard

- ...the fact that the reward received by the decision-maker often depends both on the action and on the context; therefore, the best action may vary, depending on the context;
- ...the fact that the actions of the decision-maker often induce changes in the context; therefore, the best action should balance immediate vs long-term rewards.

A more general approach to the full reinforcement learning problem should thus take into consideration the context at each moment—captured by the state x_t—and the long-term impact of actions—captured by whichever optimality criterion should be optimized.

Let us suppose, for example, that the decision maker adopts the expected total discounted reward, introduced in Section 6.2, as an optimality criterion. We can evaluate a policy π

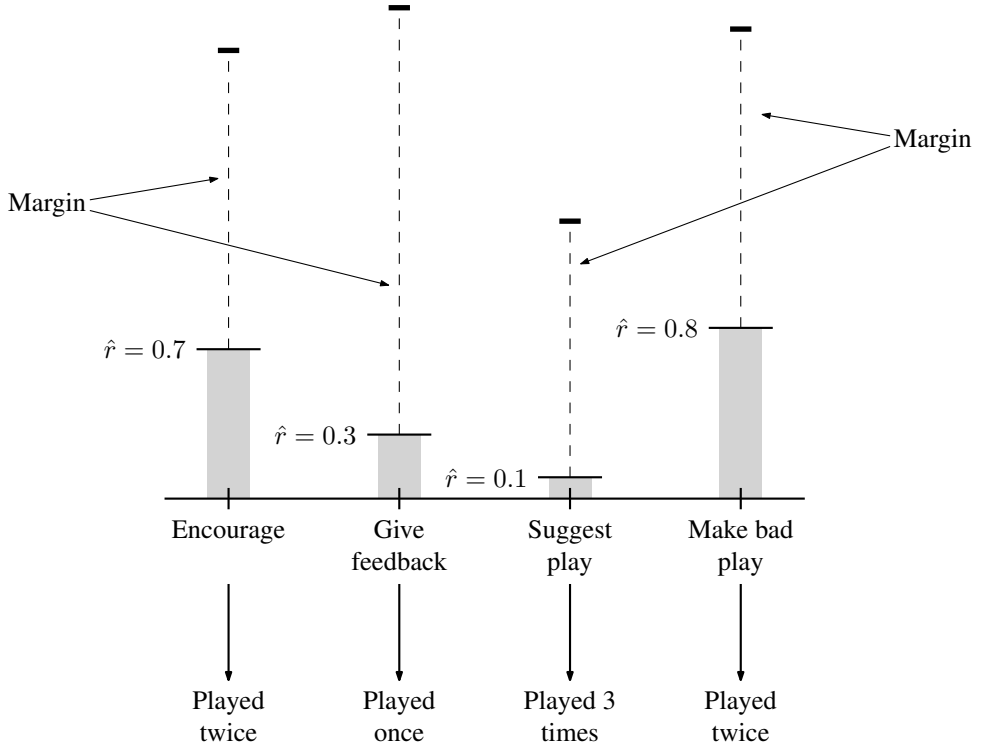


Figure 18: *Illustration of the UCB algorithm in the LIREC scenario. In this case, action "Give feedback" will be selected because, although it has a low average reward thus far, it has been played a small number of times and has, therefore, a large potential for yielding high reward.*

according to this criterion and define the *value of* π at a state x as the quantity

$$V^{\pi}(x) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t \mathrm{r}_t \mid \mathrm{x}_0 = x\right]. \tag{11}$$

V^{π} is a real-valued function defined over $\mathcal{X}$ and, for any given x, the value $V^{\pi}(x)$ is precisely the expected total discounted reward obtained by the decision maker if it starts at state x at time step 0 and always selects its actions according to policy π. Given any policy π, the function $V^{\pi} : \mathcal{X} \to \mathbb{R}$ can be computed using dynamic programming [56]. The ultimate goal of the decision-maler is, thus, to compute a policy π^* such that, for all $x \in \mathcal{X}$ and every other policy π,

$$V^{\pi^*}(x) \geq V^{\pi}(x).$$

That one such policy exists is a well-established result in the reinforcement learning theory [56]. Moreover, a policy π^* is optimal if and only if it verifies

