

# Towards Argumentative Decision Graphs: Learning Argumentation Graphs from Data

Pierpaolo Dondio<sup>1</sup>

<sup>1</sup>*School of Computer Science, Technological University Dublin, Grangegorman Campus, D07 EWW4, Dublin, Ireland*

## Abstract

In this paper we present a novel data-mining model called argumentative decision graphs (*ADG*). An *ADG* is a special argumentation framework where arguments have a rule-based structure and an attack relation is defined among arguments. *ADGs* are graph-like models learnt from data in a supervised way that can be used for classification tasks. As in a decision tree, given a set of input features, an *ADG* returns the value of the target variable. Unlike decision trees, the output of an *ADG* can be also an undecided status, occurring when the model does not have enough reasons to predict a value for the target variable. This is due to the use of argumentation semantics to identify what arguments of an *ADG* are accepted and consequently make a prediction about the target variable. Unlike Bayesian Networks, *ADGs* are not required to be acyclic, but they can have any topology. Advantages of *ADGs* are the possibility of using different semantics to make predictions, the ability to deal with incomplete input data and to generate compact explanations. We evaluate a preliminary greedy algorithm to learn an *ADG* from data using public datasets and we compare our results with Decision Tree in terms of balanced accuracy and size of the model. Our results provide evidence to further progress our research.

## Keywords

Argumentation, Data Mining, Decision Tree, Graph models, Explainability

## 1. Introduction

In this paper we describe a novel data-mining model called Argumentative Decision Graphs (*ADG*). An *ADG* is a supervised data-mining model that learns the relationships between a target variable and a large enough set of examples. In this first paper we present a preliminary algorithm to learn *ADGs* from data for binary classification.

An argumentative decision graph is an extension of Dung's abstract argumentation graphs [1]. As in Dung's framework, each node of the graph represents an argument, the links represent an attack relation among the arguments and no support relation is defined. However, unlike Dung's abstract framework, arguments have a rule-based structure with a premise (called support) and a conclusion. Some arguments have a conclusion that can be used to predict the value of the target variable, while other arguments are not used to predict the target variable directly, but rather they are used to interact with other predictive arguments.

---


*AI<sup>2</sup>2021, 5<sup>th</sup> Workshop on Advances in Argumentation in Artificial Intelligence*

✉ pierpaolo.dondio@tudublin.ie (P. Dondio)

🆔 0000-0001-7874-8762 (P. Dondio)



© 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

The main reason to introduce *ADG* is the willingness to bring the advantages of symbolic non-monotonic reasoning into data-mining models. An *ADG* is an extension of an abstract argumentation framework, that is a symbolic reasoning system able to represent partial and conflicting knowledge and to formalize a large set of non-monotonic semantics. For this reason, an *ADG* can deal with partial information and make predictions when some of the inputs are missing.

The predictions of an *ADG* are made by applying an argumentation semantics. We first recall that an argumentation semantics is a set of postulates used to identify the set of acceptable arguments, called extensions. In the labelling approach [2] adopted in this paper, the effect of an argumentation semantics is to assign a label *in*, *out* or *undec* to each argument. This means that an argument can respectively be accepted, rejected or deemed undecided. The *undec* label represents a situation in which the semantics has no reasons to definitely accept or reject an argument. A first advantage of using argumentation semantics is that the various semantics offer a rich toolset to model different ways of making decisions. Grounded semantics, for instance, represents a skeptical acceptance strategy, while preferred semantics is a so-called *credulous* semantics that has less conservative conditions to accept an argument. A second advantage is that an *ADG* could output an *undecided* status. This means that an *ADG* can either make a prediction or abstain from making one. This happens in situations in which the input data create a conflict that cannot be resolved by the rules of the semantics and therefore there is no ground to take a decision. The undecided status provides an *ADG* with a way of quantifying the uncertainty generated by conflicting information. Third, the use of an attack relation between arguments coupled with an argumentation semantics generate compact and understandable explanations. In a distinctive non-monotonic fashion, a single argument can invalidate many others and directly or indirectly support a specific conclusion. Even in a large argumentation graph, the arguments that are responsible for a conclusion could be a small subset. This could make the explanations more compact and understandable.

Data-mining models that could be considered similar to *ADGs* are explainable models such as Decision Trees [3], or graph-like structures learnt from data such as Bayesian Networks [4]. In this paper, we provide an introduction to *ADGs* and we underline the differences between *ADG* and similar models. In the second part of this paper, we also provide a first greedy algorithm to learn an *ADG* from data and we evaluate the performance of such algorithm versus decision trees using benchmark classification datasets.

The paper is organized as follows. In section 2 we recall the basics of abstract argumentation semantics, in section 3 we introduce our *ADG* model, while in section 4 we critically compare them to decision trees. Section 5 describes an algorithm to learn *ADG* from data that is evaluated in section 6, while related works are described in section 7.

## 2. Abstract Argumentation Semantics

**Definition 2.1.** *An argumentation framework  $AF$  is a pair  $\langle Ar, \mathcal{R} \rangle$ , where  $Ar$  is a non-empty finite set whose elements are called arguments and  $\mathcal{R} \subseteq Ar \times Ar$  is a binary relation, called the attack relation. If  $(a, b) \in \mathcal{R}$  we say that  $a$  attacks  $b$ . An argument  $a$  is initial if it is not attacked by any argument, including itself.*

An argumentation framework  $AF = \langle Ar, \mathcal{R} \rangle$  identifies a directed graph. We define the restriction

of an argumentation framework to a set of nodes  $S$  as the framework  $AF_{\downarrow S}$  corresponding to the vertex-induced subgraph of  $AF$  identified by  $S$ :

**Definition 2.2.** Given  $AF = \langle Ar, \mathcal{R} \rangle$ , the **restriction** of  $AF$  to a set of nodes  $S \subseteq Ar$  is the argumentation framework  $AF_{\downarrow S} = \langle S, \mathcal{R}_S \rangle$  where  $\mathcal{R}_S = \mathcal{R} \cap (S \times S)$ .

