UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Comparison between machine learning and human learning from examples generated with machine teaching

DEGREE FINAL WORK

Degree in Computer Engineering

Author: Gonzalo Jaimovitch López

Tutors: Cèsar Ferri Ramírez, José Hernández-Orallo

Year 2019-2020

Dedication

Lo primero de todo, muchas gracias a mi familia y amigos por su apoyo durante el recorrido académico y el tiempo dedicado a escribir este trabajo.

Mamá, Papá y Ani, gracias por creer en mí. No habría llegado hasta aquí si no fuera por vuestra paciencia y dedicación. Este logro es tanto mío como vuestro. Enhorabuena

Cristina, gracias por tus correcciones y recomendaciones, pero sobre todo, gracias por tu amistad.

Cèsar y José, gracias por las oportunidades que me habéis brindado y el tiempo que me habéis dedicado. Me siento muy afortunado.

A los miembros del grupo DMIP, gracias por acogerme con cariño y respeto. He aprendido mucho con vosotros. En especial, gracias a Lidia y David por aclarar mis dudas con tanta amabilidad.

María, gracias por regalarme tantos momentos felices.

Valéry, gracias por salvarme la vida en los tiempos difíciles.

Tomi, gracias por inspirarme a querer llegar más lejos.

Alberto, gracias por enseñarme el lado bueno de las cosas.

Finalmente, gracias a los 30 participantes anónimos que dedicaron parte de su tiempo para colaborar en este proyecto.

Acknowledgements

The author was partially funded by the DMIP group of the Valencian Research Institute for Artificial Intelligence (VRAIN)

Resum

En contrast al terme àmpliament conegut com aprenentatge automàtic (*Machine Learning*), branca d'èxit en el camp de la Intel·ligència Artificial (IA), sorgeix el concepte d'ensenyament automàtic (*Machine Teaching*). Un dels seus objectius principals és l'optimització de l'aprenentatge mitjançant l'elecció d'exemples etiquetats que constituiran el conjunt d'entrenament per als models d'aprenentatge. Recentment, s'han començat a desenvolupar aproximacions enfocades a la utilització d'aquestes tècniques d'obtenció d'exemples centrades en l'aprenentatge d'humans. Aquesta aplicació està fortament relacionada amb la branca de IA explicable, més concretament amb les explicacions basades en exemples, la finalitat dels quals és transmetre als humans allò que una màquina ha aprés. En aquest treball es proposa realitzar una comparació del procés d'aprenentatge des d'exemples generats per *Machine Teaching* entre diferents sistemes d'aprenentatge com un sistema de programació funcional inductiva (MagicHaskeller), una xarxa neuronal profunda basada en el model *Transformer* (GPT-2) i humans. Per a concloure, l'efectivitat de les explicacions basades en exemples amb aquesta configuració és analitzada. Els resultats obtinguts assenyalen la necessitat de proporcionar informació addicional junt als conjunts òptims d'exemples, extrets mitjançant la configuració de *Machine Teaching* aplicada en aquest treball.

Paraules clau: *Machine teaching*, Intel·ligència Artificial explicable , Programació inductiva, Explicacions basades en exemples, Models *Transformer*

Resumen

En contraste al término ampliamente conocido como aprendizaje automático (*Machine Learning*), rama de éxito en el campo de la Inteligencia Artificial (IA), surge el concepto de enseñanza automática (*Machine Teaching*). Uno de sus objetivos principales es la optimización del aprendizaje mediante la elección de ejemplos etiquetados que constituirán el conjunto de entrenamiento para los modelos de aprendizaje. Recientemente, se han empezado a desarrollar aproximaciones enfocadas a la utilización de estas técnicas de obtención de ejemplos centradas en el aprendizaje de humanos. Esta aplicación está fuertemente relacionada con el concepto de IA explicable, más concretamente con las explicaciones basadas en ejemplos, cuya finalidad es transmitir a los humanos aquello que una máquina ha aprendido. En este trabajo se propone realizar una comparación del proceso de aprendizaje desde ejemplos generados por *Machine Teaching* entre diferentes sistemas de aprendizaje como un sistema de programación funcional inductiva (MagicHaskeller), una red neuronal profunda basada en el modelo *Transformer* (GPT-2) y humanos. Para concluir, la efectividad de las explicaciones basadas en ejemplos con esta configuración es analizada. Los resultados obtenidos señalan la necesidad de proporcionar información adicional junto a los conjuntos óptimos de ejemplos, extraídos mediante la configuración de *Machine Teaching* aplicada en este trabajo.

Palabras clave: *Machine teaching*, Inteligencia Artificial explicable, Programación inductiva, Explicaciones basadas en ejemplos, Modelos *Transformer*

Abstract

In contrast to the term widely known as Machine Learning, a successful branch of Artificial Intelligence (AI), the concept of Machine Teaching arises. One of its main objectives is the optimization of learning through the choice of labeled examples that will constitute the training set for the learning models. Recently, some approaches have focused on the use of these techniques to obtain examples for human learning. This application is strongly related to the field of explainable AI, more specifically to exemplar-based explanations, whose purpose is to convey to humans what a machine has learned. In this paper we propose to make a comparison of the learning process from examples generated by Machine Teaching among different learning systems like an inductive functional programming system (MagicHaskeller), a transformer-based deep neural network (GPT-2) and humans. To conclude, the effectiveness of the exemplar-based explanations using this setting is discussed. The obtained results highlight the necessity of providing additional information alongside the optimal example sets, extracted using the machine teaching setting applied in this work.

Key words: Machine teaching, Explainable AI, Inductive programming, Exemplar-based explanations, Transformer Models

Contents

Appendices

List of Figures

List of Tables

List of algorithms

CHAPTER 1
Introduction

As stated in [12], "intelligence is the ability of a decision-making entity to achieve success in a variety of goals when faced with a range of environments". According to this definition, intelligence is not anymore restricted to biological entities, even if artificial *general* intelligence is way far to achieve.

In the last years, AI (Artificial Intelligence) systems, particularly the field of machine learning, has gained increasing popularity due to its outstanding performance in tasks such as image classification or speech recognition. The ability of machines to learn from vast amounts of information has allowed them to outperform humans in scenarios where they are clearly overwhelmed [22]. For instance, in [5], an experiment showcased how deep neural networks reached higher classification accuracies than human experts when classifying microscopy images.

Trust in the decisions made by machine learning systems is necessary as they become more and more present in human lives. This trust might be unreachable when the models cannot be explained to humans, especially in stages like healthcare or self-driving. This problem is addressed in the emerging field known as explainable AI [33].

One of the approaches of explainable AI consists of giving examples to humans so that they can build their own explanations. The challenge with this setting relies on the appropriate selection of the examples to be provided so that explaining happens effectively and, ideally, efficiently. In this context, recent research has linked another emerging AI field as a possible solution to the example selection: machine teaching. With this setting the tables have turned, since machines become the teachers and humans the learners.

1.1 Motivation

1.1.1. Personal Motivation

This thesis was offered by Cèsar Ferri and José Hernández-Orallo, its supervisors, after a long search for a project where I could get introduced to AI.

During the bachelor's degree, I have been able to experience the wide variety of possibilities this professional profile can offer. The exciting world of information systems and technologies does not make the decision easy, considering the decision of choosing a theme where I should spend my time for months.

I'm not going to lie. I always felt AI was some kind of super natural magic that provided machines with free will and consciousness. After attending some degree courses

and other related activities, my perception changed completely: statistics, mathematics, optimization, computation, etc.

The magic had gone, but the interest did not disappear with it. I was told medicine could change forever, with the knowledge of the very best experts scaled up so a greater amount of people can have access to accurate diagnostics. Self-driving vehicles being part of our future and reducing drastically the number of accidents. Personalized education that will democratize the access to quality education and will help more and more people achieve their goals. And these examples are just part of what AI research and applications can offer to humanity.

I don't know where I will be in the future, but in the end I found out that the time I dedicated writing this thesis increased my interest in the field, making a career in AI one great option to dedicate some of my precious life time to.

1.1.2. Professional Motivation

As AI systems become more and more present in the human lives, trustworthy decisions are required in multiple critical areas or services. For instance, a person who is about to take a long ride in a self-driving car and doesn't know why the decisions are taken by the AI system will probably not want to risk her life considering the multiple misfortunes that can occur on the road. Consequently, a doctor might not risk to accept the cancer-free diagnostic provided by an AI system without understanding why the system returned that prediction [33].

In a recent future, not only will the industry have to care about building excelling AI systems but also about finding the way of successfully conveying their behavior to the stakeholders. Therefore, there is an increasing need for acquiring the capability of effectively explaining the actions or decisions of these systems to humans. Alternatives to forcing AI systems to be based on explainable models while sacrificing their performance lead to other choices like exemplar-based explanations [11], introduced in this work. This is one of the main reasons why there exists an interest in evaluating exemplar-based explanation quality when explaining models or concepts to humans.

On the other hand, machine teaching arises as a starting point from where to build efficient exemplar-based explanations. It might also enhance the machine learning process interaction between humans and machines, possibly improving the way machines learn, guided by human domain experts rather than by data only [31]. With all this, one of the main drivers of this thesis is procuring further insights about the effectiveness of the explanations when using machine teaching techniques to obtain the exemplar-based explanations for both human and machine learning.

1.2 Objectives

The main objective of this work is to compare the learning capacity of different systems when examples are generated with machine teaching.

For this purpose, the following specific objectives are addressed:

- Design an experiment, its setting and the procedure to evaluate the performance of the human and machine learners.
- Select machine learning techniques that are compatible with the learning task so a comparison with humans can be undertaken.

- Generate example sets, with a machine teaching setting, and additional examples for the experiment explanations.
- Implement the experiment tests for the different teaching scenarios using appropriate tools.
- Analyse the obtained results after evaluating the different learners using the implemented tests.

1.3 Methodology

In order to fulfil the above-mentioned objectives, the followed methodology is laid out in the following 5 blocks: (1) theoretical framework review, (2) experiment design, (3) teaching scenario implementation, (4) experiment deployment and execution and (5) result analysis and discussion.

Figure 1.1: Blocks in the methodology followed for this work

In (1), a review of the machine teaching theory, main aspects of machine learning and human learning, and the areas of explainable AI and inductive programming will be performed. Following this, in (2), a machine teaching setting will be studied and set as the main approach for this work. In addition, the experiment will be designed, choosing the lesson domain and deciding about the different learning phases or number of examples to provide in each phase, among others. Further on in (3), the different learners will be selected, the study of the picked machine learning systems will be undertaken and the learner-specific tests to evaluate the learning progress will be implemented. Afterwards, test deployment and result gathering will be performed in (4). Finally, the results of the experiments will be disclosed in (5), analysing the effectiveness of the teaching/explaining process with this setting and comparing the results between humans and machines.

1.4 Structure

The rest of this document is structured as follows: Chapter 2 starts with an introduction to machine learning (Section 2.1) and a general characterisation of human learning and its influential factors (Section 2.2). Then, a wide review of the machine teaching definition, specifics, approaches and evolution is presented (Section 2.3). To conclude the chapter, an overview of explainable AI (Section 2.4) and inductive programming (Section 2.5) as related areas to this work is undertaken. In Chapter 3, the universal language known as P3 (whose concepts will be taught in the experiments) is introduced (Section 3.1), followed by the description of the machine teaching setting applied to obtain the examples for explaining the sampled concepts (Section 3.2). Then, the experiment characterisation (Section 3.3) is developed (deciding how to sample concepts, the number of examples to provide, how to provide them to the learners, etc.), alongside the teaching protocol to adopt in the experiments conducted. Finally, the different teaching scenarios included in this work are outlined, depicting the machine learning systems employed and explaining the tools used to implement and deploy the evaluation tests for the experiments (Section

3.4). Afterwards, in Chapter 4, the accuracies obtained by each of the learners considered for the experiments are presented, along with the learning curves which will help to draw a comparison among the different learners, perform an analysis and extract the conclusions, formulated in Chapter 5. Finally, the reader can find some annexed materials at the end of the document, in appendices A, B, C and D.

CHAPTER 2
Theoretical Framework

In this chapter, an overview of machine learning and human learning is presented. Following this, an introduction to the field of machine teaching will be carried out, along with some of the different approaches that have been proposed in the recent years. Finally, the fields of explainable AI and inductive programming will be presented, due to their connections and interest for this work.

2.1 Machine Learning

As the main goal of this work is to compare the performance of humans and machines in a machine teaching scenario, it seems relevant to establish the foundation from which this comparison will be carried out.

Machine learning is a subfield of computer science and AI that studies algorithms and techniques aiming to solve complex problems, with the use of data, which are hard to program using conventional programming methods.

In other words, machine learning algorithms "make decisions without being specifically programmed to make those decisions" [13]. The term was introduced by Arthur Samuel in 1952, and was followed by several breakthrough achievements like the postulation of the Perceptron by Frank Rosenblatt, which led to high-end-application neural networks known today [37].

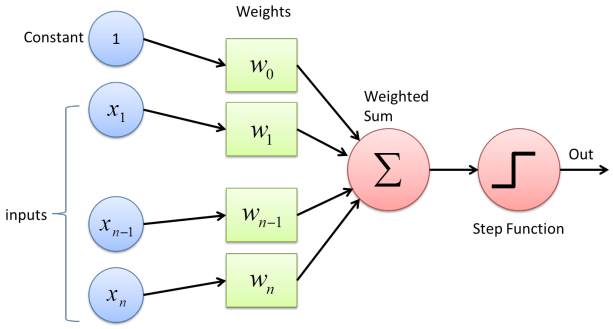

Figure 2.1: Perceptron algorithm (Source: [36])

Mostly, these algorithms are built on statistical models based on datasets related to the problem domain. Through the machine learning literature, different types of learning have been addressed, being the following a primary classification:

- **Supervised learning**: the machine is given a dataset comprised of labeled examples, which means that the right answers of the provided instances for the prob-

lem to solve are available. The answers or labels can be classes in a classification problem, or units of a specific attribute in a regression problem. For instance, determining incoming email as spam or not spam is a classification problem. The resulting algorithm, trained on different emails labeled as "spam" or "not spam" from a dataset, is capable of classifying new emails. Conversely, an example of a regression problem can be the prediction of house prices. The resulting algorithm, trained on house attributes from a dataset (like the number of rooms or the m^2 of the yard) and given their prices, is capable of predicting the price of new houses.

- **Unsupervised learning**: the machine is given a dataset comprised of unlabeled examples. In these cases, the right answers for a specific problem are not provided. One of its well-known forms is clustering (see Figure 2.2). For instance, with a dataset comprised of client information, a clustering algorithm is capable of discovering those groups that share similar features.

- **Semi-supervised learning**: the machine is given a dataset comprised of labeled and unlabeled examples. Usually, in this setting the number of unlabeled examples is higher than the number of labeled ones. The unlabeled examples help by providing more information about the data distribution [6].

- **Reinforcement learning**: the machine perceives data from an environment, and can execute actions considering its actual state and the different possible states derived from its actions, while optimizing a specific goal choosing the most rewarding action [37, 6, 29]. "The longer a machine is allowed to observe and learn, the better it is able to learn the longer-term impact of its decisions" [37]. One remarkable example is the case of AlphaGo, a reinforcement learning algorithm from Google's DeepMind subsidiary which beat the world's best player in the complex Chinese game known as "Go" [2].

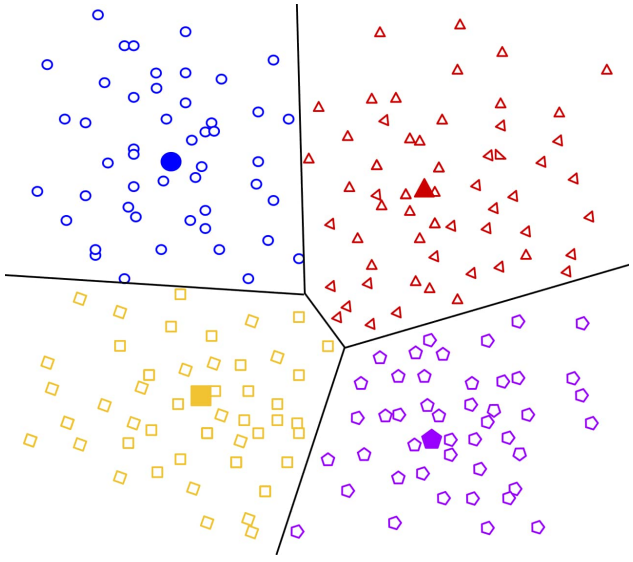

Figure 2.2: Centroid-based clustering (Source: [7])

The process of building a model is called training, and the data used for the training process is known as the training set [6]. In supervised learning, even if these algorithms could be trained using all the information in the datasets, one common way to proceed is to separate it into a training set, a validation set, and a test set. With the dataset partition, a proper evaluation of model performance can be undertaken before the deployment of a model. Other techniques such as the cross-validation might be applied [6]. These settings

aim to evaluate the power of generalization[1] obtained with the given algorithm, trying to avoid the problems of overfitting[2] and underfitting[3].

