Explainable Machine Learning for Liver Transplantation

Pedro Cabalar[a], Brais Muñiz[a,b], Gilberto Pérez[a,b], Francisco Suárez[c,d]

[a] *University of Corunna, SPAIN*
{cabalar,brais.mcastro,gilberto.pvega}@udc.es
[b] *CITIC Research Center*
University of Corunna, SPAIN
[c] *Digestive Service,*
Complexo Hospitalario Universitario de A Coruña (CHUAC)
francisco.suarez.lopez@sergas.es
[d] *Instituto de Investigación Biomédica de A Coruña (INIBIC)*
University of Corunna, SPAIN

Abstract

In this work, we present a flexible method for explaining predictions made by decision trees in human-readable terms and we apply it to obtain explanations for long-term (five years) survival predictions for liver transplant recipients. We present a decision tree that has been obtained through machine learning applied on a data set collected at the liver transplantation unit from the Coruña University Hospital Center (CHUAC) concerning transplant cases from 2009 to 2014. We provide an implementation of this method in the form of a publicly available `Python` tool we have called `Crystal-tree`. We also illustrate the method with some examples of the explanations we get from `Crystal-tree` on the obtained decision tree, including multiple adaptations for different languages and/or different levels of expertise. We show that the resulting explanations remain simple for dense, deep trees even when some features are used in a repetitive manner. Finally, we briefly explain the implementation of the tool which uses a logic programming encoding as a back-end.

Keywords: Explainable Artificial Intelligence; Answer Set Programming; Logic Programming; Machine Learning; Liver Transplantation

1. Introduction

When Artificial Intelligence (AI) techniques are applied in a sensitive domain such as Healthcare, the generation of accurate predictions or conclusions may become useless if the system is not additionally capable of providing good *explanations* for them[1]. As an example, consider the liver transplantation domain and take the problem of matching a donor with a recipient on a basis of some prediction for the recipient's survival. This matching decision will obviously have a critical impact on the recipient's life, but perhaps on some of the patients in the waiting list too, depending on their condition. Moreover, the final outcome could eventually carry out legal implications for the doctor or the hospital. A machine learning algorithm may show a high accuracy in its predictions but, if it does not provide explanations, it will easily cause distrust on the humans involved, such as raising the doctor's suspicion on a possible lack of medical criterion or the patient's doubts about lack of fairness due to a blind statistical method.

The generation of explanations for the outcome of ML algorithms is an active field of research nowadays. The main difficulty here comes from the fact that most ML methods act as black boxes, so that their predictions are the result of complex mathematical processes and are difficult to explain or reproduce by humans. One ML technique that does not suffer from this black box limitation is *decision tree* (DT) learning [1]. In DT, the result of the learning process is a binary tree where each non-leaf node checks some condition on an input feature, like for instance '`rec_afp` ≤ 9' (recipient's alfafetoprotein lower or equal than 9) and then has two child subtrees to be followed depending on whether the condition is true or false (see, for instance, Figure 1 in page 11). A prediction can then be easily explained in human terms by just following the corresponding path in the tree. Although DTs are usually outperformed by other ML algorithms in terms of prediction accuracy, their simplicity and readability makes them a valuable choice when explanations are required, as happens in healthcare. In fact, in the recent survey [2] of articles that apply AI to organ transplantation, more than a half of the approaches include a DT-based learning algorithm.

In the case of liver transplantation, studies [3] and, more recently, [4] used Neural Networks for predicting survival of recipients before and after the surgery (respectively), but they do not make any special emphasis in the interpretability of the obtained models. On the other hand, Bertsimas et al. [5]

[1]The right to obtain an explanation for an automated decision is, in fact, one of the main concerns behind the European *General Data Protection Regulation* (GDPR). See https://gdpr-info.eu/

trained two DTs on a large dataset with 1,618,966 observations to predict 3-month removal from the waiting list (including a possible fatal outcome) for a given patient. The data set was split into patients with and without hepatocellular carcinoma (HCC) and two respective DTs were obtained by applying ML techniques. The two DTs were implemented in an interactive online tool[2] that provides several graphical display facilities plus a simulator to make a prediction using 4 input variables from some patient's data. The tool does not provide a specific explanation for the simulated prediction but, instead, it allows graphically browsing the DT by hand, collapsing or expanding some parts of the tree. Building the explanation through this manual process becomes rather cumbersome, especially in the case of the non-HCC tree, whose excessive width and detail makes the graphical manipulation of the tree a difficult task prune to potential errors. This is, in fact, a common problem in many DTs obtained from large data sets: their accuracy is normally at odds with their simplicity. Good predictions are obtained at the cost of the generation of a dense DT which eventually lacks of the simplicity and ease of use that is expected from an explainable AI tool.

In this work, we present a flexible method for explaining any decision tree in friendly, human-readable terms, i.e. using natural language text. The method consists in first representing the DT as a logic program and then using the logic programming explanation tool `xclingo` [6]. This tool accepts natural language annotations in the logic program and uses them to generate compound explanations for the program results. The explanations provided remain simple even for large or complex decision trees. We also provide an open-source, ready to use tool called `Crystal-tree` that transparently implements our approach. As a case study, we use a real domain and tackle the problem of predicting 5-year survival for donor-recipient pairs in liver transplantation. For this purpose, a liver transplant data set was collected at the Digestive Service of the *Coruña University Hospital Center* (CHUAC), Spain. We learnt a DT from the data set and then applied our method for providing clear, natural language explanations. Depending on the use of the language, different kinds of explanations can be easily provided from the same final DT model: for instance, on the one hand, we implemented technical explanations for the medical experts and, on the other hand, simplified ones for non-specialists such as patients, relatives, etc. The explanations are also

[2]http://www.opom.online/

provided in different languages (English, Spanish and Galician).