An abstract argumentation semantics identifies a set of arguments that can survive the conflicts encoded by the attack relation  $\mathcal{R}$ . Dung's semantics require a group of acceptable arguments to be conflict-free (an argument and its attackers cannot be accepted at the same) and admissible (the set of arguments defends itself from external attacks).

**Definition 2.3.** A set  $Arg \subseteq Ar$  is **conflict-free** iff  $\forall a, b \in Arg, (a, b) \notin \mathcal{R}$ .

**Definition 2.4.** A set  $Arg \subseteq Ar$  defends an argument  $a \subseteq Ar$  iff  $\forall b \in Ar$  such that  $(b, a) \in \mathcal{R}, \exists c \in Arg$  such that  $(c, b) \in \mathcal{R}$ . The set of arguments defended by  $Arg$  is denoted  $\mathcal{F}(Arg)$ . A conflict-free set  $Arg$  is **admissible** if  $Arg \subseteq \mathcal{F}(Arg)$  and it is **complete** if  $Arg = \mathcal{F}(Arg)$ .

We follow the labelling approach of [2], where a semantics assigns to each argument a label *in*, *out* or *undec*.

**Definition 2.5.** Let  $AF = \langle Ar, \mathcal{R} \rangle$ . A **labelling** is a total function  $\mathcal{L} : Ar \rightarrow \{in, out, undec\}$ . We write  $in(\mathcal{L})$  for  $\{a \in Ar \mid \mathcal{L}(a) = in\}$ ,  $out(\mathcal{L})$  for  $\{a \in Ar \mid \mathcal{L}(a) = out\}$ , and  $undec(\mathcal{L})$  for  $\{a \in Ar \mid \mathcal{L}(a) = undec\}$ .

**Definition 2.6.** Let  $AF = (Ar, \mathcal{R})$ . A **complete labelling** is a labelling such that for every  $a \in Ar$  holds that:

1. if  $a$  is labelled *in* then all its attackers are labelled *out*;
2. if  $a$  is labelled *out* then it has at least one attacker that is labelled *in*;
3. if  $a$  is labelled *undec* then it has at least one attacker labelled *undec* and no attackers labelled *in*.

**Definition 2.7.** Given  $AF = (Ar, \mathcal{R})$ ,  $\mathcal{L}$  is the **grounded** labelling iff  $\mathcal{L}$  is a complete labelling where  $undec(\mathcal{L})$  is maximal (w.r.t. set inclusion) among all complete labellings of  $AF$ .  $\mathcal{L}$  is the **preferred** labelling iff  $\mathcal{L}$  is a complete labelling where  $in(\mathcal{L})$  is maximal (w.r.t. set inclusion) among all complete labellings of  $AF$ . A **stable** labelling is a complete labelling with  $undec(\mathcal{L}) = \emptyset$ .

The grounded semantics, first introduced by Pollock [5], is a skeptical semantics that can be computed in polynomial time by accepting initial arguments and then any argument defended directly or indirectly by initial arguments. In this first paper, grounded semantics is the only semantics used by an ADG.

### 3. Argumentative Decision Graphs

In this section we introduce the notion of argumentative decision graphs. We work with a dataset composed by a set of features, each of them taking values in a finite set. A feature represent the target variable of the classification task. Informally, an argumentative decision graph is an

extension of Dung’s abstract argumentation graph [1] where arguments have a rule-based structure with a premise (also called the support) and a conclusion. The premise of each argument consists of a feature and an associated value, while the conclusion is a value for the target variable. Some arguments might have an empty conclusion, meaning that their role is not to predict the target variable directly, but rather to interact with other predictive arguments.

Formally, we consider a dataset  $\mathcal{D}$  represented by a  $N \times M$  matrix-like data structure, where each row is called an instance of the dataset and each column is called a feature. An instance can be represented by a tuple  $t = \langle v_1, \dots, v_m \rangle$ , where  $v_i$  is the value associated with the  $i^{\text{th}}$  feature  $f_i$ . We consider the predicates of the form  $f_i(v)$ , with the meaning “the feature  $i$  has the value  $v$ ”.

Given a tuple  $t = \langle v_1, \dots, v_i, \dots, v_m \rangle$ , the predicate  $f_i(v)$  is *verified* by  $t$  iff  $v_i = v$  (the value of the  $i^{\text{th}}$  component of the tuple  $t$  is equal to  $v$ ), and *not verified* otherwise. One of the features  $f_i$  is called the target variable and it is denoted with  $y$ , and therefore the predicates involving the target variable have the form  $y(v)$ . The set of all the predicates is called  $\mathcal{P}_f \cup \mathcal{P}_y$ , where  $\mathcal{P}_y$  is the set of predicates regarding the target variable  $y$  and  $\mathcal{P}_f$  the set of predicates regarding the other features  $\{f_1 \dots f_m\}$ .

An ADG is an argumentation framework where each argument  $a$  has a structure  $a = \langle \phi, \theta \rangle$ , where  $\phi \in \mathcal{P}_f$  and  $\theta$  is either the empty set or a target variable predicate  $y(v)$ . An argument with a non-empty conclusion is called a *predictive* argument. An attack relation is defined over the arguments. The definition is therefore the following:

**Definition 3.1.** An argumentative decision graph ADG is an argumentation framework  $AF = (Ar, \mathcal{R})$  where each  $a \in Ar$  has the form  $a = \langle \phi, \theta \rangle$ ,  $\phi \in \mathcal{P}_f$ ,  $\theta \in \mathcal{P}_y \cup \emptyset$  and  $\mathcal{R} \subseteq Ar \times Ar$ .

**Example 1.** Let us consider the dataset  $\mathcal{D}_1$  in Table 1, describing Paul’s activities on Sunday. The dataset has the following four Boolean features:  $w$  (whether the weather is windy),  $s$  (whether the weather is sunny),  $k$  (whether Paul has a sore knee or not) and  $l$  (whether Paul has a fishing licence) and a target variable  $a$  (*activity*), that takes the two values  $\{surf, fish\}$ . Using the features of  $\mathcal{D}_1$ , the following two ADGs are given:

- $ADG_1 = \langle \{a_1, a_2, a_3\}, \{(a_1, a_2), (a_2, a_3), (a_3, a_2)\} \rangle$
- $ADG_2 = \langle \{a_1, a_2, a_3, a_4\}, \{(a_1, a_2), (a_2, a_3), (a_3, a_2)\} \rangle$

where  $a_1 = \langle k(yes), \emptyset \rangle$ ,  $a_2 = \langle w(yes), a(surf) \rangle$ ,  $a_3 = \langle s(yes), a(fish) \rangle$ ,  $a_4 = \langle w(no), a(fish) \rangle$ .