Some prominent achievements in machine learning are:

- Speech recognition as the ability of a machine to recognize speech and convert it to text, improving significantly the human-computer interaction [37].
- The introduction of the Transformer model [43] and the largest OpenAI's Transformer-based language model GPT-3, with 175 billion parameters [4].
- Computer vision as the ability of a machine to perceive and process real-time images. Driverless driving is one of its possible applications [37].

Since the 2000s, machine learning experimented an accelerated progress, due to the availability of large amounts of data on the Internet, the increased computational and storage power, or the improvements in big data algorithm optimization [37], among other things. Very recently, a focus has been set on the development of application-specific circuits so machine learning applications (specifically neural networks) are implemented as a hardware-based solution in a brain-inspired fashion, rather than the traditional software-based solution [44]. This field is known as neuromorphic computing.

In contrast to the era of the 1990s, when AI was almost restricted to the brightest engineers and scientists, fields like machine learning become now more accessible to the general population, as the interest has increased tremendously in the recent years [37]. This enhanced accessibility has been significantly affected by the release of open source libraries for widespread programming languages (like Python), highlighting Keras, TensorFlow or PyTorch. Other recent approaches aim to facilitate the access to machine learning application development like H2O AutoML or the Machine Teaching approach by Microsoft.

While machine learning algorithms get satisfactory results, in so many cases it is not very clear how a problem is being solved. This is especially notorious for machine learning algorithms based on neural networks [37]. Some applications in finance (e.g. loan applications) or healthcare (e.g. predicting a disease) need the models and predictions to be explainable [27]. This interpretability problem is addressed in the area of explainable AI, introduced in Section 2.4.

2.2 Human Learning

"Human learning is mostly a directed process, guided by other people: parents, teachers and society in general" [22]. As stated in [32], human learning is social. However, even if human learning can be supervised, it can also be a spontaneous and self-motivated process [16]. One of the distinguishing characteristics of human learning, affected by the social interactions, is the ability of imitation [32, 16]. It has been observed that children are more predisposed to reenact actions when those are produced by a person rather than by inanimate agents [32].

When learning happens by observing and imitating other human beings, the learning process is accelerated (being faster than individual discovery), and the learning opportunities are multiplied [32]. Humans are born immature and, since then, the neural archi-

[1] "How well a model trained on the training set predicts the right output for new instances" [1]

[2] When the model fits really good the training data, but fails to generalize with new instances of the distribution

[3] When the obtained hypothesis poorly fits the training data

tecture starts developing from the initial learning setting, allowing more-complex future learning [32].

Another critical driver of human learning potential is the language acquisition [32] as a powerful tool for communication, in such a way that a message sender finds the right signs or words so a receiver understands the meaning of a message [22]. "The observation that humans are able to cover a wide range of concepts and can learn from very few examples suggests that humans share a prior and may communicate, and teach, accordingly" [23].

Humans in general are good at learning or generalizing from a few examples [22, 30]. In addition, humans are good at understanding problem-solving independently of the domain because of their ability of using background knowledge according to the context [34], and are able to use their knowledge in richer ways for action, imagination or explanation than machines do [30]. One example can be found in [24]: "(...) even very complex Turing-complete (universal) concept classes in natural language can be transmitted using just a few examples. For instance, when humans are said that 'dollars', 'euros' and 'yens' are positive examples but 'deutschemarks' are not, most understand that the concept is about currencies that are legal tender today".

In contrast, most machine learning state-of-the-art models (e.g. deep neural networks) are data-hungry, i.e., good at learning from vast amounts of data without the need of background knowledge [22]. For instance, progress in areas like NLP (Natural Language Processing) has led to the development of outstanding deep neural networks such as GPT-3, a task-agnostic model trained using huge data repositories like Common Crawl or Wikipedia [4]. This does not imply every machine learning paradigm is strictly data-hungry; an example is Inductive Programming (explained in Section 2.5).

The machine's ability of learning from large datasets is closely related to its unlimited (not infinite) memory retrieval capacity, which is limited in the case of humans [35]. Moreover, human learning performance might be affected by their emotions, which result critical for the understanding of human intelligence [32].

Other considerations have stated that human learning performance can be better if they are aware that a teacher assistance might be helpful rather than if they do not know about the presence of a teacher [41], or the fact that human learners are sensitive to the sequential order of teaching items that are provided in a teaching scenario [46].

In [22], the authors inform about their expectation of a change in machine learning such that the renovated efforts to make it more "human-like crystallise". The work in [30] suggests that "the principles of compositionality, causality and learning to learn..." (characteristics of human learning) "...will be critical in building machines that narrow this gap" (where humans outperform machines in learning from few examples and using the gained knowledge in richer ways). Other works suggest the building of robots that can learn like human infants through imitation and observation [32, 16].

2.3 Machine Teaching

Machine teaching is an emerging field in AI, which has its origins in the 1990s [45]. However, its popularity has considerably increased in the last years [42].

As stated in [45], machine teaching can be understood as an inverse problem to machine learning, where the main goal is to find the optimal training set that produces a target model considering a learning algorithm [11], so learning happens efficiently [42]. However, not every machine teaching scenario implies teaching a machine learning

model, as the learning of cognitive learning models, concept languages and even richer languages are part of the machine teaching interest [11].

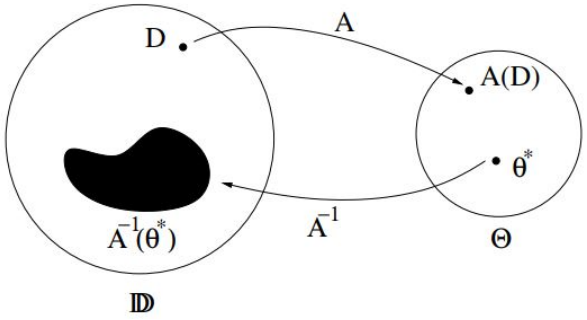

Figure 2.3: Machine teaching as an inverse problem to machine learning (Source: [45])

In Figure 2.3, the inverse relation between machine learning and machine teaching can be observed. Let $D \in \mathbb{D}$ be a training set as part of the space of training sets, and $A(D)$ the model returned, using a machine learning algorithm, as part of the model space Θ. In the machine learning case, the aim is to obtain the model $A(D)$. On the other hand, in a machine teaching setting, given a target model $\theta^* \in \Theta$, the goal is to obtain the optimal training set (from now on, the optimal training set will be named "witness set") provided by $A^{-1}(\theta^*)$. It is important to consider that the mapping between $\mathbb{D}$ and Θ is not bijective, so different sets from $\mathbb{D}$ can be consistent with θ^*. Therefore, a specific criterion must be set in order to compute the A^{-1}. The optimal approach will be discussed later. Furthermore, the inverse problem is given as a comprehension example, not implying this is the way to proceed in order to obtain the optimal training set or witness set, as it will be seen in this chapter.

One of the main misunderstandings about machine teaching can be represented by the following question: "which is the interest in machine teaching if we already have the target model?" [45]. The applications of machine teaching are closely related to the role adopted by humans and machines in the teaching scenario (see Figure 2.4) comprising the teacher and the learner (or learners), as explained in [46]:

- Machines as the teacher and the learner: one example can be a data poisoning attack, where the teacher as a malicious agent wants the learner to learn an undesirable model by sending poisoned data. These attacks are widely known as adversarial attacks in the AI literature.
- Machine as the teacher and human as the learner: one example can be a computer tutoring system using a cognitive model of the human learner to obtain the optimal lesson. In addition, recent work has focused on using this machine teaching setting to explain AI models, establishing a close relationship with the field of explainable AI [11].
- Human as the teacher and machine as the learner: one example can be the use of domain experts to build machine learning models faster and better by providing the considered training set. This setting has gained increasing media attention thanks to Microsoft's interest in the field, advocating the direct interaction of human knowledge in the machine learning process rather than learning from data alone [31].
- Humans as the teacher and the learner: even if this stage is not a general focus in machine teaching, the insights of the field might help to enhance pedagogy. For in-

stance, discovering an effective training set for teaching a specific model or concept to humans can be introduced in future lessons. Recent work like [40] could fit this application. However, the individual differences happening in humans are pointed as a limitation for a one-fit-for-all solution.

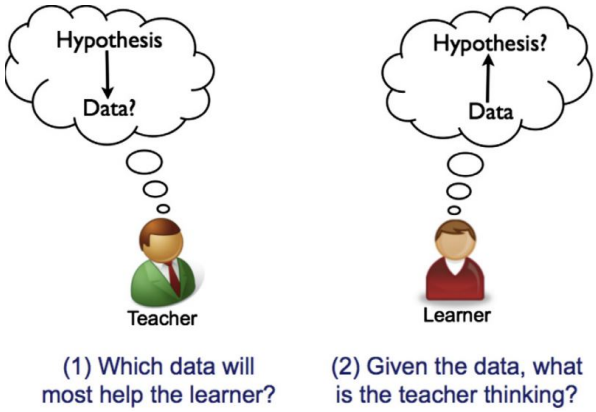

Figure 2.4: Teaching scenario: the teacher and the learner (Source: [41])

From now on, the focus will be set on the scenario where machines are the teachers, and humans or machines are the learners. The rest of the section will be organized as follows: in Section 2.3.1 the teaching dimension approach and a related theoretical model will be explained; in Section 2.3.2 a discussion about the teaching dimension limitations for certain problems will take place, along with the introduction of a different approach, the teaching size; to conclude, in Section 2.3.3, the relevance of priors in the teaching scenario will be explained.

2.3.1. The Teaching Dimension

Most of the research in machine teaching is based on the idea of the teaching dimension, which can be generalized as the minimum number of examples so that the learner can identify a concept [42]. There are some variants of the teaching dimension, such as the Recursive Teaching Dimension or the Preference-Based Teaching Dimension [46, 14, 42].

The classical teaching dimension can be formalized as follows [24]: having a possible infinite instance space X with instances $x_i \in X$ that can be positive examples (represented by a pair $\langle x_i, 1\rangle$) or negative examples (represented by a pair $\langle x_i, 0\rangle$), a concept is a binary function over X, with the possible outputs {0,1}. A concept language or class C is composed of a possibly infinite number of concepts. An example set S is a possibly empty set of examples. It is said that a concept c satisfies S, denoted by $c \vDash S$, if $c(x_i) = 1$ for the positive examples of S and $c(x_i) = 0$ for the negative ones. All concepts satisfy the empty set. After this, the teaching dimension of a concept c can be defined as follows:

$$TD(c) \triangleq \min_{S} \{|S| : \{c\} = \{c' \in C : c' \vDash S\}\} \tag{2.1}$$

One of the teacher algorithms in machine teaching, as explained in [46], will be used to reinforce the teaching dimension notion. Specifically, the teacher algorithm can be written as a two-level optimization problem considering a machine learner model, which is presented below:

$$\min_{D,\hat{\theta}} \quad ||\hat{\theta} - \theta^*||^2 + \eta||D||_0 \tag{2.2}$$

$$s.t. \quad \hat{\theta} = arg\min_{\theta \in \Theta} \sum_{(x,y) \in D} l(x, y, \theta) + \lambda ||\theta||^2 \tag{2.3}$$

In these equations, θ^* represents the target model, D the training set, Θ the hypothesis space, l the loss function, λ the regularization term, η the teaching dimension trade-off term and $\hat{\theta}$ the final model that will be learned using D (it does not necessarily have to be equal to θ^*). The Equation 2.3 is an example of the regularized empirical risk minimization framework, whose goal is to obtain the weights of a model that minimizes the training error while avoiding overfitting or underfitting (which are regulated with λ). Conversely, the Equation 2.2 aims to minimize the difference between the learned model (using D as the training set) and the target model to teach, and the number of examples to provide so the learner can identify $\hat{\theta}$. Note that the learning process acts as a subroutine in order to get the witness set.

Once the aforementioned teacher algorithm has been presented, a more general formal definition might be introduced:

$$\min_{D, \hat{\theta}} \quad TeachingRisk(\hat{\theta}) + \eta TeachingCost(D) \tag{2.4}$$

$$s.t. \quad \hat{\theta} = MachineLearning(D) \tag{2.5}$$

The $TeachingRisk(\hat{\theta})$ represents the satisfaction of the teacher. If the final model to be learned by the learners differs significantly from the target model that the teacher wants the learner to learn, it can be said that the teaching goal has not been accomplished. On the other hand, the $TeachingCost(D)$ represents the number of examples to give so that the learner can learn the target model, i.e., the teaching dimension. In this setting, it is assumed that the greater the number of examples required, the greater the cost.

Different variations of the above-explained schema can be found, with a trade-off between the $TeachingRisk(\hat{\theta})$ and the $TeachingCost(D)$ as the minimization goal depending on the requirements of the problem to solve. Moreover, this schema assumes that the teacher has full knowledge of the learner learning algorithm, which could not be the case. Then, learners become a black box[4] or grey box[5] to the teacher, and other techniques where the teacher tests the learner during the teaching process can be applied [11, 8].

Even if the teaching dimension has been one of the main approaches in machine teaching, recent work has identified limitations which required a new formulation for the machine teaching framework, as explained in the next section.

2.3.2. The Teaching Size

In [24], the authors affirm that "learning from examples when the concept class is rich and infinite is usually considered a very hard computational problem". Recent research has placed the focus on the teaching of universal languages which are structurally rich using small witness sets [42].

As stated in [11], the teaching dimension is appropriate in those situations where all the examples in the domain D have the same size (e.g. unstructured data representations), i.e., all the examples are equally complex from a coding perspective. Nevertheless,

[4] As opposed to a white box model, it can be said that a black box model is one that cannot be internally observed, so the only possible way to understand its behaviour is by contrasting the inputs with the outputs. Other notion of a black box model refers to those models that cannot be interpreted (e.g. deep neural networks) even if the internal parameters or hyperparameters can be observed

[5] Models whose parameters or hyperparameters can be partially observed

in the case of teaching concepts from richer languages, it has been demonstrated that the teaching dimension approach can produce small witness sets for explaining concepts (in terms of the cardinality), but the examples might be enormously large [42]. This issue breaks with the theoretical expected feasibility of teaching concepts [11], and does not properly address the complexity of learning a concept [42].

In addition, when examples have structure, the notion of approximation (which can be observed in the two-level optimization problem in Section 2.3.1) does not make sense [42]. Following the example given in [42], if a hypothesis outputs '11110000' instead of '11110010', it might seem close but the concept can still be afar. Accordingly, the formulation of a new approach to address the above-mentioned problems has been conducted in recent work [42].

The notation adapted in [22] will be used to define the teaching size. "We have a possibly infinite example (or instance) space X and a possibly infinite concepts class C consisting of concepts over X. Given a concept $c \in C$ and an example set X, we say that c satisfies X, denoted by $c \vDash X$, if c is consistent with all the examples in X. From now on, we will consider examples as pairs $\langle i, o \rangle$ and concepts as functions mapping inputs with outputs. An example set $S = \{\langle i_1, o_1 \rangle, \ldots, \langle i_k, o_k \rangle\}$ is just a finite set of i/o pairs, used as witness for the teaching process. In this functional presentation of examples, we can also express that c satisfies S if $c(i) = o$ for all the pairs $\langle i, o \rangle$ in S. All concepts satisfy the empty set". By choosing an encoding function δ as the number of bits needed to encode S, $\delta(S)$ represents the size of S. With all this, the teaching size can be defined as follows:

$$TS(c) = \min_S \{\delta(S) : \{c\} = \{c' \in C : c' \vDash S\}\} \tag{2.6}$$

As it can be observed, the teaching size differs from the teaching dimension since it allows a greater number of examples in the witness sets (if needed), while these examples might be smaller in terms of the encoding function δ.