The rest of the paper is organized as follows. Section 2 describes the dataset and the ML methodology used for obtaining the decision tree model. Section 3 focuses on the `Crystal-tree` tool and how to use it. The next section explains and discusses the meaning of our explanations and how they are meant to be understood. Section 5 provides a brief description of the logic programming implementation. Finally, Section 6 discusses future work and concludes the paper.

2. Obtaining the Decision Tree that Predicts Recipient's Survival

We explain next the complete process we followed to obtain a DT that predicts the recipient's survival in a period of 5 years since transplantation. We start describing the data set, then proceed to explain the feature selection process to continue with the techniques we used to alleviate the unbalanced data set (most patients in CHUAC survive after 5 years) and, finally, the process to train the final DT.

2.1. Data set

The data used for the ML experiments were collected from the physical archives of the Digestive Service of the CHUAC Hospital in Spain. The final data set consisted of 258 transplants dating from 2009 to 2014. Each sample comprises 64 features both from the recipient and the donor. Tables 1 and 2 respectively describe the recipient's and the donor's complete sets of features. The data used for the study were manually collected by a team of five people, which included a medical expert directly related to the examined transplantation cases. Part of this information was obtained from the hospital software data base, but most of the data were retrieved from paper reports sometimes including unstructured, hand-written notes in patient clinical records. The data collection was carefully planned and, to accelerate the process, automated forms designed on purpose were used for entering the data on-site. Finally, the data was integrated into a structured relational database and the final data set used for the ML experiments was obtained from an SQL query.

2.2. Feature Selection

Before training any model, we proceeded to a feature selection based on statistical significance. To this aim, we applied a chi-square test for each can-

Table 1: Recipient features per each transplantation record. Summary column shows mean and standard deviation for numerical features and value ratios for categorical features.

Feature name	Summary
Sex	Male/Fem (0.78/0.22)
Age	55.81±9.77
Size	167.59±8.78
Weight	77.48±15.71
Medical condition	Home/Plant/ICU (0.86/0.08/0.05)
Cytomegalovirus	0/1 (0.22/0.78)
Urea	56.26±53.28
ALT	118.74±549.83
Albumin	3.63±4.38
Alpha-fetoprotein	42.01±160.15
International Normalized Ratio (INR)	1.54±1.05
Bilirubin	4.65±7.28
Creatinine	1.36±1.45
Model for End Stage Liver Disease (MELD)	17.37±7.13
Previous abdominal surgery	0/1 (0.86/0.14)
Life support therapy	0/1 (0.25/0.02)
Dyalisis	0/1 (0.95/0.05)
Recurrent encephalopathy	0/1 (0.56/0.44)
Portal Vein Thrombosis (before transplantation)	0/1 (0.88/0.12)
Ascites (before transplantation)	0/1 (0.43/0.57)
Portal Bleed (before transplantation)	0/1 (0.98/0.02)
Diabetes	0/1 (0.74/0.26)
Insuline	0/1 (0.21/0.05)
Coronary Angina	0/1 (0.97/0.03)
Hypertension	0/1 (0.79/0.21)
Transjugular Intrahepatic Portosystemic Shunt (TIPS)	0/1 (0.96/0.04)
Portal Vein Thrombosis (Yerdel Stage 3 or 4)	0/1 (0.03/0.71)
Hepatitis B virus (HBV)-related liver disease	0/1 (0.97/0.03)
Hepatitis C virus (HCV)-related liver disease	0/1 (0.74/0.26)
VIH positive	0/1 (0.97/0.03)
Acute on chronic Liver Failure	0/1 (0.24/0.03)
HCC	0/1 (0.35/0.64)
HCC meeting Milan	0/1 (0.32/0.48)
Coronary artery disease (CAD)	0/1 (0.25/0.02)
Past history of extrahepatic cancer	0/1 (0.02/0.97)
Diagnosis: Alcohol-related liver disease (ARLD)	0/1 (0.58/0.41)
Diagnosis: NASH	0/1 (0.69/0.03)
Diagnosis: Early Retransplant	0/1 (0.7/0.03
Diagnosis: Acute Liver Failure	0/1 (0.71/0.02)
Diagnosis: Cholestatic	0/1 (0.69/0.03)
Diagnosis: Late Retrasplant	0/1 (0.69/0.04)
Survial Days	950.83±858.79
Death after 5 years	0/1 (0.76/0.24)

Table 2: Donor features per each transplantation record. Summary column shows mean and standard deviation for numerical features and value ratios for categorical features.

Feature name	Summary
Age	58.7±15.04
Sex	Male/Fem (0.62/0.38)
Size	167.67±8.92
Weight	75.49±12.4
ICU stay	65.66±85.13
Ischaemic time	351.91±109.71
AST	49.36±63.65
ALT	50.85±59.0
GGT	86.12±139.47
FA	153.81±98.25
LDH	476.22±399.81
Total Bilirubin	0.82±1.29
HBV positive donor	0/1 (0.95/0.05)
Cytomegalovirus	0/1 (0.8/0.2)
Noradrenaline	0.26±0.4
Creatinine Final	1.02±1.04
Serum sodium	145.7±20.4
Macrovesicular steatosis	Low(0)/Mod(1)/Sev(2) (0.95/0.03/0.02)
Microvesicular steatosis	Low(0)/Mod(1)/Sev(2) (0.88/0.07/0.05)
Death Cause: Stroke	0/1 (0.36/0.64)
HCV antibody-positive grafts	0/1 (0.72/0.28)