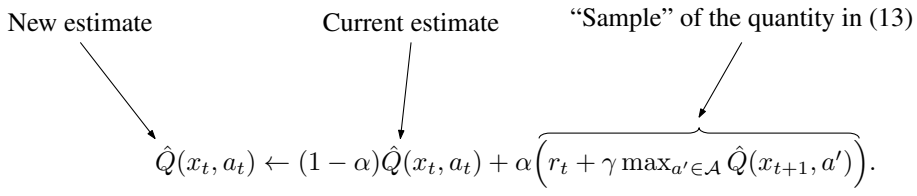
$$\pi^*(a \mid x) > 0 \text{only if} a \in \operatorname{argmax} Q^*(x, a).$$

where Q^* is the solution to the recursive equation

$$Q^*(x,a) = \mathbb{E}_{\mathrm{y}\sim p(x,a)}\left[r(x,a) + \gamma \max_{a' \in \mathcal{A}} Q^*(\mathrm{y}, a')\right], \text{for all } (x,a) \in \mathcal{X} \times \mathcal{A}. \tag{12}$$

In other words, a policy is optimal as long as it only prescribes actions that maximize Q^*, and the decision-making problem thus reduces to computing Q^*.

To address the reinforcement learning problem in its full generality, we may thus leverage the recursive relation in (12) to obtain the well-known *Q-learning algorithm*. Q-learning maintains, for each $(x, a) \in \mathcal{X} \times \mathcal{A}$, an estimate $\hat{Q}(x, a)$ for $Q^*(x, a)$. Then, upon observing a transition (x_t, a_t, r_t, x_{t+1}), Q-learning updates the Q-values associated with the pair (x_t, a_t) as



$$\hat{Q}(x_t, a_t) \leftarrow (1 - \alpha)\hat{Q}(x_t, a_t) + \alpha\overbrace{\Big(r_t + \gamma \max_{a' \in \mathcal{A}} \hat{Q}(x_{t+1}, a')\Big)}. \tag{13}$$

The scalar α is a positive step-size. Q-learning was originally proposed by Watkins [71] and is among the most widely known reinforcement learning algorithms, due both to its simplicity and its strong theoretical properties.

7 Discussion and further reading

With our discussion of reinforcement learning we conclude our overview of machine learning. The presentation herein is neither technically too detailed nor comprehensive, as it aims at providing only a first introduction to the vast field that is machine learning. The literature in machine learning is vast, and includes many very good books for all levels. For example, the books of Alpaydin [5] and [28] provide non-technical overviews of the field of machine learning. The book of Mitchell [48] or, more recently, that of Alpaydin [5], are excellent introductory texts to machine learning that cover the different types of learning discussed in this paper. The books by Flach [29] and Bishop [14] are also excellent texts, although mostly focused on supervised learning. Finally, the books by Murphy [51] and Ben-David and Shalev-Shwartz [12] are somewhat more advanced, the first providing a very up-to-date and comprehensive overview of the field, and the second focusing more on the theory of machine learning.

We have motivated and grounded our presentation on real-world problems from interactive pedagogical systems, both to convey an idea about the class of problems for which different approaches are more adequate and to facilitate the introduction of simple examples. The application scenarios depicted herein are merely illustrative, and by no means do they exhaust the wide range of applicability of the methods discussed in this paper.

Conversely, the methods discussed herein seek to provide the intuition behind several representative families of machine learning algorithms, and do not expect to cover even a small part of this vast field of research. For example,

- The decision tree learning algorithm in Section 3.1 is, perhaps, the simplest algorithm in that family, corresponds to the ID3 algorithm by Quinlan [57]. Several extensions exist that address specific limitations of ID3, including C4.5 [59], CART [16], among others. Decision trees have also been used as the base predictor upon which more complex predictors are built, including random forests [15] and gradient tree boosting [30].

- As discussed in Section 3.2, KNN falls into the broad category of instance-based learning, and is closely related to the notion of *case-based reasoning* [1]. In a sense, it is also closely related to support vector machines [70] and other kernel methods, which compute the output for an input based on how similar the query point is to other points in the dataset.