$ADG_1$  is saying that Paul goes surfing if the weather is windy (argument  $a_2$ ) and fishing if the weather is sunny ( $a_3$ ), but he cannot do both of the activities ( $a_2$  and  $a_3$  mutually attack each other), and he does not go surfing if he has a sore knee ( $a_1$  attacks  $a_2$ ).  $ADG_2$  is adding the information that Paul goes fishing if the weather is not windy ( $a_4$ ). We can represent an ADG with a directed graph where each node is labelled with the argument it represents. In figure 1  $ADG_1$  and  $ADG_2$  are shown. For readability, we also wrote the description of each argument.

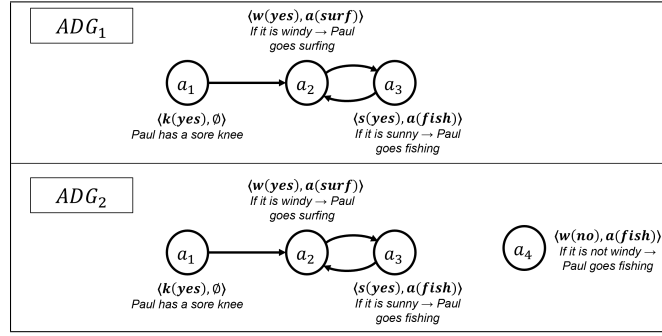
### 3.1. Well-formed ADGs

In order to be well-formed, the attack relation of an ADG has to satisfy some consistency constraints. The first constraint is that there is no attack between two arguments whose supports contain the

**Table 1**

The dataset  $\mathcal{D}$  consists of 10 tuples and four features. The values of the target variable *activity* predicted by  $ADG_1$  and  $ADG_2$  are also reported.

ID	Sunny (s)	Windy (w)	Sore Knee (k)	Activity (a)	$P_{ADG_1}$	$P_{ADG_2}$
1	Yes	Yes	Yes	Fish	Fish	Fish
2	Yes	Yes	No	Surf	und	und
3	Yes	No	No	Fish	Fish	Fish
4	Yes	No	No	Fish	Fish	Fish
5	No	Yes	Yes	Fish	unk	unk
6	No	No	Yes	Fish	unk	Fish
7	No	Yes	No	Surf	Surf	Surf
8	No	No	No	Fish	unk	Fish
9	No	Yes	Yes	Surf	unk	unk
10	No	Yes	No	Surf	Surf	Surf

**Figure 1:** Two ADGs.

same feature. This is because the two arguments are mutually exclusive, since only one of the two supports can be verified by an input tuple. The second constrain is that there is no attack between two arguments whose conclusions are the same and not empty. Third, if two arguments  $a$  and  $b$  have mutually exclusive conclusions and they use different features in their supports, there must be an attack between them: either  $a$  attacks  $b$ ,  $b$  attacks  $a$  or they symmetrically attack each other. This constrain is introduced to guarantee that the predictions of an  $ADG$  are conflict-free, i.e., two arguments with mutually exclusive conclusions cannot be accepted at the same time. Formally the three constrains are as follows. We remind how  $y$  denotes the target variable and  $f_i$  the  $i^{\text{th}}$  feature.

**Definition 3.2.** Given an  $ADG = \langle Ar, \mathcal{R} \rangle$ , the  $ADG$  is well-formed iff  $\forall a_1, a_2 \in Ar$  with  $a_1 = \langle \phi_1, \theta_1 \rangle, a_2 = \langle \phi_2, \theta_2 \rangle$  it holds that:

1. if  $\phi_1 = f_i(v_1)$  and  $\phi_2 = f_i(v_2)$  then  $(a_1, a_2) \notin \mathcal{R} \wedge (a_2, a_1) \notin \mathcal{R}$
2. if  $\theta_1 = \theta_2 \neq \emptyset$  then  $(a_1, a_2) \notin \mathcal{R} \wedge (a_2, a_1) \notin \mathcal{R}$
3. if  $a_1 = \langle f_1(v_x), y(v_1) \rangle, a_2 = \langle f_2(v_y), y(v_2) \rangle$  and  $v_1 \neq v_2 \wedge f_1 \neq f_2$  then  $(a_1, a_2) \in \mathcal{R} \vee (a_2, a_1) \in \mathcal{R}$ .

### 3.2. Making Predictions using an ADG

We consider a dataset  $\mathcal{D}$  and an ADG built from  $\mathcal{D}$ . The target variable is  $y$  and  $Y$  is the set of values that the target variable can take.

An ADG identifies a function  $P_{ADG} : \mathcal{D} \rightarrow Y \cup \{\text{und}, \text{unk}\}$  that, given an input tuple  $t \in \mathcal{D}$ , it returns either a value for the target variable or one of two additional outputs: undecided (und) or unknown (unk). The computation of  $P_{ADG}$  is described in algorithm 1. Given an input instance  $t$ , we first consider the set  $V$  of all the arguments whose support is verified by  $t$ . Then, we apply grounded semantics on the abstract argumentation framework restricted to the verified arguments. We remind how, since the ADG is well-formed, all the accepted arguments have the same conclusion. We then consider the following situations:

1. if there is at least a predictive accepted argument, the value predicted by the ADG is the value of the conclusion of any of the predictive accepted arguments.
2. if the extension is empty but there is at least one predictive argument that is undecided, then the predicted value of the ADG is the status undecided: there are arguments that could be used to predict the target variable, but they are part of conflicts that cannot be resolved.
3. in all the other cases, when there are neither accepted nor undecided predictive arguments, the value returned by the ADG is the status unknown.