didate feature against the target feature '*Death after 5 years*', a Boolean variable that points out whether the recipient survived after 5 years since transplantation or not. The chi-square test was computed using `chi2_contingency` function from the `scipy 1.5.0` python package. All numerical features were previously discretized using DTs for finding the best thresholds given the target, as described in [7]. Categorical features with more than two values were one-hot-encoded before the test, leading to a different p-value for each category. The lowest p-value was taken as representative of the whole categorical feature. As a result, we obtained a p-value for each feature in the dataset. The lowest the value, the highest the statistical significance for predicting the target feature. For the ML experiments, we selected the 7 most significant features shown in Table 3, again, in decreasing order of significance. Fewer features led to worse prediction results whereas more features led to excessively complex DTs without a relevant gain in accuracy. Five features correspond to the recipient (prefixed with '`rec_`') and only two of them to the donor (prefixed with '`don_`)'. As we can see, the most significant variable is `rec_hcv`, that is, whether the recipient was infected with Hepatitis C Virus (HCV) or not. The significance of this variable is explained because most records are previous to the generalized introduction of the successful HCV cure and so, HCV constituted a much higher potential risk of transplantation failure at that moment. The rest of features are self explanatory, except perhaps *medical condition* that tells us where did the patient come from. This feature can have one of three possible values: the patient's home, the hospital digestive ward or the intensive care unit (ICU). Regarding the donor's features, it is interesting to note that the liver microsteatosis was found to be the fourth most significant variable. As it can be observed, the second donor's attribute we use checks whether the donor's death cause was a stroke or not.

2.3. Balancing the data set

A data set is said to be *unbalanced* when the frequencies of the classes of the target feature are not balanced. As we can see in Table 1, this is in fact what happens with our target feature *Death after 5 years* where we face 76% samples for the negative class (recipient survived more than 5 years after the transplantation) versus 24% samples of the positive class. It is well known that ML algorithms, and especially DT algorithms, perform worse when trained against unbalanced data sets. In particular, we get the risk that the learning algorithm tends to predict the class with a higher frequency as

Table 3: Top 7 significant input features

From	Feature name	Short name	P-value
Recipient	HCV positive	rec_hcv	0.015
Recipient	Alpha-fetoprotein	rec_afp	0.042
Recipient	Previous abdominal Surgery	rec_abdominal_surgery	0.049
Donor	Microvesicular steatosis	don_microsteatosis	0.082
Recipient	Hypertension	rec_hypertension	0.111
Recipient	Medical condition	rec_medical_condition	0.138
Donor	Death cause: stroke	don_cva	0.146

a straightforward way to increase the accuracy results. For that reason, we applied a previous step of data set balancing before training any DT model.

When the unbalanced target variable is binary, the class with the higher (resp. lower) frequency receives the name of *majority* (resp. *minority*) class. Balancing methods try to balance majority and minority class ratios by modifying their proportions in the data set. This can be done either by removing samples from the majority class (*undersampling*) or by adding synthetic samples to the minority class (*oversampling*). In the case of *undersampling* methods, a common technique is computing clusters over the majority and then carefully select samples that will be disregarded, trying to keep the ratios within the original clusters in the resulting data set. Undersampling methods, therefore, consider a subset of the original data set where the minority class is untouched and, more importantly, all samples correspond to *real instances*. Oversampling methods, on the contrary, try to generate new minority class samples which are *similar* to the original ones. One widely used balancing method is *Synthetic Minority Oversampling Technique* (SMOTE) [8], which represents samples as points in a vector space and then generates new synthetic samples by finding points between two real samples. By doing this, it is more or less guaranteed that the synthetic samples will be realistic. However, depending on the domain, this may produce "impossible" samples. For instance, an oversampling method may generate a transplant case in which the recipient's hepatocellular carcinoma fits the Milan criteria, and yet the she does not have any hepatocellular carcinoma at all. This is because sample generation methods do not take into account any constraint already known beforehand. Given our current context, in which we are interested not only in obtaining good predictions but also in providing explanations, we have decided to stick to undersampling methods so that the DT is always generated from real samples and not synthetic ones.

We have tested several undersampling methods to search for the best choice before training the final DT model. With that purpose, a stratified train-test splitting was performed on the entire data set, preserving the unbalanced positive-negative ratio in the target class. First, a preliminary DT model was trained and tested on the unbalanced data. We used the accuracy, F1 score and kappa results of this preliminary tree on the test set as a baseline for evaluating the different balancing methods. Then, each method was applied on the training data, which was then used for training a DT. Each trained DT was finally tested against the initial unbalanced test set and compared to the baseline and the rest of the trained DTs. All the balancing methods were exclusively applied to the data set using the open-source `python` library `imbalanaced-learning` (version 0.8.0). The best method overall was *Nearmiss (version 1)* [9], which improved the accuracy from 0.78 to 0.87, the F1 measure [10] from 0.42 to 0.71 and the kappa score [11, 12] from 0.31 to 0.62 with respect to the baseline results in the initial unbalanced test set. Complete results are shown in Table 4.

Table 4: Evaluation balancing techniques

Undersampling Method	Accuracy	F1 score	kappa
Unbalanced Data (baseline)	0.78	0.42	0.31
Cluster Centroids	0.84	0.66	0.56
AllKNN	0.81	0.69	0.57
Random Undersampling	0.81	0.67	0.56
Nearmiss (version 1)	**0.87**	**0.71**	**0.62**
Nearmiss (version 3)	0.81	0.58	0.47
Instance Hardness Threshold	0.74	0.55	0.37
Neighbourhood Cleaning Rule	0.78	0.60	0.45
Onesided Selection	0.81	0.53	0.42
TomekLinks	0.82	0.53	0.42
Condensed Nearest Neighbour	0.71	0.42	0.22
Edited Nearest Neightbour	0.76	0.64	0.48
Repeated Edited Nearest Neigbours	0.68	0.57	0.37

2.4. Obtaining the Final Decision Tree

Once we selected the 7 most significant features and the best obtained balancing method, we proceeded to train a final DT. Categorical features were previously label-encoded, that is, their possible symbolic values were transformed into a discrete numerical range. For instance, the different possible Donor's Microvesicular steatosis values `Low` (below 30%), `Mod` (between 30% and 60%) and `Sev` (over 60%), were respectively encoded as 0, 1 and 2. This encoding was required because the condition in a DT node checks whether a feature x is smaller or greater than some threshold numerical value v. A train-test split was performed over the entire data set, using 80% for the training set and 20% for the test set. As explained before, we used the *Nearmiss (version 1)* method to balance the training data. The best parameters for training the decision tree were estimated by performing a 5-fold cross validation grid search over the training set. Table 5 shows the best parameters eventually obtained from this grid search. The previously described ML pipeline was performed using `scikit-learn 0.24.2`.