- The brief mention to neural networks in Section 4.2 fails to do justice to the dimension and relevance of this class of models in machine learning. Neural networks have played a particularly prominent role in machine learning with the raise of deep learning. Recent references to neural networks include the survey paper of LeCun et al. [44] and the book of Goodfellow et al. [31].

- Section 5.1 provides a brief introduction to graphical models. The book by Koller and Friedman [41] provides a very complete overview of graphical models, including inference and learning in this class of models.

- Section 6 introduces the topic of reinforcement learning. Reinforcement learning is also a vast topic and several good references exist, including the books by Sutton and Barto [67], Bertsekas and Tsitsiklis [13] and Szepesvári [68]. For a detailed discussion on bandit algorithms and related sequential prediction problems, we refer to the works of Cesa-Bianchi and Lugosi [20] and Lattimore and Szepesvári [43].

We conclude by pointing out several off-the-shelf tools that can be used to deploy learning in an interactive pedagogical system of interest. In terms of supervised learning, WEKA is a well established, stand alone framework for machine learning and data mining [72]. The SCIKIT-LEARN Python library includes a vast array of ML models and methods and several texts can be found that provide a hands-on introduction to ML using this library [50]. Similarly, R includes a large number of data-processing facilities and is also a popular tool for data-analysis and machine learning [18].

The deep learning revolution also brought a number of specialized libraries for neural network deployment, such as TENSORFLOW [2] and PYTORCH [53], as well as higher-level APIs such as KERAS [23].

Acknowledgements

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013, through the project AMIGOS (PTDC/EEISII/7174/2014), the project RAGE (Ref. H2020-ICT-2014-1/644187), and the Carnegie Mellon Portugal Program and its Information and Communications Technologies Institute, under project CMUP-ERI/HCI/0051/2013.

References

[1] A. Aamodt and E. Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications*, 7(1):39–52, 1994.

[2] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL *https://www.tensorflow.org/*.

[3] G. Adelson-Velsky and E. Landis. An algorithm for the organization of information. *Doklady Akademii Nauk USSR*, 146(2):263–266, 1962.

[4] J. Agapito and M. Mercedes Rodrigo. Investigating the impact of a meaningful gamification-based intervention on novice programmers' achievement. In *International Conference on Artificial Intelligence in Education*, pages 3–16. Springer, 2018. *https://doi.org/10.1007/978-3-319-93843-1_1*

[5] E. Alpaydin. *Machine Learning: The New AI*. MIT Press, 2016.

[6] C. Anderson and D. Keltner. The role of empathy in the formation and maintenance of social bonds. *Behavioral and Brain Sciences*, 25(1):21–22, 2002.

[7] B. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 27:469–483, 2009. *https://doi.org/10.1016/j.robot.2008.10.024*

[8] P. Auer, N. Cesa-Bianchi, and P. Fisher. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256, 2002. *https://doi.org/10.1023/A:1013689704352*

[9] P. Baffes and R. Mooney. Refinement-based student modeling and automated bug library construction. *J. Artificial Intelligence in Education*, 7:75–116, 1996.

[10] R. Baker, S. D'Mello, M. Mercedes Rodrigo, and A. Graesser. Better to be frustrated than bored: The incidence, persistence, and impact of learners' cognitive–affective states during interactions with three different computer-based learning environments. *International Journal of Human-Computer Studies*, 68(4):223–241, 2010. *https://doi.org/10.1016/j.ijhcs.2009.12.003*

[11] F. Bellotti, B. Kapralos, K. Lee, P. Moreno-Ger, and R. Berta. Assessment in and of serious games: an overview. *Advances in Human-Computer Interaction*, 2013:1, 2013. *https://doi.org/10.1155/2013/136864*

[12] S. Ben-David and S. Shalev-Shwartz. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[13] D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[14] C. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[15] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. *https://doi.org/10.1023/A:1010933404324*

[16] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Chapman and Hall/CRC Press, 1984.

[17] J. Brown and K. Van Lehn. Repair theory: A generative theory of bugs in procedural skills. *Cognitive Science*, 4(4):379–426, 1980. *https://doi.org/10.1207/s15516709cog0404_3*

[18] S. Burger. *Introduction to Machine Learning with R*. O'Reilly, 2018.