---

**Algorithm 1:** The function  $P_{ADG}$  to make predictions given a tuple  $t \in \mathcal{D}$ . The dataset  $\mathcal{D}$  is described by the features  $f_1, \dots, f_m$  and target variable  $y$  taking values in the set  $Y$ .

---

```

1 Inputs:  $ADG = \langle Ar, \mathcal{R} \rangle$ , a tuple  $t \in \mathcal{D}$ ;
2 Output: a predicted value in the set  $Y \cup \{\text{und}, \text{unk}\}$ 
3  $V \leftarrow \text{verified}(Ar, t)$ 
4  $\mathcal{L} \leftarrow \text{Grounded}(ADG_{|V})$ 
5 if  $\text{in}(\mathcal{L}) \neq \emptyset$  then
6   | if  $\exists a \in \text{in}(\mathcal{L}) \mid a = \langle \phi, y(v) \rangle$  then
7   |   | return  $v$ 
8   | else
9   |   | if  $\text{und}(\mathcal{L}) \neq \emptyset \wedge \exists a \in \text{und}(\mathcal{L}) \mid a = \langle \phi, y(v) \rangle$  then return und;
10 return unk

```

---

**Example 2.** Let us consider again  $ADG_1$  and  $ADG_2$  and the dataset  $\mathcal{D}_1$  in Table 1. Let us consider the first tuple  $t_1 = \langle \text{yes}, \text{yes}, \text{yes}, \text{fish} \rangle$  and compute  $P_{ADG_1}(t_1)$ . All the three arguments  $a_1, a_2, a_3$  are verified by  $t_1$ , and therefore the argumentation graph to be considered is equivalent to the full one. The grounded semantics returns two accepted arguments  $\{a_1, a_3\}$  since  $a_2$  is defeated by  $a_1$ . Since  $a_3 = \langle s(\text{yes}), a(\text{fish}) \rangle$  then  $P_{ADG_1}(t_1) = \text{fish}$ . If we consider the tuple  $t_2 = \langle \text{yes}, \text{yes}, \text{no}, \text{surf} \rangle$  (the weather is sunny and windy and Paul has no sore knee), only  $a_2$  and  $a_3$  are verified by  $t_2$ , and the resulting argumentation framework is a couple of mutually attacking arguments labelled undec by the grounded semantics and therefore  $P_{ADG_1}(t_2) = \text{und}$ . Regarding tuple  $t_6 = \langle \text{no}, \text{no}, \text{yes}, \text{surf} \rangle$  (weather neither sunny nor windy, Paul has a sore knee), only argument  $a_1$  is verified and accepted by grounded semantics. However, this argument does not predict the target variable and, since there are no undecided arguments, nothing can be said about the target variable and  $P_{ADG_1}(t_6) = \text{unk}$ .

By comparing the predictions of an *ADG* with the actual values of the target variable, the usual performance metrics derived from the confusion matrix such as accuracy, precision, recall and f1-score can be computed. The confusion matrix is a joint-frequency table of the predicted versus the actual values. In a binary classification the confusion matrix is a  $2 \times 2$  table. However, since an *ADG* can also return the two special values undecided and unknown, the confusion matrix will have two additional lines. In Figure 2 the confusion matrices for  $ADG_1$  and  $ADG_2$  computed using the dataset  $\mathcal{D}_1$  are shown.

$ADG_1$		Actual values		$ADG_2$		Actual values	
		Fish	Surf			Fish	Surf
Predicted values	Fish	3	0	Predicted Values	Fish	5	0
	Surf	0	2		Surf	0	2
	Und	0	1		Und	0	1
	Unk	3	1		Unk	1	1
Accuracy		0.5		Accuracy		0.7	
Decision Accuracy		1		Decision Accuracy		1	
Precision		1	1	Precision		1	1
Recall		0.43	0.66	Recall		0.71	0.66
F1-score		0.6	0.8	F1-score		0.83	0.8
Unknown		0.4		Unknown		0.2	
Undecided		0.1		Undecided		0.1	

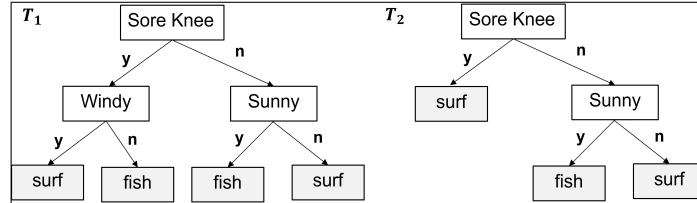
**Figure 2:** Confusion matrices for  $ADG_1$  and  $ADG_2$  for the dataset  $\mathcal{D}_1$  of Table 1.

We note how both  $ADG_1$  and  $ADG_2$  have a perfect accuracy when a prediction is generated (this is measured by the *decision accuracy* equal to 1), however  $ADG_1$  is in the unknown status in 4 out of 10 cases and undecided in one case, and therefore its overall accuracy is 0.5.  $ADG_2$  is a better predictive model since it makes predictions in more cases than  $ADG_1$  without adding any prediction error. This is reflected by its higher f1-score.

#### 4. A critical comparison with Decision Trees

In this section we outline some interesting differences and similarities between *ADGs* and decision trees. We consider again a dataset  $\mathcal{D}$  with a set of features  $\{f_1, \dots, f_n, y\}$  where  $y$  is the target variable. A decision tree (*DT*) [3] is a tree-structure where each node is associated with a test on a feature (also referred as a variable in *DT* terminology), and each of the terminal nodes of the tree has a value of the target variable associated. Given an input tuple, a decision tree returns a predicted value for the target variable (the *decision*). In order to take a decision, the tree has to be visited starting from the root node. At each node the test associated with that node is evaluated and the result of the test determines which child node has to be visited next, until a terminal node is reached and a prediction is made. Figure 3 shows an example of two decision trees for the dataset  $\mathcal{D}_1$  in Table 1.  $T_1$  has three internal nodes and four terminal nodes.  $T_2$  has two internal and three terminal nodes, and it does not use all the features (for instance *windy* weather is not used). Decision trees can be learnt from data and many efficient algorithms have been proposed, such as ID3, C4.5 and CART

[6]. The fundamental idea is similar, a tree is learnt in a top-down, recursive, divide-and-conquer approach. Starting from the root node, the algorithm selects the best variable to split the tree in two or more sub-trees. The best variable is the one that splits the tree in branches such that the target variable becomes easier to predict in each branch. Different measures for the concept of *easier to predict* are used by decision tree algorithms, such as the Gini index, the information gain (defined as the change in the entropy of the target variable conditioned to knowing the value of the variable tested for the split) and gain ratio. We now discuss some key differences and similarities between *ADGs* and decision trees (*DT*).