The final result of this learning process is the DT depicted in Figure 1. The tree has a quite manageable size: it consists of 25 nodes numbered from `#0` to `#24` and divided into 12 condition nodes and 13 leaves, and has a maximum depth of 5 levels, that is, any prediction is the result of checking 5 conditions at most. In the figure, each node shows number of samples of each class (`alive` and `not_alive`, respectively) that were traversed through at that point during training. For instance, node `#2` shows the text `values = [70, 23]` meaning that 93 samples have reached that node, 70 `alive` and 23 `not_alive`, respectively. The ratio between these two categories is also represented by the background color of the cell, in a scale that goes from dark green for nodes with higher proportion of `alive` to dark red for higher proportion of `not_alive`, using white in the middle of the scale, that is, when the ratio of the two classes is 50%/50% (as happens in node `#16`, for instance). Finally, decision (or leaf) nodes include the predicted class whereas condition nodes include their corresponding conditions, as expected. When evaluated against the test set, the tree produced an accuracy of 0.87. Since the target feature was unbalanced, also F1 measure and, most significantly, kappa score were obtained, which resulted in 0.71 and 0.62, respectively. These are reasonable results for a simple DT classifier.

3. A Tool for Explaining Decision Trees

As explained in the introduction, our interest in this work is not only to obtain a good DT classifier as the one in Figure 1, but also to provide explanations associated to its predictions. In this section, we describe our tool

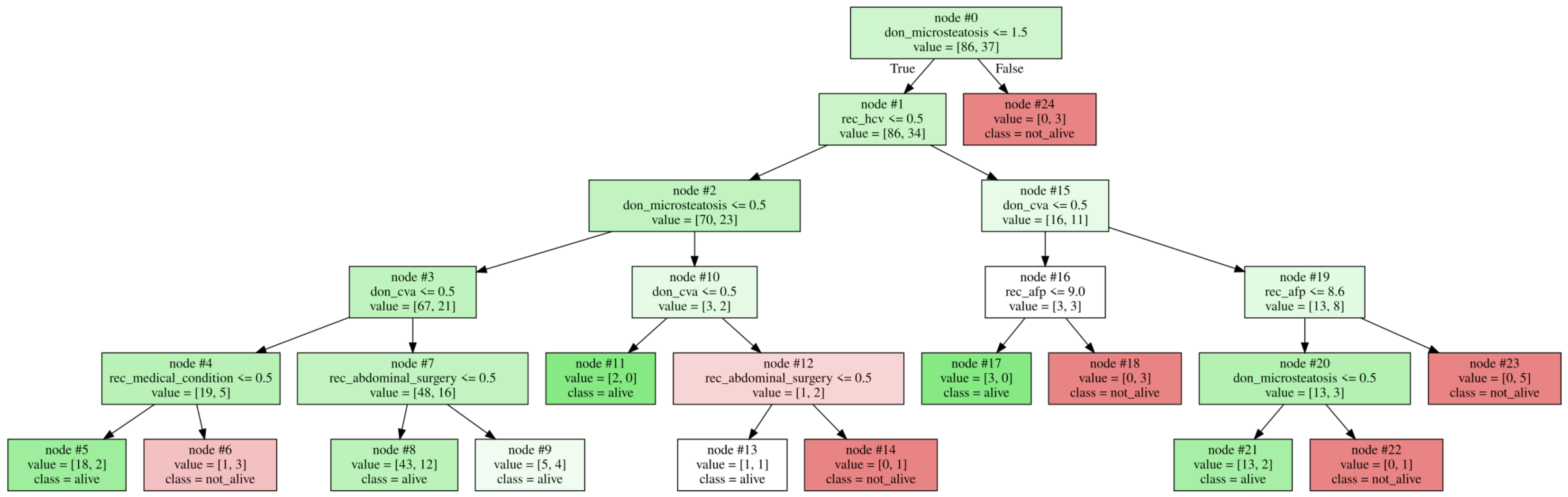

Figure 1: Obtained Decision Tree classifier.

Table 5: Grid Search parameters

Parameter	Possible values
maximum depth	5, 9, 11
splitting criterion	entropy, gini-importance
maximum features	$\sqrt{n\_features}$, $log_2(n\_features)$
Best parameters	9, entropy, $\sqrt{n\_features}$

```
*
|___Good forecast: more than 5 years, for recipient 126
|  |___The donor's microsteatosis is low (below 30%)
|  |___The donor's cause of death was not a CVA
|  |___The recipient is hepatitis c-negative
```

Listing 1: Example of explanation corresponding to the path in Figure 2.

called `Crystal-tree` which produces text explanations from a given decision tree and a set of values for the input features. The explanations include the outcome of the decision tree together with those conditions satisfied by the input values that led to the conclusion. These explanations, are provided in natural language, in a summarized-way as shown, for instance, in Listing 1 used to explain the path from Figure 2. The explanations must be read as a summarized representation of the path followed by the input sample (i.e. a recipient) until a decision node is reached. The explanations produced by `Crystal-tree` are text-based and are organized in a two level-tree structure. The first level shows the decision of the tree (for Listing 1 the recipient will survive more than 5 years) for the given input. The second level summarizes the conditions met by the input sample (i.e. the recipient) to follow a certain path and to produce the final decision. The latter includes only one sentence per each feature used to reach the decision, so that: (1) features not used in the path of the tree are not included in the explanation; and (2) all the conditions over a repeatedly used feature within the path are summarized as the narrowest range of all the thresholds. As an example, note that, the explanation shown in Listing 1 does not mention the medical condition of the recipient because it does not appear in the path (Figure 2) used to decide the outcome. Also, despite the path includes two conditions about the