To complete the notion of the teaching size (in the same way the teaching dimension notion was explained in Section 2.3.1), a learner model is included. Having a program p in a given language L satisfying the example set S, denoted by $p \vDash S$, the equivalence size of programs that are compatible with c is $Class_L(c) = \{p : \forall S, p \vDash S \iff c \vDash S\}$. The learner Φ can be understood as a function mapping sets to concepts ($\Phi(S) = c$) [22].

With the definition of the learner, the teaching size can be reformulated as follows:

$$TS(c) = \min_S \{\delta(S) : \Phi(S) \in Class_L(c)\} \tag{2.7}$$

In this setting, more than one concept might be consistent with the given set S. For instance, the concepts "identity" and "print the first character of a string" will be consistent if the witness set is comprised of a one-character input example with the equal one-character output. In such a case, the learner will not be able to properly identify the concept, so the teacher will fail in teaching the concept to the learner [22]. This problem is addressed in [42] with the introduction of priors to the teaching scenario, as explained in the next section.

2.3.3. Alignment of Priors

In the previous sections, the limitations of the teaching dimension in some scenarios was pointed out. Moreover, it has been explained that the introduction of the teaching size is not sufficient to ensure the proper identification of concepts by the learner. This happens

when the concept prior is uniform and the learner is not able to choose between two equally-consistent concepts [22].

The uniform distribution, explicitly used or implicitly assumed for finite classes, cannot be applied to infinite concept classes [24]. Recent work has introduced two priors in the machine teaching setting [22, 24, 42, 23]: the learning prior and the sampling prior. The learning prior can be defined as the expectation of the learner about the concepts, and is used to guide the learner on how to search for concepts starting from the given witness set [42]. On the other hand, the sampling prior (used by the teacher) determines what are the concepts that will be taught [22].

As stated in [22], ideally, the learning prior and the sampling prior should be aligned. One natural choice for these priors is simplicity, which can be formally defined as the size of the concepts and examples [22], highly related to other approaches such as the use of the Occam's razor[6] or MML (Minimum Message Length) / MDL (Minimum Description Length) principles [24].

Considering these two priors and the simplicity approach, the learner and the teaching size definitions explained in Section 2.3.2 can be updated. Let $l(p)$ be the length in bits of a program p in L using an appropriate encoding. Let $\prec$ be the total order of programs ordered by l, where shorter programs precede longer ones. In case of equal l, ties are broken lexicographically [22]:

$$TS_l(c) = \min_S \{\delta(S) : \Phi_l(S) \in Class_L(c)\} \tag{2.8}$$

$$\Phi_l(S) = arg \min_p \prec \{l(p) : p \vDash S\} \tag{2.9}$$

With this setting, the authors in [42] have proved theoretically and empirically that it is possible to find witness sets for explaining concepts whose size is smaller than the programs they identify, "which is an illuminating justification of why machine teaching from examples makes sense at all" [42].

Once the Equations 2.8 and 2.9 are presented, the concept of the Teaching Book introduced in [42] is described. The Teaching Book is a list consisting of entries in the form $\langle w, p \rangle$, being w the optimal witness set and p the smallest program compatible with w [22]. However, the teacher might just want to find the witness set for a specific concept. Then, the (l, δ)-optimal teacher algorithm from [22] (see Algorithm 2.1) can be used.

Algorithm 2.1 (l, δ)-Optimal Teacher Algorithm

```
for all witness sets w in increasing δ(w) and ordered by ◁ for equal size do
  if Φ_l(w) is equivalent to c then
    return w
  end if
end for
```

[6] "Occam's razor, also spelled Ockham's razor, also called law of economy or law of parsimony, principle stated by the Scholastic philosopher William of Ockham (1285–1347/49) that pluralitas non est ponenda sine necessitate, "plurality should not be posited without necessity". The principle gives precedence to simplicity: of two competing theories, the simpler explanation of an entity is to be preferred. The principle is also expressed as "Entities are not to be multiplied beyond necessity." [10]

2.4 Explainable AI

Some machine learning models do not require explanations because of its low-risk environment nature such as a movie recommender system, or methods which have already been profoundly studied and evaluated such as some in optical character recognition [33].

Nevertheless, in other environments where the decisions or predictions made by AI systems can be considered critical, the lack of an explanation of the system's decisions can reduce drastically the trust humans have towards these systems. This becomes fundamental so that humans take actions based on the given outputs or decide whether or not to deploy a model [38]. For instance, when using machine learning systems for medical diagnosis or terrorism detection, users cannot have blind faith upon the system's predictions, as the consequences might be disastrous [38]. Furthermore, a proper understanding of the model's behaviour might be used by developers to debug AI systems or to improve their performance [33, 39].

Explainable AI is the subfield of AI that aims to explain the actions and decisions of AI systems [22]. One of the key factors in explanaible AI is the notion of interpretability, which can be defined as the degree to which humans can understand why a system made a decision or the degree to which humans can predict the model's outputs [33]. In [9], the authors identify three different types of AI systems depending on the different notions of explainability (see Figure 2.5):

- **Opaque systems**: closed-source licensed systems hidden by its licensor and those systems relying on black box models.
- **Interpetable systems**: systems where the user can see and understand the mathematical mapping of inputs to outputs, so a level of understanding of the technical aspects is required. For instance, in a regression model the user can analyse the weights of the model and determine the importance given to each feature. Deep neural networks, as black box models, are unlikely to be interpretable.
- **Comprehensible systems**: systems capable of explaining the decision chain to humans from the inputs to the outputs provided. The user is responsible for identifying the key relations among the input, the output and other possible symbols produced by the system. Comprehensible models can be interpretable and vice versa.

One of the easiest ways to achieve interpretability is to use algorithms that create interpretable models [33]. Some of these interpretable models can be decision rules or linear regression models [33]. There is a trade-off between performance and explainability, as the most performing models are the most opaque [25]. Other trade-offs have been identified, between fidelity and comprehensibility or between the previously mentioned and actionability [22].

"Interpretive and comprehensible models thus enable explanations of decisions, but do not yield explanations themselves" [9]. This implies that interpretability or comprehensibility is not enough to ascertain that the explanation has been properly understood, since depending on the background knowledge of a subject and other inter-human differences, alternative interpretations can be deduced [9].

In [22], two key factors in human interpretability are explained: the simplicity bias and the confirmation bias. Simplicity, identified as a fundamental cognitive principle, must be understood in terms of the representational language in use and the background

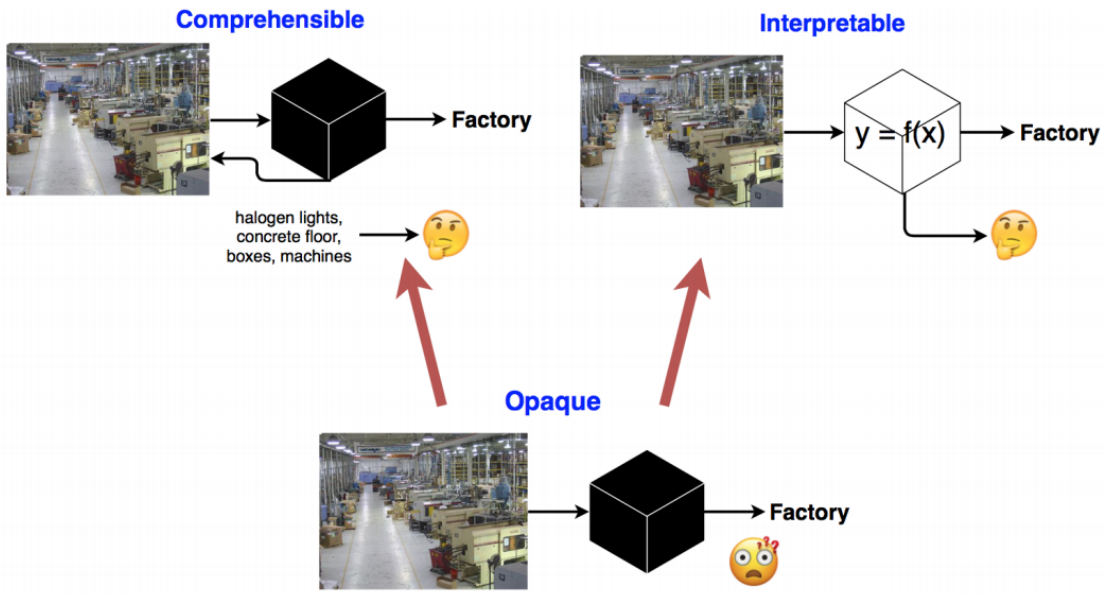

Figure 2.5: Classification of AI systems in the context of explainable AI (Source: [9])

knowledge, considering that what is simple for one person might be complicated for another one [22]. Conversely, confirmation bias determines how explanations are assimilated, as humans tend to ignore information inconsistent with their prior beliefs [33].

Since the use of interpretable models can oversimplify the problem at hand [11], other methods have been proposed in the explainable AI field, divided into model-specific and model-agnostic methods. With model-specific methods, there is still a commitment to one model over the possible alternatives [33]. Hence, by using model-agnostic methods, developers have the flexibility to choose the preferred model instead of being forced to use an interpretable one or a model-specific method one to build target solutions [39].

One of the model-agnostic approaches known as exemplar-based explanations, argued in [33], aims to explain models by providing examples to humans so that they can build their own understanding of a target model. As stated in [33], the exemplar-based explanations only make sense if they can be represented in a humanly understandable way. Recently, it has been connected with the field of machine teaching [11, 22], aiming to solve the problem of choosing the appropriate examples for the explanations.

2.5 Inductive Programming

Inductive programming is a subfield of AI and automatic programming which incorporates the approaches concerned with the learning of programs or algorithms using incomplete specifications [26]. Inductive programming "emerged as a general term to refer to inductive inference with the use of programming languages" [20].

The aim is learning declarative (functional or logic) programs from a small number of (labeled) examples [34]. In contrast to machine learning, learning from few examples is possible because the systems are provided with background knowledge [17], usually using a hypothesis-driven approach instead of a data-driven approach [34]. This is closely related to the way humans solve problems with just a few examples, by choosing the appropriate background knowledge according to the context [34]. An overview of the differences between inductive programming and machine learning can be seen in Table 2.1.

	Inductive Programming	**Other Machine Learning Paradigms**
Number of examples	Small	Large, for example, big data
Kind of data	Relational, constructor-based datatypes	Flat tables, sequential data
Data Source	Human experts, software applications, Human-Computer interaction, and others	Transactional databases, Internet, sensors (IoT), and others
Hypothesis Language	Declarative: general programming languages or domain-specific languages	Linear, non-linear, distance-based, kernel-based, rule-based, probabilistic, etc
Search strategy	Refinement, abstraction operators, brute-force	Gradient-descent, data partition, covering, instance-based, etc.
Representation learning	Higher-order and predicate/ function invention	Deep learning and feature learning
Pattern comprehensibility	Common	Uncommon
Pattern expressiveness	Usually recursive, even Turing-complete	Feature-value, not Turing-complete
Learning bias	Using background knowledge and constraints	Using prior distributions, parameters and features
Evaluation	Diverse criteria, including simplicity, comprehensibility	Oriented to error (or loss) minimisation
Validation	Code inspection, divide-and-conquer debugging, background knowledge consistency	Statistical reasoning (only a few techniques are locally inspectable)

Table 2.1: Inductive programming comparison with other machine learning paradigms (Source: [17])

Inductive programming is especially useful when the number of examples is small but the hypothesis space is large as in Turing-complete languages [17]. The background knowledge acts as a powerful explicit bias that can reduce drastically the search space [17]. By using a declarative approach, the background knowledge can be represented with a single language, increasing user accessibility and facilitating the knowledge revision and inspection [17].

```
\f -> ?
f "hello" == "olleh"
the predicate is f "hello" == "olleh"


reverse
```

Figure 2.6: Program induction with one example using MagicHaskeller

"Inductive programming is essentially a search problem" [17]. The size of the solution in terms of the functions used by an inductive programming system to provide a solution is known as the $depth(d)$, while the number of functions that comprise the hypothesis space (where the background knowledge may be included [21]) is known as the $breadth(b)$ [34]. In [34], the hardness of the hypothesis search is stated as $O(b^d)$, being almost constant on the number of examples. If the background knowledge does not con-

tain key auxiliary concepts, the search becomes difficult as the system needs to search within the rest of the functions in the hypothesis space [21]. On the other hand, if the background knowledge contains too many concepts, the hardness of the search might be affected too, with the increase of b [34, 21].

The inductive programming field can be divided into two main subfields, depending on the main language behind the systems: inductive logic programming and inductive functional programming. Some general-purpose inductive programming systems are Progol (logic), IGOR2 (functional), MagicHaskeller (functional), FLIP (functional and logic), Metagol (logic) and gErl (functional) [34]. As the languages behind these systems (Haskell, Prolog, etc.) have a high diversity of functions and possible combinations between them, the hardness of the problem to solve becomes problematic. This is a reason why DSL (Domain-Specific Language) systems may result appropriate as they considerably reduce the search space. However, the effort to create new DSLs for new applications or domains and the loss of accessibility are some of the drawbacks which might give preference to general-purpose systems [34].

CHAPTER 3
Experimental Design and Implementation

In this chapter, the settings, procedures and used tools for the conducted experiment will be explained. First, the universal language P3 will be introduced, as the language whose sampled concepts are to be explained in the experiment. Following this, the machine teaching setting applied to obtain the exemplar-based explanations will be characterised, and the settings of the experiment will be outlined. Finally, the different teaching scenarios and the tools and procedures employed are specified.

3.1 The P3 Language

P3 is an esoteric programming language composed of 7 instructions [42] as a variation of P'' [1], a Turing-complete language created by Corrado Böhm, using Brainfuck[2] (another P'' variation) syntax. Being Turing-complete or universal implies that it is capable of computing any computable function.

Programs written in P3 operate on an input-output tape divided into cells. The pointer moves are executed using the instructions '<' (to the left) and '>' (to the right) of the tape. The instructions '+' and '-' are used to change the value of a pointed cell. The instructions to perform loops are '[' and ']', being '[' the start and ']' the end of the loop. Finally, the last instruction 'o' is used to output the value of a pointed cell.

Following the special setting in [42], the alphabet consists of three symbols $\sum = \{0, 1, .\}$, so the P3-programs receive binary inputs and generate binary outputs. The special symbol '.' is used to control loop access and exit, and the halting of programs. If the instruction is '[' and the pointed cell value is '.' , the loop is not accessed, jumping to the next instruction after the corresponding ']' . If the instruction is ']' and the pointed cell value is '.' , the loop ends. Finally, if the instruction is 'o' and the pointed value is '.' , the program halts. Furthermore, '.' is used to pad the input tape before and after the binary input. For this work, it is considered that the input is left-padded by just one cell.

The tape alphabet has the cycling order ('0' < '1' < '.' < '0'). When executing the instruction '+' to the pointed cell, the value changes to the next corresponding one following that order. For instance, if '+' is applied to '.' , the value of the cell will change to '0'. On the other hand, when executing '-' to the pointed cell, the value changes to the previous one following that order. For instance, if '-' is applied to '.' , the value of the cell will change to '1' .

[1] P'' https://en.wikipedia.org/wiki/P\T1\textquoteright\T1\textquoteright
[2] Brainfuck https://en.wikipedia.org/wiki/Brainfuck

The output tape is written in a write-only sequential manner when the instruction 'o' is executed, adding the value of the pointed cell. Once the program halts, the final output of a P3-program given a binary input is the corresponding value written on the output tape. A guided example, retrieved from [42], will be used to clarify the P3 behavior.

Consider the P3 program:

> [o >] < [<] > o

Consider now an input '10010' . The program starts with the pointer in the first position of the input string (with the value '1') and moves the pointer to the right (with the value '0'). As the value of the pointed cell is not '.' , the loop starts. Each value from the input string, starting at the second position, is printed in the output tape. In the last iteration, when the pointer moves to the right of the last character of the string, the value of the pointed cell is '.' (as the input tape is left and right padded with '.'), so the first loop ends. After this, the pointer moves to the left, pointing to the last character of the string, and the second loop starts. It will finish when the pointer moves to the left side of the first character of the string. Finally, the pointer moves to the right, pointing the first character of the string, and its value is printed. Then, the program halts since there are not more instructions left to execute. The output tape has a final value of '00101' . Basically, the program skips the first bit, outputs the rest and goes back to output the skipped bit at the end.