**Figure 3:** Two decision trees for the dataset  $\mathcal{D}$  of Table 1.

**Entry point.** A decision tree has a single entry point - the root of the tree - and a specific order in which nodes are tested to generate a prediction. On the contrary, an *ADG* has no entry point and arguments can be evaluated in any order. The consequence is that, in case of partial information - that is the value of one or more features is missing or unknown - an *ADG* has a higher chance to return a prediction. For instance, if the information about the feature *sore knee* is not available  $T_1$  and  $T_2$  cannot return a prediction, while  $ADG_1$  and  $ADG_2$  could.

**Adding new knowledge.** Let us suppose that new knowledge needs to be added to the dataset in the form of a new feature/variable. An argumentation graph is easier to be expanded; in order to add a new argument, it is enough to define the new set of attacks involving the new node. The previous structure of the graph is unchanged. On the other side, adding a variable to a decision tree without changing its previous structure can be done only by adding new terminal nodes testing the new variable or use the variable as the new root. Retraining the tree would in general generates a very different tree where the new variable might appear in different part of the graph. It can be observed that also for an *ADG* in order to find the optimal graph it might be necessary to modify the previous nodes and links. However, an *ADG* has more options when it comes to accommodate a new variable without changing the existing graph due to a less constrained topology.

**Representation of knowledge.** The topology of a *DT* and the one of an *ADG* are certainly different. However, despite the differences, both of the two models represent knowledge as a set of logical rules in disjunctive normal form. A tree is an acyclic graph where every path is disjoint from the other. Every path represents a distinct rule to reach a conclusion, and once a path is taken it is not possible to move back to another path. Multiple paths are usually present to reach the same conclusion. Therefore a *DT* represents knowledge as a set of mutually exclusive rules and each split represents a specialization of the support of a rule. On the contrary an *ADG* represents knowledge as a set of default rules, potentially defining an inconsistent (conflicting) set and subjects to exceptions. Every unidirectional attack provides an exception to a rule (=an argument) and



therefore specializing further when the rule can be used. Depending on the context and the data, the conflict-based representation of the *ADG* could generate more understandable and compact models than the mutually exclusive rules generated by the *DT* or vice versa.

**Explanations.** Decision Trees are regarded as one of the most explainable data mining models. The tree structure, if not too complex, can be easily understood by a human. Each branch of the tree represents a specific input case, and each branch identifies a logical rule consisting of the conjunction of all the tests used to reach the terminal node from the root. For instance, *sore knee* and *not windy weather* is an explanation for Paul going fishing in the tree  $T_2$ . The length of an explanation is the length of the path from the root node to the terminal node. The average length of all the paths - often weighted by the number of tuples covered by each path - is a measure of the average length of the explanations generated by the decision tree.

Regarding an *ADG*, the structure can be also understood by a human if the argumentation graph has a small size, but it could be argued that the free topology of an argumentation graph makes it harder to be understood. However, even when the graph is large, the application of the chosen semantics reduces the number of arguments that are necessary to explain a decision. In general, the arguments defeated by an accepted argument are irrelevant to define the decision and, if we are using *grounded semantics* and the set of accepted arguments is not empty, the undecided arguments are also irrelevant. Moreover, we do not need all the accepted arguments since the acceptability of some arguments might depend on the acceptability of other arguments. The explanations could therefore be compact also for a large graph. The analysis of explanations is an important issue that will be covered in future works.

**Variable Importance.** In a decision tree, the importance of a variable is measured by the information gain, that is the reduction in the conditional entropy of the target variable. In other words, a variable is important if, by knowing its value, the target variable can be predicted with more certainty. In an *ADG*, various measures of the importance of a variable could be defined. For instance, the importance of an argument could be proportional to the number of input tuples for which the argument is necessary to obtain a prediction. Note how, since an argument refers to a specific feature-value pair, an *ADG* has a more fine-grained notion of variable importance. An aggregated value for each feature can then be provided.

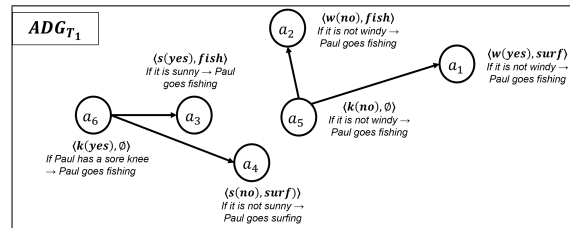
**Semantics.** Another difference between *DT* and *ADG* is the way their outputs are produced. An *ADG* is evaluated using argumentation semantics to solve the conflicts encoded in the graph. Given an input tuple, an *ADG* is evaluated by mean of an argumentation semantics that labels the arguments as accepted, rejected or undecided. The undecided label provides an *ADG* with a *built-in* way of quantifying the uncertainty deriving from conflicting information. Moreover, semantics provide a way of representing different strategies to accept arguments. While the *grounded semantics* is the most skeptical semantics maximizing the set of undecided arguments, semantics such as the *preferred* are credulous semantics maximizing the set of accepted arguments. Semantics other than the *grounded* are often multi-status, meaning that they generate multiple sets of accepted arguments, all consistent with the semantics used. Multi-status semantics could be used to present multiple consistent scenarios to the decision makers, each of them with valid reasons to be accepted.