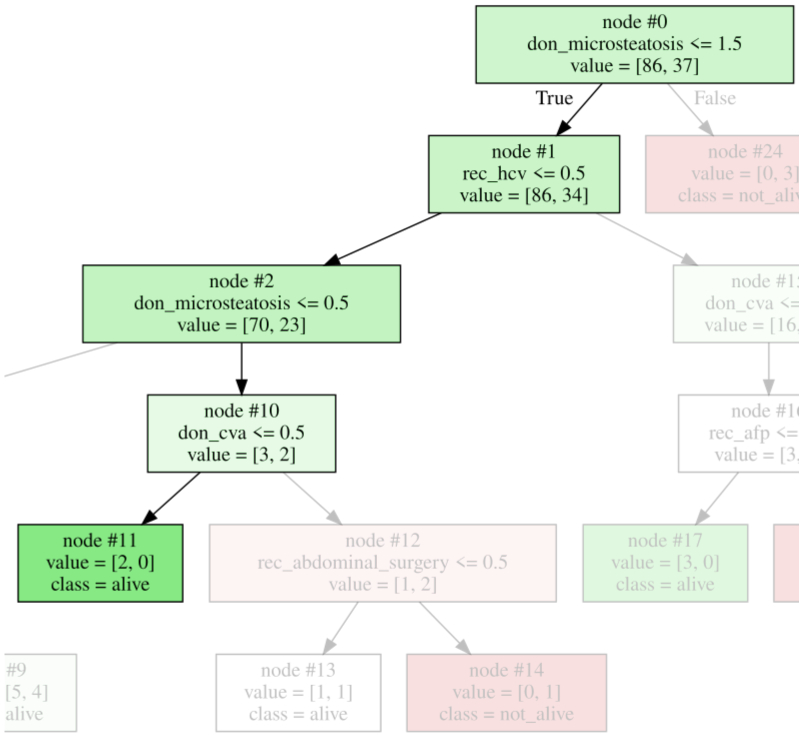

Figure 2: A complete path in the decision tree.

```
*
|___Good forecast: more than 5 years, for recipient 126
|  |___The donor had not fatty liver
|  |___The donor's cause of death was not a stroke
|  |___The recipient has not an hepatitis C virus infection
```

Listing 2: A version of the explanation in Listing 1 for non-specialists.

```
*
|___Buen pronóstico: más de 5 años, para el receptor 126
|  |___Órgano donante de buena calidad
|  |___El receptor no tiene una infección por hepatitis C
|  |___El donante no tenía hígado graso
```

Listing 3: Spanish version of the explanation in Listing 2.

donor's microsteatosis (less than 60% and less than 30%) both of them were summarized in just one sentence.

By default, the tool generates domain-independent explanations, including conditions that just refer to the variable name and its possible values, like for instance, `rec_hcv = 0`. However, the user can personalize the text in the explanations using some sort of templates: note how `rec_hcv = 0` is replaced by '`The recipient is hepatitis-c negative`' in the explanation of Listing 1. This feature can be used for adapting the explanations to different kind of situations, or users. With the same decision tree and input data, the outcome explanation can be adapted to different technical levels, to give more or less detail (some variables can be hidden, even under certain conditions) or to be expressed in different languages. As an illustration, Listing 2 provides an alternative explanation intended for a non-specialist user, such as the recipient herself, whereas Listing 3 displays the same explanation but generated using the Spanish text labels.

The behavior of `Crystal-tree` is controlled through a `Python` library, so other kind of adaptations are potentially possible. For instance, considering again the translation example, the explanations could be dynamically translated to any language selected by the user. Listing 4 shows an example of usage of the `Python` library. From lines 1 to 9, the code loads the data,

```python
import sklearn
from Crystal-tree import CrystalTree

# Loads a dataset
X, y = sklearn.datasets.load_iris(return_X_y=True, as_frame=True)

# Trains a decision tree
clf = sklearn.tree.DecisionTreeClassifier()
clf.fit(X,y)

# Translates the classifier into an explainable logic program
crys_tree = CrystalTree(clf)

# Creates the labels used by the tree
# if skipped, default labels will be used
setup_traces(crys_tree)

# Print explanations for input X
crys_tree.explain(X)
```

Listing 4: Example of usage of the `Crystal-tree` library.

and trains a Decision Tree classifier (which is saved in the variable `clf`) on the whole data set. Line 12 creates a `CrystalTree` object from the trained decision tree. Line 19 generates an explanation for each input sample in the vector `X`. As explained before, by default, `Crystal-tree` will use generic text for generating the explanations. To adapt the text to the domain, additional `Python` code is needed. The function `setup_traces` in line 16, adapts the explanations to get the same result as shown in Listing 1. Listing 5 shows the code within the function. To personalize the explanations, `Trace` objects are added into the `CrystalTree` object. This will introduce new text in the explanations, which will overwrite the default. These `Trace` objects can be associated either to a certain target class, as in the lines 4 and 7, or to a certain feature, in which case, a set of conditions can be set. If the feature is used in the corresponding path and the provided conditions are satisfied by the input sample, then the explanation will include the trace text. The specified text pattern can also include three special place holders, `%_class`, `%_instance` or `%_t` (in lines 5 and 8, and 32 respectively) that will be replaced by the names of the class, the identifier of the input sample or the value of the threshold used, respectively. More detailed instructions on this topic (including examples) together with the source code and installation instructions are available in the `Crystal-tree` public github repository[3].