A P3 Simulator written in Python, implemented for this work, is located in the Appendix A.

3.2 Machine Teaching Setting

The P3-based concepts to teach in the experiment will be obtained by reusing the Teaching Book computed in [42][3].

The considered example set space W is restricted to example sets with at most 7 alphabet characters (in this case, the alphabet is $\sum = \{0, 1\}$) without contradictory examples, repeated pairs or sets where all the outputs are empty, since they identify the empty program. Hence, the maximum teaching size allowed is 7. With the previous considerations, the example set space is comprised of 17,252 example sets.

The program space P is also restricted, without considering programs which do not include the output instruction 'o' and those with an unbalanced number of brackets (i.e. different number of '[' instructions and ']' instructions). Furthermore, a fixed-time-limit function is applied, as some programs may not halt.

By executing Algorithm 3.1, where the given length-lexicographic order $\prec$ (see Equation 2.9) for the considered program space is { < > + - [ ] o }, 5062 distinct programs are retrieved. For each example set, a program is found, so different example sets are consistent with the different programs. The optimal witness set for a program p will be considered as the example set associated with the first appearance of the program p in the file "res7.txt" [3], assuming that the orders presented in the file are equivalent to the orders $\lhd$ and $\prec$ in the Algorithm 3.1.

[3]The software related to the work in [42] is available at https://github.com/ceferra/The-Teaching-Size-with-P3

Program	Witness Set	Teaching Size
- [- o]	{('', '0'), ('0', '')}	2
+ [+ o o o]	{('', '111'), ('1', '')}	4
[o >]	{('010', '010')}	6
[> o o]	{('101', '0011')}	7

Table 3.1: Some programs, witness sets and their teaching sizes from "res7.txt"

Algorithm 3.1 Preprocessed Teaching Book filling algorithm

TeachingBook = []
for all witness sets $w \in W$ in increasing $\delta(w)$ and ordered by $\lhd$ for equal size **do**
  for all program $p \in P$ in increasing $l(p)$ and ordered by $\prec$ for equal size **do**
    if (for each $\langle i, o \rangle$ in w, $p(i) = o$**) then**
      insert $\langle w, p \rangle$ in *TeachingBook*
      exit for
    end if
  end for
end for

Note that Algorithm 3.1 assumes the use of a perfect model of the learner when the learning prior and the sampling prior are aligned. So a learner using the same representational language (in this case, P3), the same coding and assuming the strong simplicity priors, can unequivocally identify a target program once it has been given the associated witness set by the teacher.

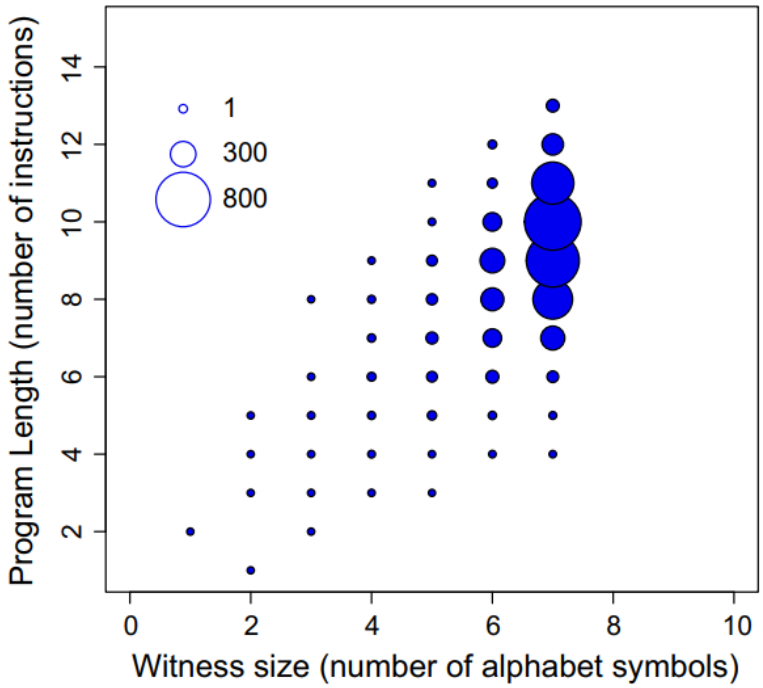

Figure 3.1: Scatter plot of programs according to their witness size (Source: [42])

3.3 Experiment Setting

The experiments in [22] showed how minimising the witness set can be too extreme for human learners to identify a given concept, especially as the internal representation mechanisms and codings differ from the learner model considered in the machine teaching setting.

The main objective of the experiment undertaken in this work is to evaluate the effectiveness of the explanations based on the witness sets, obtained with the aforementioned machine teaching setting, when the learners are both humans and machines with different representational languages (even if they share the strong simplicity priors) and to capture the learning progress when additional or redundant examples are provided. For these purposes, test examples (which include a given input, but do not include the corresponding output) will be used to evaluate the learning progress.

Once the source of the concepts and the machine teaching setting used to obtain the witness sets have been identified, several decisions must be taken in order to establish the experiment setting. Specifically, the following points must be considered:

- Number and concepts to teach.
- Number and source of the additional or redundant examples to provide.
- Example presentation method: individual (only one example is presented in each learning phase) or batch (multiple examples are presented in each learning phase).
- Number and source of the test examples for the evaluation.

The decisions are highly influenced by and focused on the human scenario, as it is the scenario that is hardest to control. The size of the experiment must be reasonable so the participants finish it without suffering from overload, affecting their performance and therefore, the results.

All of the supporting calculations have been performed with the use of Google Colaboratory[4]. Google Colaboratory or "Colab" is a free tool based on the open source project Jupyter Notebook, which allows the users to dynamically run Python code in a client-server application fashion, by benefiting of the cloud storage and computation of the Google Virtual Machines.

3.3.1. Experiment Characterisation

Concept Sampling

As explained before, the size of the experiment must be set considering human participants. It is believed that a reasonable number of different concepts to teach could range between 10 and 15, so the learning of an acceptable number of concepts can be analysed.

Another consideration is the sampling criterion. One option might be sampling randomly 10 to 15 concepts from the 5062 different programs that are found in the file "res7.txt". In Figure 3.2, it can be observed that the majority of the programs are concentrated between sizes from 6 to 8 alphabet characters. By performing a uniform probability sampling, concepts between this range will have more probability to be included in the experiment, while concepts with sizes of 1 or 2 will not probably have presence. As

[4]The detailed followed procedures can be reviewed in https://github.com/gonzalojaimovitch/P3-Machine-Teaching/blob/master/General/GeneralExperimentSettingProcessing.ipynb

it can be interesting to have different program complexity in terms of the size (note that greater program size complexity does not imply that a program is harder to learn), other approaches are considered.

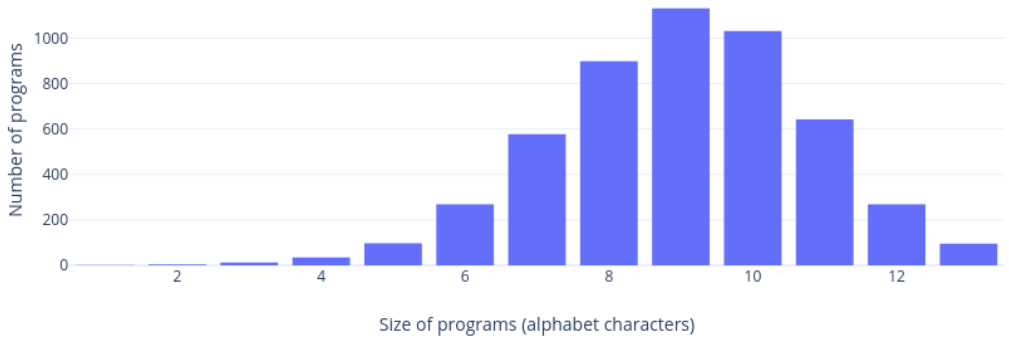

Figure 3.2: Bars plot of programs according to their size in "res7.txt"

One approach, closely related to the decisions about the source of the redundant examples and the test examples for the experiment, is to sample concepts which have a minimum number of different examples within their associated witness sets, returned when executing Algorithm 3.1. For instance, the program '+o' has the optimal witness set {('', '0')}, as the first example set following $\delta(w)$ and $\triangleleft$ returned as a pair $\langle w,$'+o'$\rangle$ when the teacher executes Algorithm 3.1. However, other example sets identified that program: {('0', '1')}, {('', '0'), ('1', '')}. Considering the machine teaching setting, these examples can be accounted as "highly informative" since, alongside the witness set, they can help the perfect learner to identify the concept. Hence, it seems interesting to use these examples as the redundant examples (in addition to the witness set) to support the learning of the concepts by the learner, and possibly as the test examples to evaluate the learning progress.

The above-explained approach will be the chosen one for the concept sampling. Since the number of additional or redundant examples will be 5, and the test examples to complete by the learners will be 5 too, the sampled concepts from "res7.txt" must fulfil the requirement of having 10 or more different examples within their associated witness sets (without considering the examples in the optimal witness set). Figure 3.3 represents the number of different programs in "res7.txt" according to their size that fulfil this requirement. Programs with 10 or more alphabet characters do not fall in this space. Also, the number of possible programs to sample decreases from 5062 to 170.

Even if these considerations imply a high bias when selecting the programs for the experiment, trying to have a good representation of different complexity degrees (independently of the complexity effect when teaching concepts) forces a biased sampling, which will be considered as acceptable for this work.

Finally, the programs are randomly sampled among programs with the same size that fulfil the above-mentioned requirement (e.g. the program with size 1 in the experiment is sampled from the set of programs with size 1 in "res7.txt" which have at least 10 different additional examples within their associated witness sets without considering the optimal witness set), obtaining a total of 9 different programs for the experiment (see Table 3.2).

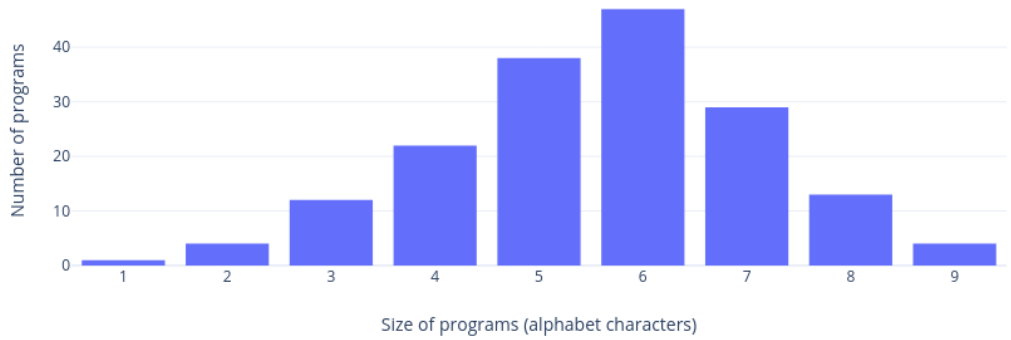

Figure 3.3: Bars plot of programs with 10 or more different examples within the possible witness sets (without considering the optimal witness set) according to their size

Program	Size
o	1
>o	2
>+o	3
o+oo	4
>>>-o	5
>-[o<]	6
-[-<]>o	7
+[>+o<+]	8
-[-[o<-]]	9

Table 3.2: P3 Sampled Programs for the experiment

Redundant examples

The additional or redundant examples, as explained in the previous subsection, are sampled among the different examples comprising the possible witness sets of the sampled programs, without considering the examples which are already in the optimal witness set. The chosen number of additional examples is 5, keeping in mind again the human scenario.

Other alternatives were considered, like simply sampling examples from a distribution of inputs (e.g. inputs with 1 to 10 alphabet characters), or choosing examples not included in the selected witness set of a program in non-decreasing size and following $\triangleleft$ for breaking ties in cases of same size. This last option resembles the simplicity priors described in Section 2.3.3. Nonetheless, these examples might have less informational power than those which are part of the possible witness sets for a concept. This is the reason why the above-explained approach is the chosen one.

Individual or batch approach

Once again, considering humans as the main focus for the experiment characterisation, the batch approach is chosen in opposition to the individual approach. Having optimal witness sets with maximum dimension of 4 examples, and considering the 5 redundant

examples, means that an individual evaluation of the learning progress would need 9 different evaluation phases for each concept independently of the number of test examples (in the worst case). Even if the witness set examples are provided in a batch manner, the resulting 6 different evaluation phases for each concept to teach would imply a total of 54 evaluation phases for teaching the 9 concepts of the experiment. Therefore, the teaching examples for each concept will be presented in 3 different batches: (1) first batch comprising the witness set, (2) second batch comprising 2 additional examples and (3) third batch comprising the remaining 3 additional examples. This way, the final number of learning phases is reduced to 27.

In Section 2.2, the sensitiveness of humans to the sequential order of the given examples in a teaching scenario is mentioned. For these experiments, the deterministic approach will be to present the witness set in the same order of examples found in "res7.txt", and the redundant examples following the order of appearance in "res7.txt".

Test examples

The first considered option for the source of the test examples, equal to the one followed for the additional examples, was to use examples from the possible witness sets of the sampled concepts without considering the examples included in the optimal witness set. Only the input of these examples would be presented to the participants, and the hidden output will be used to evaluate the learning progress.

However, in an advanced phase of the project, when most of the scenarios were already implemented, a different approach was suggested. The main reason why a new approach was considered is the fact that the above-mentioned source of the test examples could be not representative of a more realistic input space, and a wrong conception of the learning progress could be derived. For instance, considering a concept in "res7.txt" which generates a specific output for inputs starting with '0' and a different one for inputs starting with '1', and the example space (of examples within the possible witness sets for that concept without considering the examples in the optimal witness set) comprised only by examples whose inputs start with '0', a proper evaluation of examples whose inputs start with '1' cannot be performed.

Hence, the final approach, aiming to sample test examples whose inputs are more representative of a real input scenario considering the experiment domain, is to generate all the possible binary inputs until a given size (5 in this case, as a human-limitation adequate boundary) and perform the sampling of 5 inputs to be part of the test examples. This approach requires the use of an output generator given the concept and the sampled inputs. The developed mapping tool (or P3 Simulator) is given in Appendix A.

A new sampling of the concepts was not performed due to the advanced stage of the project. Only programs with 10 or more different examples (5 for the additional examples and 5 for the test examples) in "res7.txt" (without including the examples in the optimal witness set) were considered, and by applying this approach, programs with 5 or more different examples in "res7.txt" could be considered (see Concept Sampling).

3.3.2. Teaching Protocol

The proposed teaching protocol for a concept lesson is the following:

1. Present the witness set to the learner.
2. Ask the learner to complete the test examples (only the input is presented).

3. Present the first 2 additional examples (alongside the witness set) to the learner and repeat Step 2.
4. Present the last 3 additional examples (alongside the witness set and the first 2 additional examples) to the learner and repeat Step 2.

With the addressed decisions about the experiment and after performing the required data processing and samplings, the concepts to teach, teaching examples ordered by teaching phase and the test examples are represented in Table 3.3.

3.4 Teaching Scenarios

In this work, four different learners have been considered in order to evaluate the effectiveness of the machine teaching setting applied, where a machine is the teacher, and humans and other machines are the learners:

1. Mathematical Expectancy Learner.
2. Human Learner.
3. MagicHaskeller (Inductive Functional Programming System) Learner.
4. GPT-2 (Transformer Neural Network) Learner.

The different scenarios and the tools supporting the specific requirements of each scenario are explained in the following subsections.

3.4.1. Mathematical Expectancy Scenario

This scenario is inspired by the interest of analyzing how good average learner performance could score in the experiment by using the outputs observed on the teaching examples, considering each output frequency from a mathematical expectancy perspective.