**Expressiveness.** Regarding the expressiveness of the two models, they are equivalent. Given

an *ADG*, it is possible to define a *DT* computing exactly the same function. Indeed, if we compute the output of an *ADG* for all the input tuples, we can build an exhaustive *DT* where each path represents an input tuple and the value predicted by the terminal nodes is the same as the *ADG* output. Given a *DT*, an equivalent *ADG* can be built by following these rules:

1. for each directed link  $l$  from a node  $N$  to a terminal node  $M$  of the *DT*, create a predictive argument  $\langle f_i(v_n), y_m \rangle$ , where  $f_i$  is the variable tested at node  $N$  and  $v_n$  is the value of  $f_i$  identifying the link  $l$  connecting  $N$  to the terminal node  $M$ , and  $y_m$  is the value of the target variable predicted at node  $M$ .
2. for each link  $l$  from a node  $N$  to node  $M$  where  $M$  is non-terminal, create an argument  $\langle f_i(v_n), \emptyset \rangle$ , where  $f_i$  is the variable tested at  $N$  and  $v_n$  is the value of the feature identifying the link  $l$ .
3. after having found the arguments of the *ADG* using rules 1 and 2, for each argument  $a$  associated to a link  $l_a$  from node  $N$  to  $M$ , add to the attack relation  $\mathcal{R}$  the pair of nodes  $(a, b)$ , where  $b$  is any argument associated to a link  $l_b$  that is on a directed path  $p$  connecting  $N$  to a terminal node so that  $l_a \notin p$ , unless the two arguments use the same feature or predict the same value for the target variable.

The idea is that terminal nodes provide the arguments to predict the target variable, while the non-terminal nodes provide the non-predictive arguments. Regarding the attack relation, an argument generated by a link  $l$  from node  $N$  to  $M$  attacks all the arguments generated by links that are mutually exclusive with  $l$  and belonging to the sub-tree identified by the node  $N$ .



**Figure 4:** The *ADG* equivalent to  $T_1$ .

As an example, let us consider the decision tree  $T_1$ . We build the corresponding *ADG*. According to rule 1, the *ADG* has the following predictive arguments:  $a_1 = \langle w(\text{yes}), \text{surf} \rangle$ ,  $a_2 = \langle w(\text{no}), \text{fish} \rangle$ ,  $a_3 = \langle s(\text{yes}), \text{fish} \rangle$ ,  $a_4 = \langle s(\text{no}), \text{surf} \rangle$ . According to rule 2, it has the following two non-predictive arguments:  $a_5 = \langle k(\text{no}), \emptyset \rangle$ ,  $a_6 = \langle k(\text{yes}), \emptyset \rangle$ . Regarding the attack relation, according to rule 3 the only attacks are:  $a_5$  attacks  $a_3$  and  $a_4$ , while  $a_6$  attacks  $a_1$   $a_2$ . Argument  $a_5$  does not attack  $a_6$  and vice versa since they use the same feature and they are therefore mutually exclusive. The resulting *ADG* is shown in figure 4. We conclude by observing that, despite the two models are equivalent, this does not guarantee that the two algorithms will learn similar models from data. In fact, the way the relationship between input tuples and target variable is learnt and the way knowledge is represented imply that the two models will in general be quite different.

## 5. An algorithm to learn ADGs from data

In this section we present a first simple algorithm to learn *ADG* from data. The algorithm is a preliminary attempt, and it can be improved in multiple ways. However, it represents a starting point to study and test the potential of *ADGs*.

---

**Algorithm 2:** The function `BuildADG` to learn an *ADG* from a dataset  $\mathcal{D}$ . The dataset  $\mathcal{D}$  is described by the features  $f_1, \dots, f_n$  and target variable  $y$ . The target variable is binary and takes the values  $y^+, y^-$ .  $F_i$  represents the set of all possible values for the feature  $f_i$ . The function `eval` returns the selected performance indicator for a given *ADG*

---

```

1 Inputs: Dataset  $\mathcal{D}$ , a performance threshold  $\Delta$ 
2 Outputs: an ADG
3 Function BuildADG( $\mathcal{D}, \Delta$ ):
4   perf  $\leftarrow 0$  ; Arg  $\leftarrow \emptyset$ 
5   for each distinct pair ( $f_i, v_n$ ) do
6     Arg  $\leftarrow$  Arg  $\cup \langle f_i(v_N), y^+ \rangle \cup \langle f_i(v_N), y^- \rangle \cup \langle f_i(v_N), \emptyset \rangle$ 
7   ADGBest  $\leftarrow \langle \emptyset, \emptyset \rangle$ 
8   while Arg  $\neq \emptyset$  do
9     ADGBest  $\leftarrow$  ADGold
10    for  $a \in$  Arg do
11      ADGnew  $\leftarrow$  AddArg( $a, ADG_{Best}$ )
12      perfnew  $\leftarrow$  eval(ADGnew)
13      if perfnew  $>$  perf +  $\Delta$  then
14        perf  $\leftarrow$  perfnew
15        ADGbest  $\leftarrow$  ADGnew
16        abest  $\leftarrow$  a
17      if ADGBest == ADGold then
18        return ADGbest
19      Arg.remove( $a$ )
20    return ADGbest
21
22 Function AddARG( $a = \langle f_a(v_a), y_a \rangle, ADG = \langle Arg, \mathcal{R} \rangle$ ):
23   for  $b = \langle f_b(v_b), y_b \rangle \in$  Arg do
24     ADG←  $\leftarrow \langle Arg \cup \{a, b\}, \mathcal{R} \cup \{(a, b)\} \rangle$ 
25     ADG→  $\leftarrow \langle Arg \cup b, \mathcal{R} \cup \{(b, a)\} \rangle$ 
26     ADG↔  $\leftarrow \langle Arg \cup b, \mathcal{R} \cup \{(a, b), (b, a)\} \rangle$ 
27     if  $f_a \neq f_b \wedge y_a \neq y_b \neq \emptyset$  then
28       ADG  $\leftarrow X \in \{ADG_{\leftarrow}, ADG_{\rightarrow}, ADG_{\leftrightarrow}\}$  where eval( $X$ ) is maximal
29     else if  $f_a \neq f_b$  then
30       ADG  $\leftarrow X \in \{ADG, ADG_{\leftarrow}, ADG_{\rightarrow}, ADG_{\leftrightarrow}\}$  where eval( $X$ ) is maximal
31   return ADG