[19] G. Castellano, I. Leite, A. Paiva, and P. McOwan. Affective teaching: Learning more effectively from empathic robots. *Awareness Magazine: Self-Awareness in Autonomic Systems*, 2012.

[20] N. Cesa-Bianchi and G. Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006. *https://doi.org/10.1017/CBO9780511546921*

[21] S. Chandra, R. Paradeda, H. Yin, P. Dillenbourg, R. Prada, and A. Paiva. Do children perceive whether a robotic peer is learning or not? In *Proc. 2018 ACM/IEEE Int. Conf. Human-Robot Interaction*, pages 41–49, 2018.

[22] M. Chi, K. Vanlehn, D. Litman, and P. Jordan. Empirically evaluating the application of reinforcement learning to the induction of effective and adaptive pedagogical strategies. *User Modeling and User-Adapted Interaction*, 21(1-2):137–180, 2011. *https://doi.org/10.1007/s11257-010-9093-1*

[23] F. Chollet et al. Keras. *https://keras.io*, 2015.

[24] B. Cooper, P. Brna, and A. Martins. Effective affective in intelligent systems – Building on evidence of empathy in teaching and learning. In A. Paiva, editor, *Affective Interactions*. Springer, 1999.

[25] P. Cortez and A. Silva. Using data mining to predict secondary school student performance. In *Proc. 5th Future Business Technology Conference*, pages 5–12, 2008.

[26] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2:303–314, 1989. *https://doi.org/10.1007/BF02551274*

[27] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. Royal Statistical Society B*, 39(1):1–38, 1977.

[28] P. Domingos. *The Master Algorithm*. Basic Books, 2015.

[29] P. Flach. *Machine Learning*. Cambridge University Press, 2012. *https://doi.org/10.1017/CBO9780511973000*

[30] J. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232, 2001.

[31] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[32] A. C. Graesser, R. Vasile, and D. Sidney. *AutoTutor: learning through natural language dialogue that adapts to the cognitive and affective states of the learner*. 2008.

[33] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4: 251–257, 1991. *https://doi.org/10.1016/0893-6080(91)90009-T*

[34] C. Huang and B. Mutlu. Robot behavior toolkit: Generating effective social behaviors for robots. In *Proc. 7th ACM/IEEE Int. Conf. Human-Robot Interaction*, pages 25–32, 2012.

[35] L. Hyafil and R. Rivest. Constructing optimal binary decision trees is NP-Complete. *Information Processing Letters*, 5(1):15–17, 1976. *https://doi.org/10.1016/0020-0190(76)90095-8*

[36] A. Ijspeert, J. Nakanishi, P. Pastor, H. Hoffmann, and S. Schaal. Dynamical movement primitives: Learning attractor models for motor behaviors. *Neural Computation*, 25(2):328–373, 2013. *https://doi.org/10.1162/NECO_a_00393*

[37] A. Jacq, S. Lemaignan, F. Garcia, P. Dillenbourg, and A. Paiva. Building successful long child-robot interactions in a learning context. In *Proc. 11th ACM/IEEE Int. Conf. Human-Robot Interaction*, pages 239–246, 2016.

[38] M. Kearns and U. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994.

[39] W. Knox, S. Spaulding, and C. Breazeal. Learning social interaction from the Wizard: A proposal. In *Workshops at 28th AAAI Conf. on Artificial Intelligence*, 2014.

[40] K. Koedinger and J. Anderson. Intelligent tutoring goes to school in the big city. *Int. J. Artificial Intelligence in Education*, 8:30–43, 1997.

[41] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[42] P. Langley and S. Ohlsson. Automated cognitive modeling. In *Proc. 4th AAAI Conf. Artificial Intelligence*, pages 193–197, 1984.

[43] T. Lattimore and C. Szepesvári. *Bandit Algorithms*. Cambridge University Press, 2018 (to appear).

[44] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436–444, 2015. *https://doi.org/10.1038/nature14539*

[45] I. Leite, A. Pereira, G. Castellano, S. Mascarenhas, C. Martinho, and A. Paiva. Modelling empathy in social robotic companions. In L. Ardissono and T. Kuflik, editors, *Advances in User Modeling*. Springer, 2011.