Formally, the Mathematical Expectancy Scoring Algorithm can be defined as follows. Let S be the teaching set provided to the learner, comprised of input-output pairs $\langle i, o \rangle$. Let T be the test examples set used to evaluate the performance of the learner, comprised of input-output pairs $\langle it, ot \rangle$. Let $\Pi(S)$ be a function returning the different outputs in S alongside the percentage of occurrence of the returned outputs in S pairs, in the form of output-frequency pairs $\langle od, f \rangle$. With all this, the resulting algorithm (see Algorithm 3.2) returns the mathematical expectancy score.

In plain words, one could imagine an infinite or big enough number of learners which will return, for each test example, a binary string (or empty string) following the output or answer distribution based on the output frequency of the teaching examples. For instance, if the teaching set is comprised of the pairs {('0', '0'), ('01', '1')} and the test example set is {('001', '0'), ('10', '1')}, the learners will answer 50% of the times '0', and '1' the other 50% of the times for both the first and the second test examples, obtaining a score of ((1 / 2) * (0.5)) + ((1 / 2) + (0.5)) = 0.5. Note this score can be understood as the average accuracy or the percentage of average correct outputs provided for the test examples.

The Python script with the code for computing the Mathematical Expectancy Learner scenario is located in the Appendix C.

[5] In Appendix B, a translation of the P3 Programs into Decision Rules can be found

Experiment Setting					
Concept Learning Phases = 3					
Teaching Approach: Batch Teaching					
		(| w |)	(2)	(3)	(5)
Id	P3 Program[5]	Witness Set	Additional Set I	Additional Set II	Test Examples
C1	o	{('0','0')}	{('111001','1'), ('110101','1')}	{('100110','1'), ('01010','0'), ('111100','1')}	{('00000','0'), ('11100','1'), ('00111','0'), ('11010','1'), ('0010', '0')}
C2	>o	{('10','0')}	{('01000','1'), ('01010','1')}	{('1011','0'), ('00','0'), ('001','0')}	{('01011','1'), ('0101','1'), ('0010','0'), ('100','0'), ('1','')}
C3	>+o	{('','0'), ('01','')}	{('010100',''), ('10101','1')}	{('010',''), ('100','1'), ('011101','')}	{('00010','1'), ('110',''), ('00111','1'), ('11000',''), ('101','1')}
C4	o+oo	{('0','011')}	{('10','1'), ('001','011')}	{('00','011'), ('0001','011'), ('000','011')}	{('01011','011'), ('0101','011'), ('0010','011'), ('100','1'), ('1','1')}
C5	>>>-o	{('','1'), ('1110','')}	{('10','1'), ('111001','')}	{('11','1'), ('11100',''), ('0001','0')}	{('01011','0'), ('110','1'), ('0010',''), ('101','1'), ('1000','')}
C6	>-[o<]	{('0','10'), ('00','')}	{('11','01'), ('10','')}	{('101',''), ('','1'), ('000','')}	{('01','00'), ('0000',''), ('00011',''), ('0011',''), ('1000','')}
C7	-[-<]>o	{('','0'), ('0',''), ('00','0')}	{('0001','0'), ('01','1')}	{('0101','1'), ('0010','0'), ('0110','1')}	{('01011','1'), ('0000','0'), ('00000','0'), ('100',''), ('1000','')}
C8	+[>+o<+]	{('','01'), ('01',''), ('1','')}	{('11',''), ('011','')}	{('10',''), ('0','0'), ('0100','')}	{('10101',''), ('11101',''), ('00000','1'), ('0011','1'), ('1111','')}
C9	-[-[o<-]]	{('','010'), ('0',''), ('1','')}	{('100',''), ('110','')}	{('101',''), ('10',''), ('11','')}	{('10101',''), ('0100',''), ('00000',''), ('01000',''), ('1111','')

Table 3.3: Experiment Setting

Algorithm 3.2 Mathematical Expectancy Scoring Algorithm

```
score = 0
X = Π(S)
for all test examples t ∈ T do
  for all output-frequency pairs x ∈ X do
    if t(ot) is equal to x(od) then
      score += (1 / |T|) * x(f)
    end if
  end for
end for
```

3.4.2. Human Learner Scenario

As mentioned before, the scenario where humans are the learners has a special relevance in this work. This scenario sets the link between the fields of machine teaching and explainable AI. Insights obtained from these experiments are expected to shed more light on the path to build better human explanations so AI solutions like the ones based in machine learning models become trustworthy, and therefore, applicable.

Due to its importance, the design of the experiment has been mostly shaped considering an appropriate configuration so human learners were able to perform the experiment tests. In contrast to machines, which (within their limitations) will perform the demanded task as far as they are plugged into power supply and no other affecting eventualities occur, humans are affected by (among others) memory retrieval limitations and emotions (see Section 2.2).

Inspired by the human limitations (especially the possible affecting emotions of indifference or lack of motivation), a small prize is linked to the experiment best score, in an attempt to get more commitment and interest from the participants. The score metrics will be explained in Chapter 4.

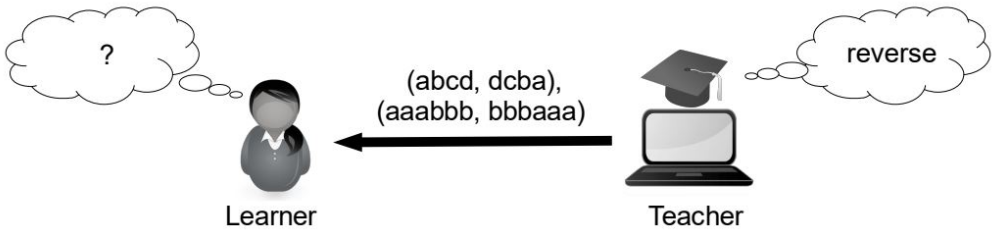

Figure 3.4: Teaching scenario with human as the learner and machine as the teacher (Source: [22])

Due to the current situation happening during the development of this work (COVID-19 pandemic), the test must be designed and performed using an online platform. The main identified requisites for the application used to implement and deploy the test are the following: (1) multiple-pages and (2) disabling of the "Back" button. As the learning of concepts following the protocol described in Section 3.3.2 is divided into tested phases where the information is provided in an incremental manner, the experiment results can be manipulated by the participants (e.g. the participant is learning a concept an has fulfilled the test examples in the phase where just the witness set has been presented, or Phase 1; in the next phase where the witness set is presented alongside the first additional examples, or Phase 2, the participant's perception of the concept can be updated as more information has been given; finally, the participant decides to come back to the Phase 1 page and change the test answers with the updated concept perception).

After a thorough search among online form applications (see Google Forms, Microsoft Forms, Qualtrics Online Survey Software, etc.), only one free version platform fulfilling the above-explained requirements was found: JotForm.

JotForm

JotForm is an online form platform which offers 5 unlimited question forms and up to 100 submissions per month in its free subscription. Among its available features, multiple-language forms, conditional logic operations, save-and-continue-later option and unique submission control by checking cookies and IP direction are highlighted.

Other non-critical features were contemplated, such as the possibility to copy the responses from one concept learning phase to the subsequent ones, helping the human participants to fill out the test (in case the participant's concept idea has not been updated from one phase to another, the outputs provided by the participant should not change, so the participant would just need to go on to the next phase). This can be implemented with JotForm using the conditional logic operations.

Figure 3.5: Human experiment using JotForm: C1 learning process

Besides asking the participants to complete the outputs for the test examples in the different learning phases, an optional text box is presented in the final learning phase of each concept so the human participants can write in natural language the idea they have about the different learned concepts. By doing so, a manual check can be performed to discern if the participant has or has not properly learned the concept.

Finally, some demographic questions about the age and level of education (obtained or currently pursuing) are asked to the participants. The form is presented in both Span-

ish and English languages[6]. All participants gave informed consent in the treatment of the retrieved data.

3.4.3. MagicHaskeller Learner Scenario

MagicHaskeller is a general-purpose inductive functional programming system based on systematic exhaustive search [28], which is one of its main distinguishing features [19]. It infers programs expressed in Haskell from input-output examples (also expressed in Haskell), with the power of solving many problems using only one example [34] (see Figure 2.6).

When receiving an input x and its corresponding output y, MagicHaskeller returns the list of functions that satisfy $f(x) == y$. The boolean predicate in Haskell is expressed as: **f x == y**.

MagicHaskeller works in two phases: (1) the *Hypothesis Generation* and (2) the *Hypothesis Selection* [34]. The system is preset with a library of functions or primitives b. These functions can be combined in order to find all the hypothesis satisfying the $f(x)$ behavior. The maximum number of possible function combinations can be restricted with the parameter d. Furthermore, the hypothesis can be constructed using lambda abstraction.

The default function library is comprised of 189 Haskell functions. Among these functions, the definition of the constants '0' and '1' (which constitute the binary alphabet symbols) are included. For the experiments, the parameter d will not be modified, with '7' being the default value. No more functions will be included in the default library either.

```
the predicate is f "0" == "0"

id

reverse

take 1
take 2
take 3
\a -> concatMap (\_ -> a) a

takeWhile isDigit
filter isDigit
takeWhile isAlphaNum
takeWhile isPrint
dropWhileEnd isSpace
dropWhileEnd isControl
filter isAlphaNum
\a -> concat (words a)
dropWhile isControl
```

Figure 3.6: Fragment of the output file written by MagicHaskeller once given the witness set for C1

Considering this setting, and once MagicHaskeller has been installed[7] in an Ubuntu 16.04.7 LTS server, a Python script is used to generate the Boolean predicates required by MagicHaskeller (considering the examples to provide depending on the concept and the different learning phases), to create other required elements such as the desired output files, and to start the execution[8].

[6]The human experiment form can be reviewed in https://form.jotform.com/201955335123349

[7]MagicHaskeller http://nautilus.cs.miyazaki-u.ac.jp/~skata/MagicHaskeller.html

[8]MagicHaskeller Experiment GitHub Repository https://github.com/gonzalojaimovitch/P3-Machine-Teaching/tree/master/MHExperiments

Once the results are obtained (see Figure 3.6), the deterministic approach applied is to consider the first output function as the learned hypothesis from the MagicHaskeller learner. In Figure 3.6, it can be noticed that results are output in an incremental number of d, as the simplicity bias used by MagicHaskeller. The followed order in case of ties is unknown.

3.4.4. GPT-2 Learner Scenario

GPT-2 is a transformer-based language model trained on a dataset of 8 million web pages, with 1.5 billion parameters on its largest version. Basically, the model returns the next predicted word or token considering the previous words as the given context (see Figure 3.7). Its remarkable significance, among other facts, may be due to the state-of-the-art results obtained in domain-specific language modeling tasks with a "zero-shot" [9] setup. Some of the tasks GPT-2 is capable to perform are reading comprehension, machine translation, question answering or summarization. Furthermore, "fine-tuning" [10] can be applied to obtain more detailed control of the desired output [3].

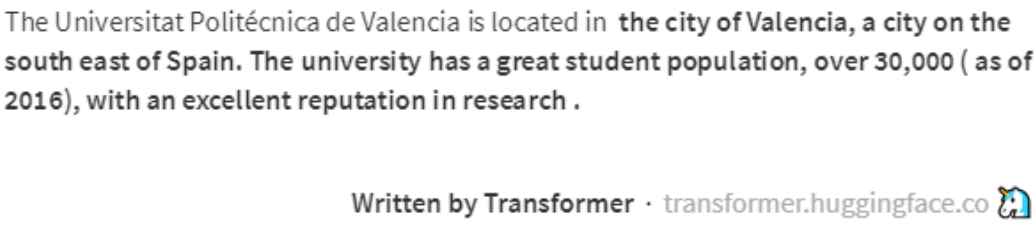

Figure 3.7: GPT-2 text completion using the web app from HuggingFace[11]

The chosen execution platform will be once again Google Colab. Furthermore, GPT-2 will be executed using the OpenAI's GitHub repository[12]. Note that some modifications on the GPT-2 source code must be undertaken for the purpose of the specific task on hand. The experiment Colab notebook and the cloned repository with the modifications and added scripts can be reviewed on the GitHub repository of this experiment[13].

The GPT-2 chosen model is the 774M-parameters model, aiming to use one of the largest available models considering the computational limitations of the environment. GPT-2 can be used in two different modes: (1) unconditional mode and (2) interactive mode. With the unconditional mode, no given input is considered, returning randomly some content. On the other hand, the interactive mode uses the console prompt to ask the user for some input, and then bases its results on the given context. The appropriate mode for the experiment is the interactive mode (as the teaching examples must be provided), with the modifications explained below.

One of the main considerations for this scenario is the form of the input which will be provided to GPT-2. There is no specified way to provide the inputs to the model, even though some ways of passing the inputs might help GPT-2 to return better results.

[9] For instance, in image classification, a zero-shot learning approach would be classifying the image of a zebra when this class did not appear in the training set used to build the model

[10] Start from a pre-trained model and use a new dataset to continue with the training (considering the original training set is not drastically different from the new one)

[11] HuggingFace "Write With Transformer" https://transformer.huggingface.co/doc/distil-gpt-2

[12] OpenAI's GPT-2 Repository https://github.com/openai/gpt-2

[13] GPT-2 Experiment GitHub Repository https://github.com/gonzalojaimovitch/P3-Machine-Teaching/tree/master/GPT-2Experiments

The final input form (not necessarily the optimal) will be the following one: 'Input1: *Input1Value*, Output1: *Output1Value*; (...); InputN: *InputNValue*, OutputN:'. The last input value corresponds with a test example input. With this setting, it is expected that GPT-2 will be able to connect the different examples provided as they follow an enumerated series. Therefore, the returned value will be treated as the output for "InputN" considering the N-1 previous labeled examples.

A Python script is used to generate the different queries based on the different concepts and learning phases (e.g. the query for the first test example when teaching C1 with just the witness set would be 'Input1: 0, Output1: 0; Input2: 00000, Output2:'). The empty string is represented using the '-' symbol. Once all the 135 queries have been generated, the Python script "nointeractive.py" (as a modification of the original "interactive_conditional_samples.py") is executed.

The default hyperparameters of the model are preserved, with the exception of the 'length' (of the output returned), which is modified to return a given number of 3 tokens. By doing so, the experiment execution performance is improved, without asking GPT-2 for more information than the required (generally). If 'length ' is set to 1 or 2, some incomplete answers might be returned.

The output results from GPT-2 are not consistent for the same given input, as the power of generating natural-appearance text with the model depends on the use of random sampling among the possible results, not just returning the highest-probability next word given the context. Nonetheless, deterministic results can be obtained by setting the 'top-k' parameter to 1, but the results are poor. Hence, the random setting will be applied by using the default 'top-k' value.

As the obtained results derive from a random environment, two different approaches will be considered. In both approaches, the model will be executed n times for the same query. Only outputs (starting from the left of the returned string) formed by the alphabet symbols {0,1} or having '-' as the first element will count as correct answers. Therefore, some output transformations are required (e.g. an output '010xyz' is transformed to the correct answer '010' ; an output 'xyz010' is not considered since it starts with elements that are not part of the accepted alphabet and the first element is not '-'; an output '- 1 z' is transformed to the correct answer '-').

Most Frequent Output: gpt2-ensemble

In the first approach, the most frequent result (out of the n results obtained for the same query) will be used as the final answer from GPT-2. In case of ties, a simple random selection will be performed. For instance, if GPT-2 returns a set of outputs {00000, 0, 00000, 01} (supposing $n = 4$) for the query 'Input1: 0, Output1: 0; Input2: 00000, Output2:', the considered *Output2* will be '00000' as the most frequent output. This approach could be understood as an ensemble learning setting, specifically to the hard-voting predictor (see Figure 3.8), where n agents are considered and the most voted prediction is the system final output.