```

---

The algorithm, called `BuildADG`, builds an *ADG* incrementally by adding an argument at the time and by expanding the attack relation in order to maximize a performance indicator, such as the overall accuracy of the *ADG*. `BuildADG` is shown in algorithm 2. We consider a dataset  $\mathcal{D}$  with features  $\{f_1, \dots, f_n, y\}$  where  $y$  is the target variable, each feature  $f_i$  takes value from its corresponding set of values  $F_i$ , while the target variable is binary and it takes the two values  $y^+$  or  $y^-$ . The algorithm has two inputs: the dataset  $\mathcal{D}$  and a tuning parameter  $\Delta$ . The algorithm

starts by identifying the arguments that will be used to build the *ADG*. For each pair feature-value  $\langle f_i, v \rangle$  three arguments are added, one predicting  $y^+$ , one  $y^-$  and the neutral argument with empty conclusion  $\langle f_i(v), \emptyset \rangle$ .

Then, for each argument in *Arg* the algorithm adds to the *ADG* the argument  $a$  that increased the performance of the resulting *ADG* by the highest interval, but only if the addition of  $a$  increased the performance by at least  $\Delta$  compared to the previous *ADG*. Argument  $a$  is removed from the list of arguments and the procedure is repeated until all the arguments have been tested or it is not possible to increase the performance of the *ADG* by  $\Delta$ . The function `AddArg` is responsible for adding a new argument  $a$  to the *ADG*. For each argument  $b$  already in the *ADG*, we need to decide how the new argument  $a$  interacts with  $b$ . The resulting *ADG* must be well-formed to avoid logical inconsistencies. If argument  $a$  and  $b$  are mutually exclusive there is no need to add an attack link between them. The same is for the situation in which  $a$  and  $b$  are predicting the same value for the target variable  $y$ . If both  $a$  and  $b$  are predictive but they predict different values for  $y$  and they do not have mutually exclusive supports, an attack must be present to keep conflict-freeness. There are three possibilities:  $a$  attacks  $b$ ,  $b$  attacks  $a$  or they mutually attack each other. One of these three attacks must be present to keep the *ADG* well-formed, and the one generating the best *ADG* is kept. If either  $a$  or  $b$  are non-predictive argument, there is also the fourth possibility that there is no attack relation between the arguments (line 30).

## 6. Evaluation

In this section, we provide a first evaluation of the `BuildADG` algorithm and we compare its results to a C4.5 Decision Tree. The evaluation compares the total accuracy of the two models using three benchmark datasets. An evaluation considering other performance indicators or the explainability of the models is left for future works. The datasets used for the evaluation are well-known, publicly available datasets [7] for binary classification where all the features are categorical. We used  $\Delta = 0$  as parameters, meaning that any improvement to the *ADG* is retained. Table 2 shows the results for the three datasets. Performance was computed by randomly dividing each dataset into a training and testing set using an 80%-20% split ratio. We report the accuracy, balanced accuracy and the size of both the decision tree and the *ADG*, measured by the number of nodes and links of the graph. Table 3 shows the most important features considered by the decision tree and by the *ADG*. Information gain is used to rank features in the decision tree, while the proportion of times that an argument is necessary to make a prediction is used to rank features in the *ADG*.

**Table 2**

Results obtained by the `BuildADG` and the C.45 algorithm. The column *size* reports the number of nodes and links.

Dataset	Records	Features	Decision Tree				ADG		
			Acc	CI 95%	B. Acc	size	Acc	B.Acc	size
Bank	600	11	0.82	[0.74,0.89]	0.82	28/27	0.75	0.75	6/7
US Census	32561	14	0.84	[0.83,0.85]	0.73	14/13	0.83	0.75	21/78
Car Price	1782	6	0.94	[0.91,0.96]	0.92	20/19	0.88	0.88	8/11

In both the *bank* and *car price* datasets, the accuracy of the decision tree was statistically higher than the *ADG* accuracy. The gap was smaller for the *car price* dataset, where *ADG* registered a balanced accuracy of 0.88. For the *US census* dataset we obtained positive results: the accuracy of the *ADG* was in the 95% confidence interval of the decision tree accuracy and only marginally lower (83.4% versus 84.2%), and the balanced accuracy of the *ADG* was higher than the one of the decision tree. Regarding the size of the models learnt, we used the number of nodes and links to measure it (note how the number of links in a tree is  $N - 1$ , where  $N$  is the number of nodes). The *ADG* graph was smaller for the *car price* and *bank* dataset, but more complex for the *US census* dataset. Table 3 shows the most important variables (ranked by importance) for decision tree and for *ADG*. There is a good degree of overlapping for all the three datasets, meaning that the two algorithms substantially agreed on the most important variables.

*Discussion.* The results obtained are promising but they also expose some of the weakness of the preliminary algorithm proposed. Indeed, the algorithm is naive in several aspects. For instance, the addition of a new argument is accepted if the absolute number of correctly classified instances is increased by an interval  $\Delta$  even if the accuracy (i.e. the percentage of correct predictions) could decrease, showing how the algorithm has a bias in favour of increasing the coverage of the instances rather than maximizing the accuracy. The algorithm also needs a pruning procedure, similar to the ones used in a decision tree. The result of the *US census* dataset shows that an *ADG* can become very complex, harming the understandability of the outputs but also increasing the chance of model overfitting. A pruning strategy should reduce the complexity of the model, keeping its accuracy high. One idea could be to introduce a regularization parameter similar to the one present in the cost-based pruning mechanism of a decision tree, parameter that will penalize complex *ADGs*, forcing the algorithm to find a trade-off between complexity and performance. Finally, our preliminary algorithm is not computationally efficient, and it can be optimized in multiple ways, including an approximation using Monte Carlo simulation.