4. Understanding Tree Explanations

As explained before, a trained DT is just a binary tree where each non-leaf node (or *condition* node) is labelled with some condition of the form $x \leq v$ where x is some input feature and v is a value for x called the *threshold*. For instance, the root (node `#0`) of the DT in Figure 1 checks whether the donor's liver microsteatosis is smaller or equal than 1.5. Each condition node has two children sub-trees that correspond to following the cases where the condition is true (in the figures, the left subtree) or false (the right subtree), respectively. A leaf node, also called *decision* node, specifies the final predicted classification (in our example, whether the recipient will survive after 5 years or not).

As we saw in Figure 2, a prediction for a given transplantation case is simply obtained as the result of following the path in the tree, answering

[3]All files are publicly available in https://github.com/bramucas/Crystal-tree

```python
from Crystal-tree import Trace, Condition

def setup_labels(crystal_tree_object):
    crystal_tree_object.add_trace(
        Trace("Good forecast: %_class, for recipient %_instance",
        "prediction",class=0))
    crystal_tree_object.add_trace(
        Trace("Bad forecast: %_class, for recipient %_instance",
        "prediction", class=1))
    crystal_tree_object.add_trace(
        Trace("The donor's microsteatosis is low (below 30%)",
        "don_microsteatosis", conditions=[Condition("is",0)]))
    crystal_tree_object.add_trace(
        Trace("The donor's microsteatosis is low (between 30% and 60%)",
        "don_microsteatosis", conditions=[Condition("is",1)]))
    crystal_tree_object.add_trace(
        Trace("The donor's microsteatosis is low (over 60%)",
        "don_microsteatosis", conditions=[Condition("is",2)]))
    crystal_tree_object.add_trace(
        Trace("The donor's cause of death was not a stroke",
        "don_cva", conditions=[Condition("is",False)]))
    crystal_tree_object.add_trace(
        Trace("The donor's cause of death was stroke",
        "don_cva", conditions=[Condition("is",True]))
    crystal_tree_object.add_trace(
        Trace("The recipient is hepatitis c-negative",
        "rec_hcv", conditions=[Condition("is",False]))
    crystal_tree_object.add_trace(
        Trace("The recipient is hepatitis c-positive",
        "rec_hcv", conditions=[Condition("is",True]))
    crystal_tree_object.add_trace(
        Trace("The recipient's level of AFP is lower than %_t",
        "rec_afp", conditions=[Condition("lower_and_equal"]))
```

Listing 5: Adding labels to a `Crystal-tree` object.

each condition in accordance with the feature values from the input. It is not difficult to see that DTs that are structurally different may end up providing the same classification for any sample. For instance, we could just reorganize the tree by switching the order in which the input features are checked. However, the particular structure of the DT obtained by the learning algorithm is not casual: it actually reflects statistical information that was used during the learning process. The algorithm proceeds recursively, introducing a new condition that *splits* the current data set into two new subsets in a way that the entropy of the result is minimized (that is, the information gain is maximized). The larger the difference between the ratios for each class in a subset, the lower the entropy is. Informally speaking, the upper a condition node is in the tree, the more it helps to "*clarify the picture*" with respect to a statistical partition of the data set. In this way, this arrangement helps in minimizing the average number of questions (depth of the path) we make to obtain a prediction. However, it also produces curious effects when reading the followed path as an explanation for the prediction. A first counterintuitive effect is that a same variable may be checked several times (even more than twice) along a path depending on different thresholds values. As an example, take again Figure 1 and follow the path that reaches node `#21` from the root node `#0`. This path checks that `don_microsteatosis` is below two different thresholds. First, we check in the root node checks that microsteatosis is smaller than 1.5 but then, in node `#20`, at the fifth level, we check again that it is *below the smaller threshold* 0.5. The reason for the first check is that, during training, the condition `don_microsteatosis > 1.5` was able to discriminate a subset formed only by `not_alive` samples, leading to a zero entropy decision node `#24`. But the truth is that, if our purpose is to provide an explanation for the decision node `#21`, the first check is clearly redundant, since being below 0.5 obviously implies that it is also below 1.5. Fortunately, this redundancy can be easily removed from the explanations. The tool `Crystal-tree` compresses each explanation using the narrowest interval of values eventually required to follow the prediction path, hiding the possible redundant tests made by the tree.

A second, and more important, counterintuitive effect that appears when explaining a prediction by simply reading a path is that some of the conditions we check along the path can be *causally opposed* to the prediction eventually obtained. To see an example, suppose we had a transplant case with the following data:

```
don_microsteatosis = 1
rec_hcv = 1
don_cva = 0
rec_afp = 3
```

and we proceed to generate an explanation. If we follow the DT conditions using these data, we get the path in Figure 3 reaching node `#17`, whereas its corresponding explanation is displayed in Listing 6. While the explanation

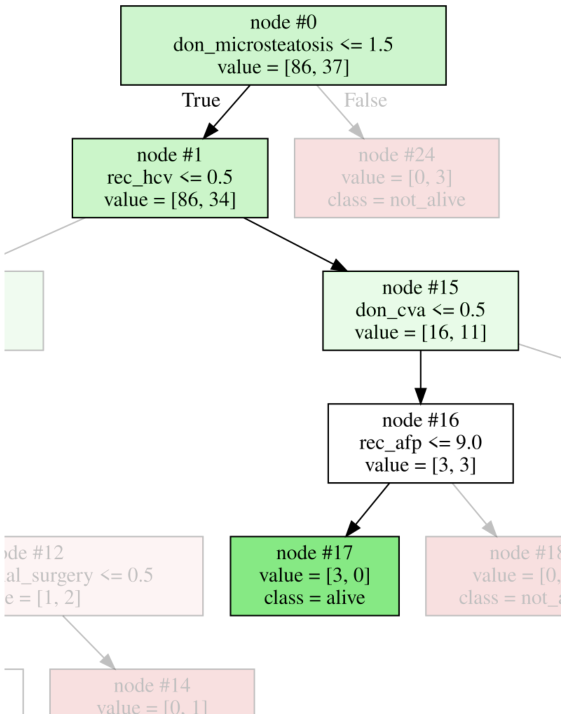

Figure 3: Another path in the decision tree, explained by Listing 6.