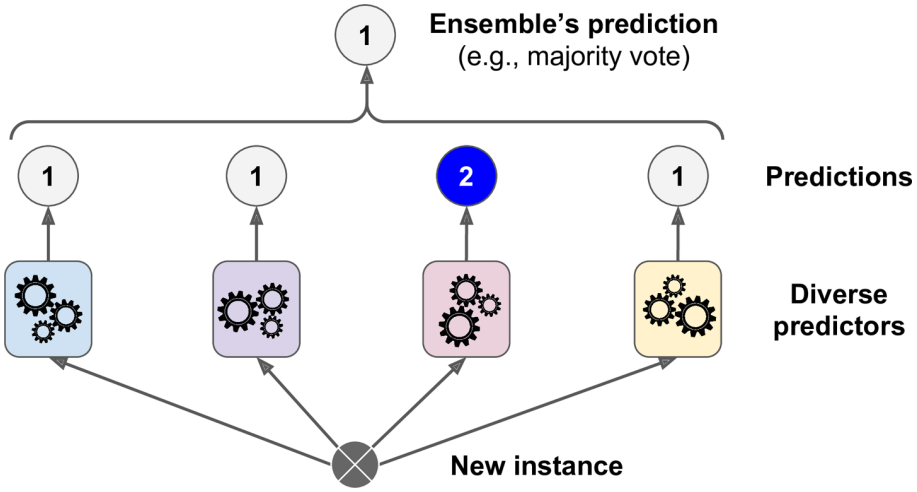

Figure 3.8: Hard voting predictor (Source: [15])

Average Performance: gpt2-expected

In the second approach, the n GPT-2 agents will be considered as independent agents. However, their performance on the task will be calculated as an average of the individual scores on the different learning phases when comparing to the other learners.

CHAPTER 4
Experimental Results

In this chapter, the results obtained through the undertaken experiments will be presented. For each learner scenario, the different concept learning curves will be plotted. Moreover, an aggregated learning curve with the average performance of the different learners is included, so a visual comparison among the learners can be performed. The performance measure used in this chapter is known as accuracy, i.e., the percentage of correct answers for the test set achieved by the learners. In some cases, as in the human scenario, the average accuracy will be plotted instead, since the performance of just one learner might not be representative of the learner-type behavior as a whole.

4.1 Mathematical Expectancy Results

Table 4.1 presents the mathematical expectancy accuracy considering a test output distribution based on the output frequency of each teaching set provided (regarding the different learning phases). This results will serve as the basis to compare the results of the rest of the students in each learning phase.

	C1	C2	C3	C4	C5	C6	C7	C8	C9
Witness Set	60%	40%	20%	60%	40%	40%	40%	40%	66,6%
Additional Set I	46,6%	40%	35%	53,3%	40%	40%	36%	48%	80%
Additional Set II	46,6%	40%	40%	56,6%	37,1%	45,7%	32,5%	45%	87,5%

Table 4.1: Mathematical expectancy accuracies for the tests

4.2 Human Results

A total number of 30 participants performed the test introduced in Section 3.4.2. The results, along with the demographic information and the natural language description of concepts provided can be found in the GitHub repository of this experiment[1].

The average accuracies obtained for the different test sets associated with the different concepts that form the experiment and considering the different learning phases are presented in Table 4.2.

[1]Human Experiment GitHub Repository https://github.com/gonzalojaimovitch/P3-Machine-Teaching/tree/master/HumansExperiments

	C1	C2	C3	C4	C5	C6	C7	C8	C9
Witness Set	1,3%	62,7%	30%	16%	38%	50%	37,3%	52%	89,3%
Additional Set I	60,0%	66,7%	27,3%	56,7%	36%	51,3%	45,3%	52%	96,7%
Additional Set II	88,7%	73,3%	21,3%	76%	38%	52%	52%	51,3%	96,7%

Table 4.2: Average accuracies for the tests scored by the human participants

4.3 MagicHaskeller Results

The first Haskell functions returned by MagicHaskeller in each learning phase are presented in tables 4.3, 4.4 and 4.5. Note that, in some cases, MagicHaskeller is not able to find a solution within the search space following the input-output behavior specified by the examples provided.

	Witness Set
C1	**id**
C2	**drop 1**
C3	**(\a -> filter (_-> null a) (show 0))**
C4	No result
C5	**(\a -> filter (_-> null a) (show 1))**
C6	No result
C7	**(\a -> foldr (_ _-> reverse (drop 1 (reverse a))) (show 0) a)**
C8	No result
C9	No result

Table 4.3: Haskell functions returned for the witness set of each concept using MagicHaskeller

	Additional Set I
C1	**take 1**
C2	**(\a -> drop 1 (take 2 a))**
C3	No result
C4	No result
C5	No result
C6	No result
C7	No result
C8	No result
C9	No result

Table 4.4: Haskell functions returned for the witness set and the first additional set of each concept using MagicHaskeller

	Additional Set II
C1	**take 1**
C2	**(\a -> drop 1 (take 2 a))**
C3	No result
C4	No result
C5	No result
C6	No result
C7	No result
C8	No result
C9	No result

Table 4.5: Haskell functions returned for the witness set, the first additional set and the second additional set of each concept using MagicHaskeller

After applying the above-presented functions to the corresponding test examples, the accuracies obtained are summarized in Table 4.6. As it can be observed, MagicHaskeller was not able to return a hypothesis for most of the concepts when redundant examples were provided.

	C1	C2	C3	C4	C5	C6	C7	C8	C9
Witness Set	0%	20%	40%	-	40%	-	0%	-	-
Additional Set I	100%	100%	-	-	-	-	-	-	-
Additional Set II	100%	100%	-	-	-	-	-	-	-

Table 4.6: Accuracies for the tests scored by the MagicHaskeller learner

4.4 GPT-2 Results

The results for both GPT-2 scenarios (gpt2-ensemble and gpt2-expected) are obtained following the procedure explained in Section 3.4.4 with $n = 40$.

gpt2-ensemble

Table 4.7 depicts the accuracies scored by an ensemble system (which returns the most repeated output and in case of ties performs a random selection) consisting of 40 GPT-2 individual predictors based on the 774M-parameters model.

	C1	C2	C3	C4	C5	C6	C7	C8	C9
Witness Set	40%	40%	40%	20%	20%	60%	40%	40%	100%
Additional Set I	40%	60%	0%	20%	60%	60%	60%	60%	100%
Additional Set II	60%	40%	20%	60%	60%	80%	60%	60%	100%

Table 4.7: Accuracies for the tests scored by the gpt2-ensemble learner

gpt2-expected

The average accuracies scored by the GPT-2 774M-parameters model, using a population (n) of 40 individual systems, are represented in Table 4.8.

	C1	C2	C3	C4	C5	C6	C7	C8	C9
Witness Set	18,5%	15,5%	15%	9,5%	32%	20%	32%	16,5%	32.5%
Additional Set I	50,5%	47%	19,5%	19%	38,5%	25,5%	36%	40%	70%
Additional Set II	54%	44.5%	34%	47,5%	41,5%	43,5%	45,5%	42%	68,5%

Table 4.8: Average accuracies for the tests scored by the gpt2-expected learner

4.5 Learning Curves

Once the results have been introduced, the learning curves are presented as the tool from where the analysis of each different learner scenario will be drawn, as well as the comparison among the different learners. The learning curve is a plot used to evaluate the evolution of learning as more examples are provided. They will also serve as the base for the conclusions stated in the next chapter.

4.5.1. Analysis of Scenarios

Hereafter, a plot for each different learner will be represented, with the accuracies obtained in the tests evaluating the learning of the different concepts.

Mathematical Expectancy Scenario

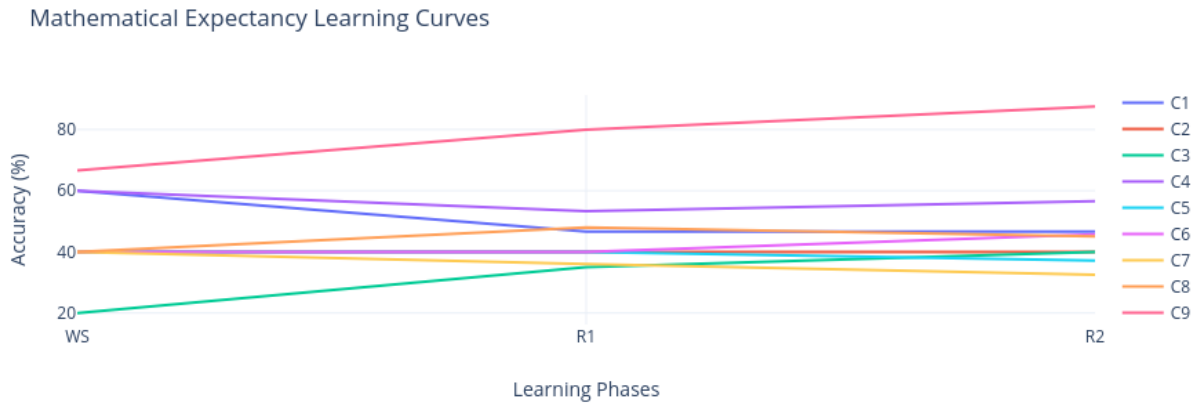

Figure 4.1: Mathematical Expectancy Scenario Learning Curves

Once the mathematical expectancy accuracies are plotted, the possibility of getting extremely high accuracies considering just the output frequencies of the teaching sets can be ruled out. This was a major concern, especially in the human learner scenario, as the results might present a spurious successful learning effect even if a learner was not appraising the input-output behavior of the examples provided. When additional examples are presented, only a few cases present a notable increase of the accuracies. Mostly, the learning progress stays constant or decreases.

Human Scenario

Figure 4.2: Human Scenario Learning Curves

For C3, the learners obtained a considerably lower score when receiving more examples. In the similar cases of C5 and C8, the score reduction can be considered as residual. For the rest of the concepts, the expected phenomenon is that learners maintain or increase their understanding of a concept when given more information, so they can refine their hypothesis.

In particular, for C1 or C4, the human participants show a remarkable evolution of their learning performance, starting with relatively low accuracies and improving their scores 60 or more points. Besides that, the highest accuracies in the experiment are the ones obtained for C9, with almost 90% of accuracy when showing just the witness set and rising to a score of almost 97% with the additional given information.

Considering a test set of 5 examples, accuracies below 100 points or so showcase the ineffectiveness of the used exemplar-based explanations for teaching the concepts to humans (even if it depends on the concept to teach and the broadness of the incorrect outputs provided for the text examples, i.e., the learned hypothesis might differ from the target concept but they may share a close behavior). Other interesting fact to mention is the observation that program complexity in terms of the size does not imply difficulty to learn, since some concepts (like C6 or C8) present better scores than shorter programs (like C3 or C5) when the witness sets are given to the learners. In addition, the highest complexity program in terms of the size, C9, is the program with the highest associated accuracy.

This trend seems to be closely related to the informational power of the examples (in terms of the universe coverage) depending on the concept to teach. For instance, for C9, there are only two possible outputs ('010' and ''). The second output is associated with the majority of the input universe values (with the exception of the input '' which returns '010'). Both outputs are presented on the witness set of that concept {('','010'), ('0',''), ('1','')}. On the other hand, for C3, just three elements of the input universe derive in the first type of output ('0'), approximately one half of the input universe derives in the second type of output ('1'), and the approximately other half of the input universe derives in the third type of output (''). Only two of the three possible outputs are presented with the witness set {('','0'), ('01','')}, making it almost impossible to ensure, for an imperfect learner (according to the machine teaching theory), the successful inference of the correct output for the unseen output cases. For a more comprehensible representation of

the concepts, the reader can find a translation of the P3 concepts into decision rules in Appendix B.

MagicHaskeller Scenario

Figure 4.3: MagicHaskeller Scenario Learning Curves

Results obtained with MagicHaskeller are quite poor, since the hypothesis for four concepts out of nine are not found in any of the learning phases. Focusing on the concepts for which some function was returned using MagicHaskeller, one of the main observations is that a full coverage of the concept (meaning accuracies of 100%) is obtained for the two concepts where the learning evolution can be analysed (C1 and C2), but only when additional examples were provided. This implies that the witness sets were not sufficient for teaching any of the concepts, and that the introduction of additional examples might help the teacher to accomplish the teaching goals.

gtp2-ensemble Scenario

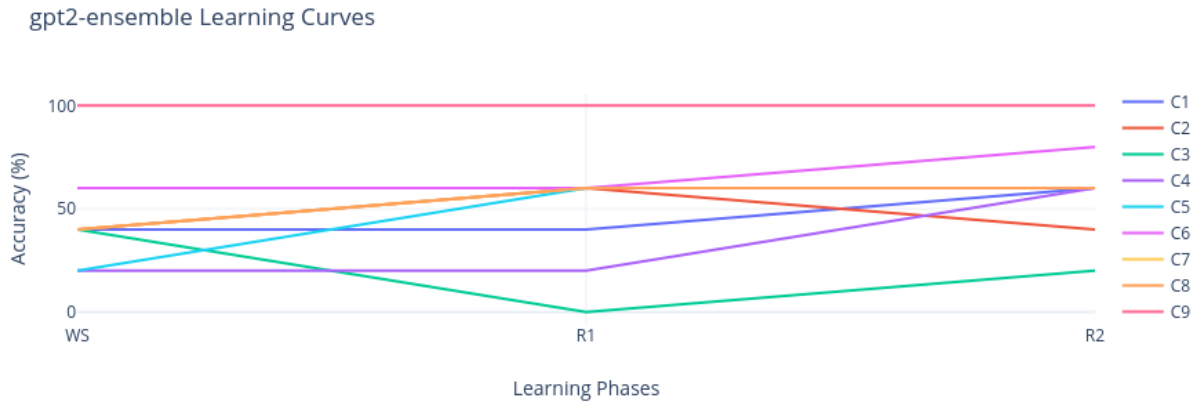

Figure 4.4: gpt2-ensemble Scenario Learning Curves

In contrast to the MagicHaskeller learner, the gpt2-ensemble system is capable of obtaining competing scores with this teaching setting. The learning curves present the expected

behavior of learning improvement or sustainment, even if some learning degradation happens for concepts like C2 or C3.

For one of the cases, specifically for C9, the witness set proves to be sufficient so the gpt2-ensemble learner fully learns the concept, maintaining the full accuracy when additional examples are provided. Even so, considering the general view of the experiment, the witness set is not a sufficient mean to successfully teach target concepts to the gpt2-ensemble learner.

gpt2-expected Scenario

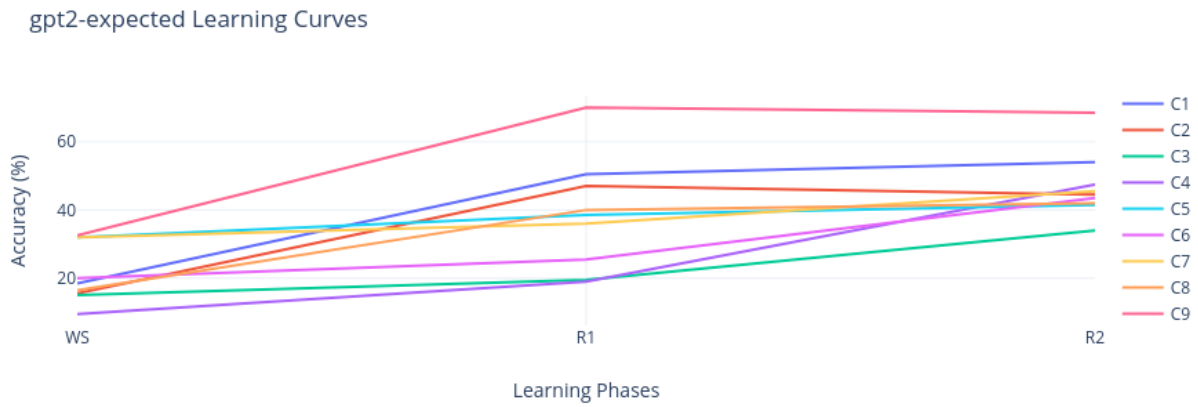

Figure 4.5: gpt2-expected Scenario Learning Curves

In this scenario, the accuracies are improved in all cases when redundant examples are provided. For C2 and C9, the unusual score reduction when the second additional set is given can be considered as residual.

No case shows the effectiveness of the witness set to fully explain a concept. Once again, the decoupling between learning difficulty and program complexity can be highlighted.

4.5.2. Learner Comparison

Since the learning behavior varies from concept to concept, the comparison will be based on the average learning performance (aggregated by concept) of the different learners considering the aforementioned learning phases.

When analysing the plot in Figure 4.6, important observations can be remarked. First of all, the average accuracy obtained by the different learners for the concepts in the experiment when being provided with the witness sets are below 50%. Furthermore, the accuracies obtained by the different learners in this learning phase are below the mathematical expectancy average accuracy, delivering a clear view of how the witness set obtained with the machine teaching setting applied in this work can be classified as insufficient for effective exemplar-based explanations. Even if the priors of the learners and the teacher were aligned, the different representational languages and codings of the learners from those of the perfect model used to extract the optimized witness set affect the teaching process in such a way that drives it into unsuccessful results.