**Table 3**

Most important variables for decision tree and *ADG*

Dataset	Decision Tree	<i>ADG</i>	$n$
Bank	age, children, income	children, income, married	2/3
US Census	relation, marital status, cap gain, degree, ed num, gender, occupation	cap gain, degree, ed num, hours, relations, age, gender	5/7
Car Price	persons, safety, maintenance, lug boot	persons, safety, price, maintenance	3/4

## 7. Related Works

Recently, there has been an increasing number of studies mixing argumentation theory and machine learning methods. However, very little studies directly face the problem of learning an argumentation graph from data. The large majority of applications are in the field of argumentation mining, where arguments are extracted automatically from text using NLP techniques either in a data-driven

fashion or in a mixed approach where an explicit structure of arguments has to be matched on the text [8],[9],[10]. In both cases, learning an argumentation graph from data is not part of the problem. In [11] the authors learnt the strength of a probabilistic argumentation frameworks using the Bayesian inference rule, while in [12] the authors proposed an algorithm to learn a probabilistic argumentation graph given a set of extensions. A different approach is the one where machine learning techniques have been adopted to predict the acceptability of arguments under a given semantics. In [13] this problem was tackled by modelling it as a multinomial classification task and by using convolutional neural networks to learn the acceptability of arguments. The work by Craandijk and Bex [14] had a similar aim. The authors proposed to use special neural networks, called argumentation graph neural network (AGNN), to learn a binary classification model predicting whether an argument is accepted or rejected. The difference from our approach is that in those approaches an argumentation framework already exists, and the problem is to learn the output of a semantics applied to the argumentation framework. On the contrary, our problem is to learn such argumentative framework from data. Moreover, our aim is to learn a Dung-like argumentation framework that could generate understandable justifications, while the aim of both [13] and [14] is not the intelligibility of the model, but rather to train a black-box deep neural network to compute a semantics accurately. In [15] the authors proposed to use genetic algorithms to learn a gradual argumentation graph, considered as an instance of a sparse multi-layer neural network. To obtain a well-interpretable model, the authors proposed to use a fitness function balancing sparseness and accuracy of the classifier. The paper presents experimental results on standard benchmark datasets from the UCI machine learning repository [7]. The results obtained showed an accuracy comparable to decision trees across the three datasets evaluated. In [16] the authors proposed a method to equip autonomous agents with the ability to argue and explain their decisions. Similar to our approach, arguments and attack relations between arguments are built from a set of training examples. The generation of arguments is also based on all the pairs feature-values found in the training dataset. Two types of attacks are defined, symmetrical rebuttal attacks and unidirectional undercutting attacks. Differently from our work, these attacks are explicitly identified by the structure of the arguments rather than being decided based on how well they fit the dataset given.

## 8. Conclusions

In this paper, we presented a novel data-mining algorithm called argumentative decision graphs (*ADG*). An *ADG* is a special argumentation framework where arguments have a rule-based structure and an attack relation is defined among arguments. *ADGs* are learnt from data in a supervised way and they can be used for classification tasks. We have discussed the main differences and similarities with similar models such as decision trees, showing a translation between the two formalisms. Unlike decision trees, the output of an *ADG* can be also an undecided status, where the graph does not have enough reasons to predict a value for the target variable. This is due to the use of argumentation semantics to identify the arguments of an *ADG* that are accepted and consequently make a prediction on the target variable. We evaluated a preliminary greedy algorithm to learn an *ADG* from data using benchmark datasets and we compared our results with the C4.5 decision tree algorithm. Our results showed how *ADG* had an accuracy lower or comparable to decision trees, a

generally less complex model and a good agreement on the importance of the variables between the two models. The algorithm presented is naive in some of its assumptions, and it can be improved in many aspects, including how arguments interact, the way the impact of an argument is evaluated and how to reduce its computational time. Overall, we believe to have provided enough evidence to justify further research into *ADGs*.

## References

- [1] P. M. Dung, On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games, *Artificial intelligence* 77 (1995) 321–357.
- [2] M. W. Caminada, D. M. Gabbay, A logical account of formal argumentation, *Studia Logica* 93 (2009) 109–145.
- [3] J. R. Quinlan, Induction of decision trees, *Machine learning* 1 (1986) 81–106.
- [4] F. V. Jensen, T. D. Nielsen, *Bayesian networks and decision graphs*, volume 2, Springer, 2007.
- [5] J. L. Pollock, *Cognitive carpentry, a blueprint for how to build a person*, Mit Press, 1995.
- [6] B. Hssina, A. Merbouha, H. Ezzikouri, M. Erritali, A comparative study of decision tree id3 and c4. 5, *International Journal of Advanced Computer Science and Applications* 4 (2014) 13–19.
- [7] D. Dua, C. Graff, UCI machine learning repository, 2017. URL: <http://archive.ics.uci.edu/ml>.
- [8] O. Cocarascu, A. Stylianou, K. Čyras, F. Toni, Data-empowered argumentation for dialectically explainable predictions, in: *ECAI 2020*, IOS Press, 2020, pp. 2449–2456.
- [9] O. Cocarascu, F. Toni, Detecting deceptive reviews using argumentation, in: *Proceedings of the 1st International Workshop on AI for Privacy and Security*, 2016, pp. 1–8.
- [10] M. Lippi, P. Torroni, Argument mining: A machine learning perspective, in: *International Workshop on Theory and Applications of Formal Argumentation*, Springer, 2015, pp. 163–176.
- [11] K. Noor, A. Hunter, A bayesian probabilistic argumentation framework for learning from online reviews, in: *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, IEEE, 2020, pp. 742–747.
- [12] R. Riveret, G. Governatori, On learning attacks in probabilistic abstract argumentation, in: *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, 2016, pp. 653–661.
- [13] I. Kuhlmann, M. Thimm, Using graph convolutional networks for approximate reasoning with abstract argumentation frameworks: A feasibility study, in: *International Conference on Scalable Uncertainty Management*, Springer, 2019, pp. 24–37.
- [14] D. Craandijk, F. Bex, Deep learning for abstract argumentation semantics, *arXiv preprint arXiv:2007.07629* (2020).
- [15] J. Spieler, N. Potyka, S. Staab, Learning gradual argumentation frameworks using genetic algorithms, *arXiv preprint arXiv:2106.13585* (2021).
- [16] L. Amgoud, M. Serrurier, Agents that argue and explain classifications, *Autonomous Agents and Multi-Agent Systems* 16 (2008) 187–209.