predicts a good forecast, only two of the conditions used in the path are known to be good justifications for the prognosis: the donor did not die due to a stroke (`don_cva=0`) and the recipient's levels of tumor marker AFP are low (`rec_afp`$\leq$9). Still, the explanation also uses the facts that the donor's microsteatosis is below 60% (`don_microsteatosis`$\leq$1.5) and that the recipient is HCV positive (`rec_hcv=1`). What may seem puzzling in

```
*
|___Good forecast: more than 5 years, for recipient 114
|  |___The donor's microestatosis is low-medium (below 60%)
|  |___The recipient is hepatitis c-positive
|  |___The donor's cause of death was not a stroke
|  |___AFP is lower than 9.0 ng/ml
```

Listing 6: Explanation corresponding to the path in Figure 3.

this example is that these two facts are not known to particularly contribute to the good prognosis: in fact, the presence of HCV could be seen as a negative factor, rather than a positive one. The problem here comes from a misunderstanding of the conditions presented in the explanation: they just correspond to the nodes followed by the tree to make a prediction but *cannot be read as necessary causes* strictly required to achieve that prediction. In other words, the truth for some of these conditions could be changed without affecting the final result. Consider, for instance, the node asking about HCV: in the complete tree, there are more cases with `not alive` when the patient is infected than when she is not. However, our current patient is HCV-positive, and so, the DT must follow that part of the tree in order to make a decision. It is after checking that the donor did not have a stroke and that the AFP is low that the tree eventually classifies the case as `alive`. Moreover, suppose that we have the same data but the patient were not HCV-positive: if we now follow the tree, we discover that the prediction is still `alive`, this time, by reaching node `#11`. In other words, with the rest of input data unchanged, *the prediction will still be* `alive` *regardless of the value of HCV*. To sum up, the fact that HCV-positive appears in the explanation just reflects how the DT has organized the conditions to be checked, but cannot be read as a necessary cause to explain the prediction. An interesting possible future line of research is to design an algorithm that collects minimal sets of necessary conditions for each possible prediction made by the tree, but this exceeds the scope of the current work.

5. Logic Programming Implementation

One of the interesting features of `Crystal-tree` is that the generation of explanations is implemented using a logic programming back-end. In principle, implementing an *ad hoc* procedural algorithm from scratch for the same purpose would not bee too difficult. The main advantages of the logic programming implementation, however, are its flexibility and declarativeness. `Crystal-tree` transforms the DT into a set of logical rules interpreted under the *stable models* (or *answer sets*) semantics [13], so these rules can be seen as a formal specification of the classifier. This has multiple potential advantages, like enabling variations of the classifier behavior by simple changes in the logical rules, studying the classifier results under uncertainty (i.e. when some variables are unknown) or allowing formal comparisons (like the analysis of equivalence) to other DTs, when represented as logic programs. Moreover, `Crystal-tree` uses an already existing declarative tool, `xclingo` [6], that allows generating explanations for logic programs under the answer sets semantics. In the rest of this section, we provide a brief, informal overview of the implementation of `Crystal-tree` based on logic programming.

Answer Set Programming (ASP) [14] is a declarative problem solving paradigm where a problem is represented as a set of rules in a logic program and its solutions are obtained in the form of models of that program called *answer sets*. ASP rules have the general form of

```
head :- body.
```

where, in the simplest case, `head` is some predicate atom and `body` is a (possibly empty) list of *literals*, that is, predicate atoms optionally preceded by negation operator `not`. Intuitively, the rule allows deriving `head` as true when all the literals in the body are also true. When a rule has an empty body, we just keep the head dropping the symbol '`:-`' and the rule receives the name of *fact*, since its head will be unconditionally true. For instance, the following program:

```
holds(55,rec_hcv,true).
holds(55,don_cva,true).
bad(I) :- holds(P,rec_hcv,true), holds(P,don_cva,true).
```

consists of two facts (the first two lines) and a (conditional) rule. In this example, the facts are representing that transplantation case (or patient) number `55` had an HCV-positive recipient and an CVA donor. Note that the conditional rule is using a capitalized argument `P`. This stands for a (universally quantified) logical variable: the rule states that any patient `P` with HCV and whose donor had CVA will be classified as `bad(P)`. As a result, the answer set for this simple program will also derive the fact `bad(55)`. This is, for instance, the result obtained by the popular ASP solver `clingo` [15].

`xclingo` [6] is an ASP tool built on top of `clingo` that partially implements the multi-valued extension of logic programs defined in [16]. The tool, in addition to obtaining the solutions of an ASP program, it also explains why a given atom was derived by tracing the relevant fired rules. To this aim, we may use textual descriptions associated to rules or atoms in the program. As an illustration, Listing 7 shows an annotated version of the previous example program and Listing 8, the `xclingo` output. The directive `%!trace_rule` allows associating a textual label to the rule occurring immediately below, when the rule condition is applied. This textual label can be parametrized using variable placeholders `%`. In our example, this placeholder is replaced by the patient number `P`. The directive `%!trace` allows attaching a text label to any derivation of a given atom, regardless of the rule(s) used to derive it. In the example, we are specifying that any derivation of `holds(P,F,V)` will be transformed into the text "`F is V`" (the patient number `P` is not used). As a result, for instance, the fact `holds(55,rec_hcv,true)` becomes the label "`rec_hcv is true`". Finally, the directive `%!show_trace` is used to choose which atoms are going to be explained: in this case, we decide to display explanations for the atom `bad(P)`.

The original DT obtained during the learning process was automatically encoded into two different `xclingo` implementations: `nodes.lp` and `paths.lp`[4]. We also use two additional files: `extra.lp`, which contains common code for both implementations, and `cases.lp`, which contains the data from the transplantation cases to be predicted represented as ASP facts.