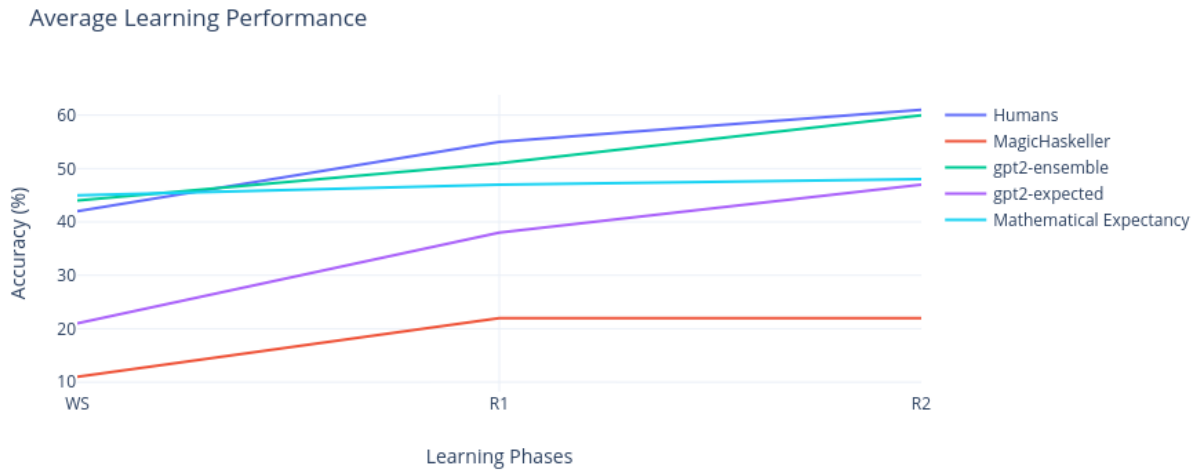

Figure 4.6: Average learning performance of the different learners in the different learning phases

The Mathematical Expectancy learner presents an almost constant learning progression when additional examples are provided. From the mathematical expectancy perspective, the fact that additional examples do not improve the average accuracy shows the high informational power contained in the witness sets obtained with the machine teaching setting applied in this work. Following the Mathematical Expectancy learner, the next highest accuracy for the witness set learning phase is obtained by the gpt2-ensemble learner, being almost the same as the obtained by the human participants.

In Section 2.2, it was underlined how humans are usually better than machines when generalizing from few examples. Surprisingly, in these experiments, a machine learning based system is capable of obtaining similar accuracies to the human ones in a few-examples learning scenario.

Although the introduction of redundant examples in the teaching process follows the tendency of improving the scores obtained by the learners, results seem still really far from the full concept learning intended. This phenomenon proves how the integration of additional examples to the witness set becomes a necessary mean so teaching might happen effectively in this environment. The questions then are how many additional examples are required and if the introduction of more and more additional examples will be sufficient for effectively teaching the concepts.

Results scored by the gpt2-expected learner display the most notable (positive) influence of additional examples in the experiment. However, these accuracies are considerably low in comparison to the human learner scenario or the gpt2-ensemble scenario, being below the mathematical expectancy average accuracy in every learning phase.

On the other hand, the poor results achieved by the MagicHaskeller learner appear to be closely related to the system's performance dependency on a proper domain background knowledge available, since in most of the cases no function combination could map the input-output behavior, and therefore no learning could be evaluated.

The reader can find the disaggregated learning curves by concept in the Appendix D.

CHAPTER 5
Conclusions and future work

The focus of this work was set on the specific case of teaching concepts with examples when a learner is not considered as the perfect model of the learner when using a machine teaching setting. The importance of aligning the priors and the representational languages becomes critical, as it determines the hypothesis order and preference given by both the teacher and the learner.

Therefore, results exposed in Chapter 4 show how the unalignment of the priors and/or the differences between the representational languages of the experiment learners and the P3 language used as the perfect learner model derive in the unfulfillment of the teaching goals. Furthermore, the given additional examples seem occasionally helpful to increase the concept perception of the learners, but sometimes still insufficient to fully teach a concept.

The special interest case, set on explaining concepts or models to humans in relation to the field of explainable AI, must face additional drawbacks. Humans are different, and so are their educational needs, making it harder to find a one-fit-for-all example set for the explanations. Moreover, emotions and physical state (in contrast to machines) might affect the teaching scenario even if the machine teaching setting tries to fulfil the human requirements using cognitive models for the witness set extraction. For instance, although a prize was linked to the human experiment, lack of motivation or tiredness could affect the performance of the human participants. However, results obtained in the human learner scenario[1] showcase how some participants were able to identify multiple concepts, specially when redundant examples were provided.

It seems remarkable that the gpt2-ensemble system presents a similar performance to that of the human learners on the witness set learning phase. One of the main differences between humans and machines underlined in the theoretical framework is that humans are generally better than machines learning from few examples since they are able to use their apparently broad background knowledge and select within it the specific knowledge required for the task in hand. With the introduction of additional examples, humans (as expected) become the highest-accuracy learners of the experiment. However, the NLP deep learning system GPT-2 still reveals a similar performance. This may make the reader think about the fact that this technology is close to overcoming humans in akin stages. What would the results have been if the recently introduced GPT-3 system from OpenAI would enter the competition?

With all this, future work on this field might be to experiment with different approaches for selecting the redundant examples (possibly obtained with machine teaching as in this work) and evaluate which of them are more effective when teaching concepts or

[1] Human Learner Scenario GitHub Repository https://github.com/gonzalojaimovitch/P3-Machine-Teaching/tree/master/HumansExperiments

models to human learners. Other pathway might be to investigate how many examples are required to effectively teach a concept or model, or even to test if the introduction of additional examples really helps the teacher to achieve the specified teaching goals.

It seems relevant to emphasise the importance of explaining the decisions of AI systems as they are being incrementally introduced in multiple human life stages. In his influential book, *21 Lessons for the 21st Century*, Yuval Noah Harari presents one clear scenario. A customer applies for a loan, but the AI system, which decides whether or not to grant it, finally determines that the customer is not eligible for that loan. Once the customer demands for an explanation asking why, the bank replies: "We don't know. There is no human which really understands the algorithm since it is based on advanced machine learning, but we trust our algorithm, so we will not grant you the loan" [18].

In the above-presented case, the AI system might be unfairly discriminating the loan applicant. If the decision chain could be observed, maybe a bad functioning of the model might be detected and accordingly corrected, or a proper explanation could be communicated to the customer including the reason or reasons why she can't be granted that loan. As similar situations will become more and more usual in the future, research towards effective (and maybe efficient) ways of explaining AI-based decisions will be key for the deployment and public acceptance of technology relying on those systems. One of the possible approaches, consisting of obtaining example sets with machine teaching for exemplar-based explanations, was evaluated in this work.

By writing this thesis, I was able to get in deeper touch with very different AI approaches such as GPT-2, MagicHaskeller or the emerging field of explainable AI. It also allowed me to stay abreast of very recent related achievements like the announcement of GPT-3, which promises to revolutionize multiple applications in the years to come. I would like to mention Sharif Shameem's JSX code generator, an application using GPT-3 API that generates JSX code pages with just a simple description of the layout[2].

Furthermore, I was able to perform human experiments, which helped me gain more insights about how to design, approach and overcome sensitive data and interaction scenarios, as a frequent task in areas such as user rating or technology acceptance. Finally, I got to work with Colab, as a promising environment for model execution on the cloud, and therefore, I got to accomplish one of my left pending tasks: learn Python.

[2]Sharif Shameem's JSX code generator https://twitter.com/sharifshameem/status/1282676454690451457

Bibliography

[1] Ethem Alpaydin. *Introduction to machine learning*. Third edition. Includes bibliographical references and index. 2014. ISBN: 0-262-32575-6.

[2] *AlphaGo | DeepMind*. https://deepmind.com/research/case-studies/alphago-the-story-so-far#our_approach. (Accessed on 07/31/2020).

[3] *Better Language Models and Their Implications*. https://openai.com/blog/better-language-models/. (Accessed on 08/10/2020).

[4] Tom B. Brown et al. "Language Models are Few-Shot Learners". In: (2020). arXiv: 2005.14165 [cs.CL].

[5] Antoine Buetti-Dinh et al. "Deep neural networks outperform human expert's capacity in characterizing bioleaching bacterial biofilm composition". In: *Biotechnology Reports* 22 (2019), e00321.

[6] Andriy Burkov. *The hundred-page machine learning book*. Vol. 1. Andriy Burkov Quebec City, Can., 2019.

[7] *Clustering Algorithms | Clustering in Machine Learning*. https://developers.google.com/machine-learning/clustering/clustering-algorithms. (Accessed on 07/31/2020).

[8] Sanjoy Dasgupta et al. "Teaching a black-box learner". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. Long Beach, California, USA: PMLR, Sept. 2019, pp. 1547–1555. URL: http://proceedings.mlr.press/v97/dasgupta19a.html.

[9] Derek Doran, Sarah Schulz, and Tarek R Besold. "What does explainable AI really mean? A new conceptualization of perspectives". In: *arXiv preprint arXiv:1710.00794* (2017).

[10] Brian Duignan. *Occam's razor*. Dec. 2018. URL: https://www.britannica.com/topic/Occams-razor.

[11] Cèsar Ferri, José Hernández-Orallo, and Jan Arne Telle. "Teaching Explanations by Examples". In: ().

[12] David B. Fogel. "The evolution of intelligent decision making in gaming". In: *Cybernetics and Systems* 22.2 (1991), pp. 223–236. DOI: 10.1080/01969729108902281. eprint: https://doi.org/10.1080/01969729108902281. URL: https://doi.org/10.1080/01969729108902281.

[13] Keith D. Foote. *A Brief History of Machine Learning - DATAVERSITY*. https://www.dataversity.net/a-brief-history-of-machine-learning/#. (Accessed on 07/30/2020).

[14] Ziyuan Gao et al. "Preference-based teaching". In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 1012–1043.

[15] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.

[16] Alison Gopnik. *The Ultimate Learning Machines - WSJ*. https://www.wsj.com/articles/the-ultimate-learning-machines-11570806023. (Accessed on 07/29/2020).

[17] Sumit Gulwani et al. "Inductive programming meets the real world". In: *Communications of the ACM* 58.11 (2015), pp. 90–99.

[18] Yuval Noah Harari. *21 Lessons for the 21 Century*. New York : Spiegel Grau, 2018.

[19] Robert Henderson. "Incremental Learning in Inductive Programming". In: *Approaches and Applications of Inductive Programming*. Ed. by Ute Schmid, Emanuel Kitzelmann, and Rinus Plasmeijer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 74–92. ISBN: 978-3-642-11931-6.

[20] José Hernández-Orallo. "Deep knowledge: Inductive programming as an answer". In: *Approaches and Applications of Inductive Programming (Dagstuhl Seminar 13502)*. Vol. 3. 2013, pp. 43–66.

[21] José Hernández-Orallo. *SMALL but DEEP: What can we learn from inductive programming?* "Approaches and Applications of Inductive Programming" Dagstuhl Seminar. 2013.

[22] José Hernández-Orallo and Cèsar Ferri. "Teaching and explanations: aligning priors between machines and humans". In: Oxford University Press, 2020. Chap. Unknown.

[23] Jose Hernández-Orallo and Jan Arne Telle. "Finite and Confident Teaching in Expectation: Sampling from Infinite Concept Classes". In: *24th European Conference on Artificial Intelligence (ECAI2020)*. 2020.

[24] Jose Hernández-Orallo and Jan Arne Telle. "Finite biased teaching with infinite concept classes". In: *arXiv preprint arXiv:1804.07121* (2018).

[25] Andreas Holzinger. "From machine learning to explainable AI". In: *2018 World Symposium on Digital Intelligence for Systems and Machines (DISA)*. IEEE. 2018, pp. 55–66.

[26] *inductive-programming.org • The IP Community | Introduction*. https://inductive-programming.org/intro.html. (Accessed on 08/01/2020).

[27] Uday Kamath, John Liu, and James Whitaker. *Deep learning for nlp and speech recognition*. Vol. 84. Springer, 2019.

[28] S. Katayama. "MagicHaskeller: System demonstration". In: *Proceedings of AAIP 2011 - 4th International Workshop on Approaches and Applications of Inductive Programming* (Jan. 2011), pp. 63–70.

[29] Miroslav Kubat. *An introduction to machine learning*. Springer, 2017.

[30] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. "Human-level concept learning through probabilistic program induction". In: *Science* 350.6266 (2015), pp. 1332–1338.

[31] *Machine teaching - How people's expertise makes AI more powerful*. https://blogs.microsoft.com/ai/machine-teaching/. (Accessed on 07/23/2020).

[32] Andrew N Meltzoff et al. "Foundations for a new science of learning". In: *science* 325.5938 (2009), pp. 284–288.

[33] Christoph Molnar. *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable*. https://christophm.github.io/interpretable-ml-book/. 2019.

[34] Lidia Contreras Ochando et al. “General-purpose Declarative Inductive Programming with Domain-Specific Background Knowledge for Data Wrangling Automation”. In: *CoRR* abs/1809.10054 (2018). arXiv: 1809.10054. URL: http://arxiv.org/abs/1809.10054.

[35] Kaustubh R Patil et al. “Optimal teaching for limited-capacity human learners”. In: *Advances in neural information processing systems*. 2014, pp. 2465–2473.

[36] *Perceptron Definition | DeepAI*. https://deepai.org/machine-learning-glossary-and-terms/perceptron. (Accessed on 07/30/2020).

[37] Gopinath. Rebala, Ajay. Ravi, and Sanjay. Churiwala. *An Introduction to Machine Learning [electronic resource]*. 1st ed. 2019. 2019. ISBN: 3-030-15729-6.

[38] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “" Why should I trust you?" Explaining the predictions of any classifier”. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 1135–1144.

[39] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “Model-agnostic interpretability of machine learning”. In: *arXiv preprint arXiv:1606.05386* (2016).

[40] Ayon Sen et al. “Learning to Read through Machine Teaching”. In: *arXiv preprint arXiv:2006.16470* (2020).

[41] Patrick Shafto, Noah D Goodman, and Thomas L Griffiths. “A rational account of pedagogical reasoning: Teaching by, and learning from, examples”. In: *Cognitive psychology* 71 (2014), pp. 55–89.

[42] Jan Arne Telle, José Hernández-Orallo, and Cèsar Ferri. “The teaching size: computable teachers and learners for universal languages”. In: *Machine Learning* 108.8-9 (2019), pp. 1653–1675.

[43] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[44] Alba Vicente Olmo. “Novel photonic switching components with non-volatile response for telecom applications”. PhD thesis. 2019.

[45] Xiaojin Zhu. “Machine teaching: An inverse problem to machine learning and an approach toward optimal education”. In: *Twenty-Ninth AAAI Conference on Artificial Intelligence*. 2015.

[46] Xiaojin Zhu et al. “An overview of machine teaching”. In: *arXiv preprint arXiv:1801.05927* (2018).