[46] S. Lemaignan, A. Jacq, D. Hood, F. Garcia, A. Paiva, and P. Dillenbourg. Learning by teaching a robot: The case of handwriting. *Robotics and Automation Magazine*, 23(2):56–66, 2016.

[47] J. Lester, E. Ha, S. Lee, B. Mott, J. Rowe, and J. Sabourin. Serious games get smart: Intelligent game-based learning environments. *AI Magazine*, 34(4):31–45, 2013.

[48] T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[49] P. Mota, F. S. Melo, L. Coheur, and M. Eskenazi. Modeling students self-studies behaviors. In *Proc. 14th Int. Conf. Autonomous Agents and Multiagent Systems*, pages 1521–1528, 2015.

[50] A. Müller and S. Guido. *Introduction to Machine Learning with Python*. O'Reilly, 2016.

[51] K. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[52] A. Paraschos, C. Daniel, J. Peters, and G. Neumann. Using probabilistic movement primitives in robotics. *Autonomous Robots*, 42(3):1–23, 2017.

[53] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. In *Proc. NIPS 2017 Workshop on Automatic Differentiation*, 2017.

[54] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[55] H. Pierson and M. Gashler. Deep learning in robotics: A review of recent research. *Advanced Robotics*, 31(16):821–835, 2017.

[56] M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2005.

[57] J. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[58] J. Quinlan. Learning with continuous classes. In *Proc. 5th Australian Joint Conf. Artificial Intelligence*, pages 343–348, 1992.

[59] J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[60] A. Qureshi, Y. Nakamura, Y. Yoshikawa, and H. Ishiguro. Robot gains social intelligence through multimodal deep reinforcement learning. In *Proc. 16th IEEE-RAS Int. Conf. Humanoid Robots*, pages 745–751, 2016.

[61] S. Ritter, J. Anderson, K. Koedinger, and A. Corbett. Cognitive tutor: Applied research in mathematics education. *Psychonomic Bulletin & Review*, 14(2):249–255, 2007.
[62] H. Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535, 1952.
[63] A. Samuel. Some studies in machine learning using the game of checkers. *IBM J. Research and Development*, 3(3):210–229, 1959. Reprinted in IBM J. Res. Devel. 44:1/2, pp. 206-226, 2000.
[64] J. Self. Bypassing the intractable problem of student modelling. *Intelligent tutoring systems: At the crossroads of artificial intelligence and education*, 41:1–26, 1990.
[65] P. Sequeira, P. Alves-Oliveira, T. Ribeiro, E. Di Tullio, S. Petisca, F. S. Melo, G. Castellano, and A. Paiva. Discovering social interaction strategies for robots from restricted-perception Wizard-of-Oz studies. In *Proc. 11th ACM/IEEE Int. Conf. Human-Robot Interaction*, pages 197–204, 2016.
[66] S. Serholt, W. Barendregt, T. Ribeiro, G. Castellano, A. Paiva, A. Kappas, R. Aylett, and F. Nabais. EMOTE: Embodied-perceptive tutors for empathy-based learning in game environment. In *Proc. 7th Eur. Conf. Games Based Learning*, pages 790–792, 2013.
[67] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.
[68] C. Szepesvári. *Algorithms for Reinforcement Learning*. Morgan & Claypool Publishers, 2010.
[69] K. Vanlehn, C. Lynch, K. Schulze, J. Shapiro, R. Shelby, L. Taylor, D. Treacy, A. Weinstein, and M. Wintersgill. The Andes physics tutoring system: Lessons learned. *J. Artificial Intelligence in Education*, 15(3):147–204, 2005.
[70] V. Vapnik. *Statistical Learning Theory*. Wiley, 1998.
[71] C. Watkins. *Learning from delayed rewards*. PhD thesis, King's College, University of Cambridge, May 1989.
[72] I. Witten, E. Frank, and M. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2011.
[73] D. Wolpert. The lack of *a priori* distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390, 1996. *https://doi.org/10.1162/neco.1996.8.7.1341*
[74] B. P. Woolf. *Building intelligent interactive tutors: Student-centered strategies for revolutionizing e-learning*. Morgan Kaufmann, 2010.