APPENDIX A
P3 Simulator (Python)

```
alphabet = {'1','0','.'};
operators = {'<','>','+','-','o','[',']'}

inList = []
progList = []

inPointer = 0
progPointer = 0

output = ''

def runP3(input, program):
  #Check if the given values are correct
  checkInput(input)
  checkProgram(program)

  #Create lists
  if input == '':
    inList = ['.']
  else:
    inList = list(input)
  progList = list(program)

  #Initialize pointers
  inPointer = 0
  progPointer = 0

  #Initialize left-padding controller
  leftpad = False

  while(True):
    #Check if program ends
    if progPointer > (len(progList)-1):
      return output
    pointedOp = progList[progPointer]

    #Choose Instruction
    if pointedOp == "<":
      if inPointer > 0:
        inPointer = inPointer - 1
      elif leftpad == False:
        #Not necessary to change the pointer since it will stay as 0
        inList.insert(0,'.')
        leftpad = True
      progPointer = progPointer + 1

    elif pointedOp == ">":
      if inPointer < (len(inList) - 1):
```

```
        inPointer = inPointer + 1
      else:
        inList.append('.')
        inPointer = inPointer + 1
      progPointer = progPointer + 1

    elif pointedOp == "+":
      val = inList[inPointer]
      if val == '0':
        inList[inPointer]='1'
      elif val == '1':
        inList[inPointer]='.'
      else: inList[inPointer]='0'
      progPointer = progPointer + 1

    elif pointedOp == "-":
      val = inList[inPointer]
      if val == '0':
        inList[inPointer]='.'
      elif val == '1':
        inList[inPointer]='0'
      else: inList[inPointer]='1'
      progPointer = progPointer + 1

    elif pointedOp == "o":
      if inList[inPointer] != '.':
        output = output + inList[inPointer]
      else:
        progPointer = len(progList)
      progPointer = progPointer + 1

    elif pointedOp == "[":
      if inList[inPointer] == '.':
        cond = len(progList)
        i = (progPointer + 1)
        while i < cond:
          if progList[i] == "]":
            progPointer = i
            cond = i
          i = i + 1
      else:
        progPointer = progPointer + 1
    elif pointedOp == "]":
      if inList[inPointer] != '.':
        cond = -1
        i = (progPointer - 1)
        while i > cond:
          if progList[i] == "[":
            progPointer = i
            cond = i
          i = i - 1
      else:
        progPointer = progPointer + 1


def checkInput(string):
  for c in string:
    if c not in alphabet:
      print("Incorrect input")
      exit()

def checkProgram(string):
  for c in string:
    if c not in operators:
```

```
113         print("Incorrect program")
114         exit()
```

APPENDIX B
Experiment P3 Programs and Decision Rules

Id	P3 Programs	Decision Rules[1]

1	o	`if input == '':`
		`  print('')`
		`else:`
		`  print(input[:1])`

2	>o	`if input == '' or input == '0' or input == '1':`
		`  print('')`
		`else:`
		`  print(input[1])`

3	>+o	`if input == '' or input == '0' or input == '1':`
		`  print('0')`
		`elif input[2] == '0':`
		`  print('1')`
		`else:`
		`  print('')`

4	o+oo	`if input == '':`
		`  print('00')`
		`elif input[1] == '0':`
		`  print('011')`
		`else:`
		`  print('1')`

[1]The lists are 1-indexed

5	>>>-o	`if input == '' or len(input) <= 3:`
		`  print('1')`
		`elif input[4] == '0':`
		`  print('')`
		`else:`
		`  print('0')`

6	>-[o<]	`if input == '':`
		`  print('1')`
		`elif input == '0':`
		`  print('10')`
		`elif input == '1':`
		`  print('11')`
		`elif input[2] == 0:`
		`  print('')`
		`elif input[:2] == '01':`
		`  print('00')`
		`else:`
		`  print('01')`

7	-[-<] >o	`if input == '':`
		`  print('0')`
		`elif input == '0' or input[:1] == '1':`
		`  print('')`
		`else:`
		`  print(input[2])`

8	+[>+o<+]	`if input == '':`
		`  print('01')`
		`elif input == '0':`
		`  print('0')`
		`elif input[:1] == '1' or input[:2] == '01':`
		`  print('')`
		`else:`
		`  print('1')`

9	-[-[o<-]]	`if input == '':`
		`  print('010')`
		`else:`
		`  print('')`

APPENDIX C

Script for the Mathematical Expectancy Learner (Python)

```
1
2  #https://www.geeksforgeeks.org/python-find-most-frequent-element-in-a-list/
       adapted
3
4  #Return the different-elements-of-a-list frequencies in a list of lists [value,
       frequency]
5  def list_frequency(passed_list):
6    num = ['']
7    for i in passed_list:
8      if not [i,passed_list.count(i)] in num:
9        if num[0] == '':
10         num[0]= [i, passed_list.count(i)]
11       else:
12         num.append([i, passed_list.count(i)])
13
14
15   return num
16
17 witnessSets = [[('0', '0')],[('10', '0')],[('', '0'), ('01', '')],[('0', '011')
       ],[('', '1'), ('1110', '')],[('0', '10'), ('00', '')],[('', '0'), ('0', '')
       , ('00', '0')],[('', '01'), ('01', ''), ('1', '')],[('', '010'), ('0', ''),
        ('1', '')]]
18 teachingSets1 = [[('111001', '1'), ('110101', '1')],[('01000', '1'), ('01010',
       '1')],[('010100', ''), ('10101', '1')],[('10', '1'), ('001', '011')],[('10'
       , '1'), ('111001', '')],[('11', '01'), ('10', '')],[('0001', '0'), ('01', '
       1')],[('11', ''), ('011', '')],[('100', ''), ('110', '')]]
19 teachingSets2 = [[('100110', '1'), ('111100', '1'), ('01010', '0')],[('001', '0
       '), ('00', '0'), ('1011', '0')],[('010', ''), ('100', '1'), ('011101', '')
       ],[('0001', '011'), ('00', '011'), ('000', '011')],[('11', '1'), ('0001', '
       0'), ('11100', '')],[('', '1'), ('101', ''), ('000', '')],[('0101', '1'), (
       '0010', '0'), ('0110', '1')],[('10', ''), ('0100', ''), ('0', '0')],[('10',
        ''), ('101', ''), ('11', '')]]
20
21 testSets = [[('00000', '0'), ('11100', '1'), ('00111', '0'), ('11010', '1'), ('
       0010', '0')], [('01011', '1'), ('0101', '1'), ('0010', '0'), ('100', '0'),
       ('1', '')], [('00010', '1'), ('110', ''), ('00111', '1'), ('11000', ''), ('
       101', '1')], [('01011', '011'), ('0101', '011'), ('0010', '011'), ('100', '
       1'), ('1', '1')], [('01011', '0'), ('110', '1'), ('0010', ''), ('101', '1')
       , ('1000', '')], [('01', '00'), ('0000', ''), ('00011', ''), ('0011', ''),
       ('1000', '')], [('01011', '1'), ('0000', '0'), ('00000', '0'), ('100', ''),
        ('1000', '')], [('10101', ''), ('11101', ''), ('00000', '1'), ('0011', '1'
       ), ('1111', '')], [('10101', ''), ('0100', ''), ('00000', ''), ('01000', ''
       ), ('1111', '')]]
22
```

```
#Given a list of outputs (or an output element), return the accuracy for the testList labels. The frequency of the different outputs is given, applying the appropriate weight for each case
def thirdAgent2(outputs, weights, testList):
  result = 0

  for j in range(0, len(testList)):
    for k in range(0, len(outputs)):
      if outputs[k] == testList[j]:
        result = result + (1/len(testList)) * weights[k]

  return result



numConcepts = 9
numPhases = 3

accuracyList = ['']*numConcepts

for i in range(0, numConcepts):
  accuracyList[i] = [0]*numPhases
  for i2 in range(0, numPhases):
    if i2 == 0:
      passed_list = [value[1] for value in witnessSets[i]]
    elif i2 == 1:
      passed_list = [value[1] for value in witnessSets[i]]  + [value[1] for value in teachingSets1[i]]
    else:
      passed_list = [value[1] for value in witnessSets[i]]  + [value[1] for value in teachingSets1[i]] + [value[1] for value in teachingSets2[i]]
    values = list_frequency(passed_list)
    calc_len = len(passed_list)
    print(str(i + 1) + " " + str(values))

    accuracyList[i][i2] = thirdAgent2([value[0] for value in values], [value[1]/calc_len for value in values], [value[1] for value in testSets[i]])

for i in range(0, len(accuracyList)):
  for j in range(0, len(accuracyList[i])):
    print("Concept " + str(i + 1) + " Phase " + str(j + 1) + ": " + str(accuracyList[i][j]) +  "%")
```

APPENDIX D
Disaggregated Learning Curves

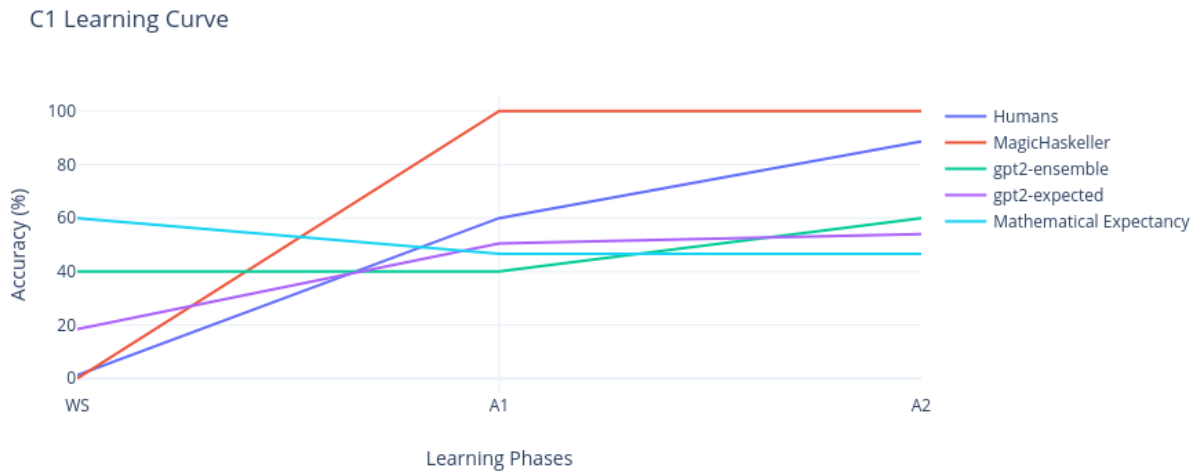

Figure D.1: C1 Learning Curves

Figure D.2: C2 Learning Curves

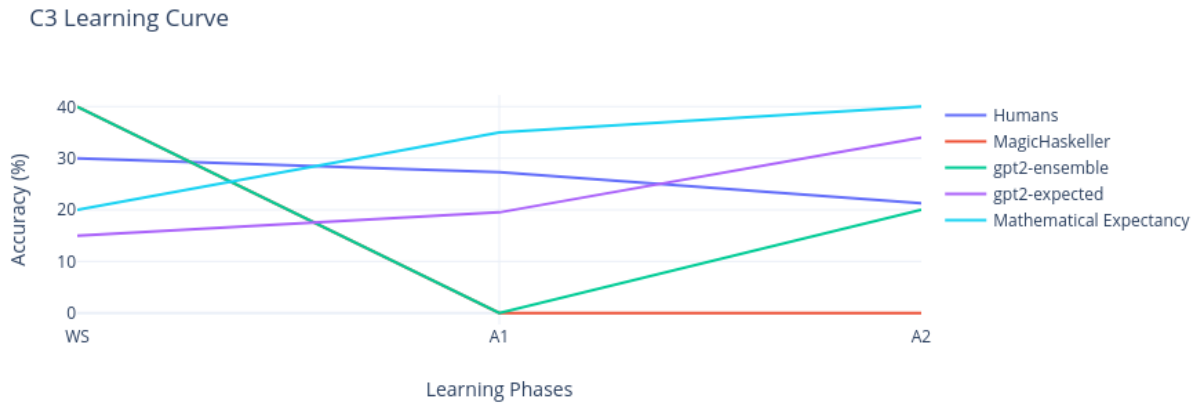

Figure D.3: C3 Learning Curves

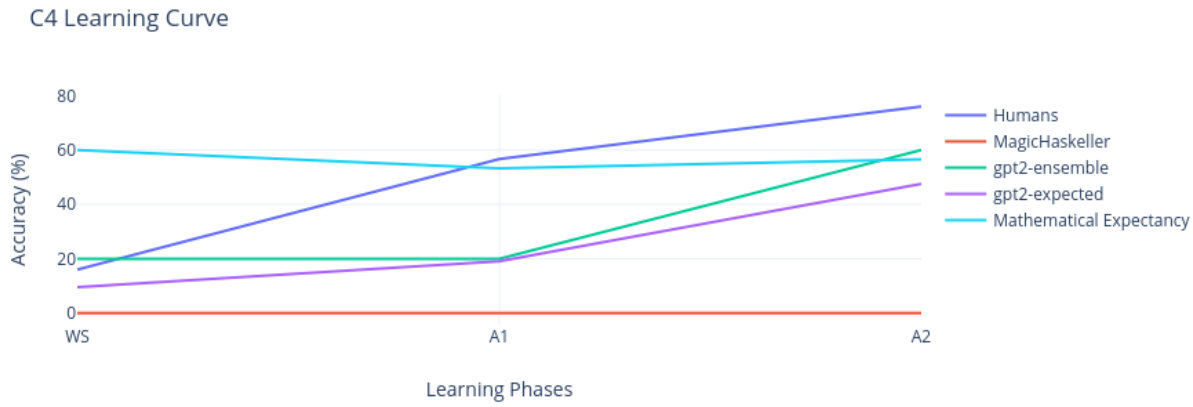

Figure D.4: C4 Learning Curves

Figure D.5: C5 Learning Curves

Figure D.6: C6 Learning Curves

Figure D.7: C7 Learning Curves

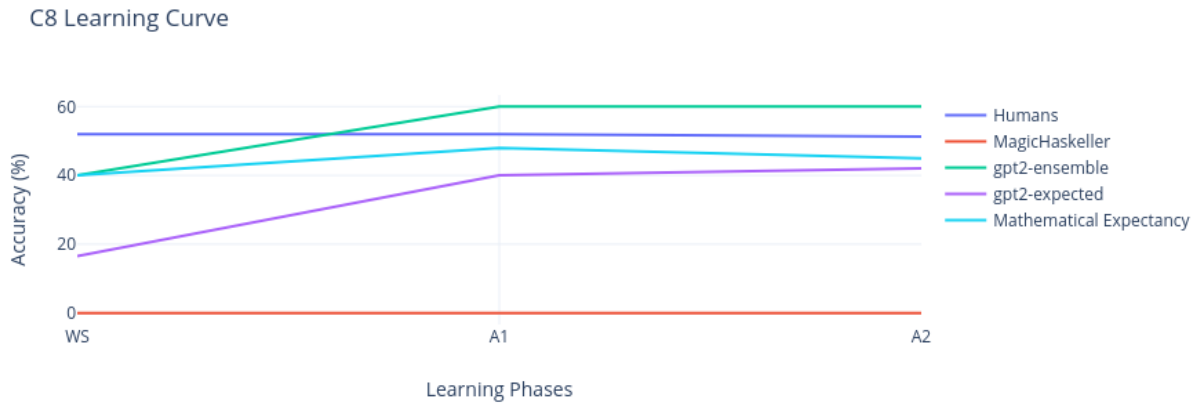

Figure D.8: C8 Learning Curves

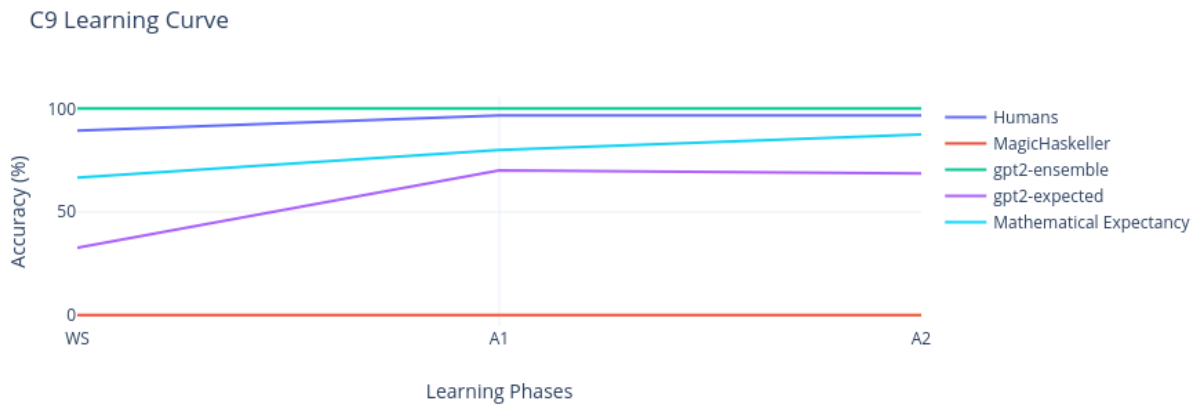

Figure D.9: C9 Learning Curves