The `nodes.lp` program (partially shown in Listing 9) directly represents each DT edge using predicate `tree_node(N,Instance,Dir)` where `N` is the child node to be activated and `Dir` its direction below the tree (left or right). Note that, as the logic programs do not accept floating point numbers, the resulting DT translations scale the numeric values to integers; when the explanations are finally constructed, these values are returned to floating point numbers automatically. As we can see, each rule is annotated with a `%!trace_rule` describing the decision condition. Leaves are encoded as rules with `class(I,C)` where `I` identifies the sample to be predicted and explained,

[4]All files publicly available in https://github.com/bramucas/Crystal-tree

```
holds(55,hcv,true).
holds(55,don_cva,true).
%!trace_rule {"Patient % may fail",P}
bad(P) :- holds(P,hcv,true), holds(P,don_cva,true).
%!trace {"% is %",F,V} holds(P,F,V).
%!show_trace bad(P).
```

Listing 7: `xclingo` annotated program.

```
*
|__Patient 55 may fail
|  |__rec_hcv is true
|  |__don_cva is true
```

Listing 8: `xclingo`'s explanation for `bad(55)`

```
tree_node(0,I,left) :- le(I,rec_hypertension,50).
tree_node(1,I,left) :- gt(I,rec_hcv,50), tree_node(0,I,left).
                          (...)
tree_node(6,I,left) :- le(I,rec_afp,635), tree_node(5,I,left).
class(I, alive) :- tree_node(6,I,left).
```

Listing 9: Fragment from `nodes.lp`.

```
class(I, alive) :-
 le(I,rec_abdominal_surgery,50),
 gt(I,don_cva,50),
 gt(I,rec_hcv,50),
 le(I,rec_hypertension,50),
 between(I,rec_afp,509,635),
 le(I,don_microsteatosis,50).
```

Listing 10: Fragment from `paths.lp`.

```
%!trace {"% > %", F, T} gt(I,F,T).
%!trace {"% <= %", F, T} le(I,F,T).
%!trace {"% in (%,%]", F, Min, Max} between(I,F,Min,Max).
%!trace {"Class: % (instance %)", C, I} class(C,I).
```

Listing 11: Traces used by `paths.lp`.

and `C` represents the decided class for that sample (i.e. `alive` or `not_alive`). Listing 12 shows an example of explanation generated using `nodes.lp`. As we can see, the cascade form reflects the order in which conditions are applied when traversing the tree, which comes from the (decreasingly) discriminatory power of each condition. However, as a tree grows in depth, they become less readable and most discriminant features tend to be used repeatedly with different thresholds (as happens here with `rec_afp`) making the explanation less clear.

```
*
|___Class: not_alive (instance 14)
|  |___rec_afp > 5.09
|  |  |___don_microsteatosis <= 0.5
|  |  |  |___rec_afp <= 6.35
|  |  |  |  |___rec_abdominal_surgery <= 0.5
|  |  |  |  |  |___don_cva > 0.5
|  |  |  |  |  |  |___rec_afp <= 12.44
|  |  |  |  |  |  |  |___rec_hcv > 0.5
|  |  |  |  |  |  |  |  |___rec_hypertension > 0.5
```

Listing 12: Example of explanation generated by `nodes.lp`.

On the other hand, `paths.lp` (Listing 10) just encodes a rule per each leaf in the original tree. The head of the rule encodes the class of the leaf, and the body is a conjunction of all conditions traversed in the path. The explanation from Listing 12 obtained with `nodes.lp` is reformulated as Listing 13 when using `paths.lp`. As we can see, the order in which the conditions are checked when following the DT is not reflected any more: all conditions occur at the same level. However, we get the advantage that each feature involved in the explanation is displayed *only once*, showing the value or interval of values that has been used to get the final prediction. For this reason, `Crystal-tree` translates any decision tree into its `paths.lp` version by default for obtaining explanations.

```
*
|___Class: not_alive (instance 14)
|  |___rec_abdominal_surgery <= 0.5
|  |___don_cva > 0.5
|  |___rec_hcv > 0.5
|  |___rec_hypertension <= 0.5
|  |___rec_afp in (5.09,6.35]
|  |___don_microsteatosis <= 0.5
```

Listing 13: Example of explanation generated by `paths.lp`.

6. Conclusions

We have presented a flexible method for explaining predictions made by decision trees in human-readable terms. We have implemented this method in the form of a publicly available `Python` tool we have called `Crystal-tree`. As a real domain application, we have used this tool to obtain explanations for long-term (five years) survival predictions for recipients of a liver transplant. To this aim, we have used machine learning techniques to obtain a decision tree from a data set collected at the liver transplantation unit from the Coruña University Hospital Center (CHUAC) covering the transplant cases from 2009 to 2014. Using this decision tree, we have provided different examples of the explanations we obtain from `Crystal-tree`, including multiple adaptations for different languages and/or different levels of expertise. We have shown how the resulting explanations avoid repetitive references to a same feature, keeping a simpler description, something crucial for larger or deeper decision trees. Also, we have discussed why these tree-based explanations should not be understood under a causal reading, since the conditions displayed are not always necessary to obtain a prediction, but just reflect the decisions taken by the tree. Finally, we have briefly explained the logic programming implementation of the tool.

As positive conclusions, we emphasize the good evaluation results obtained by the trained Decision Tree, given the (relatively) small size of the collected data set, the lack of balance in the selected target feature and the added difficulty which is intrinsic to a long-term prediction of five years.

Also, the `Crystal-tree` library is pretty straightforward and easy to use for any user familiar with `Python` machine learning tools, which is, in fact, one of the most used environments for machine learning nowadays. Besides, the explanations obtained from the tool always remain short and simple, even for deeper decision trees. This is because we display, at most, one justification line per each different feature used in a tree path, avoiding the repetition of conditions on a same variable. Also, the explanation labels are easily adaptable for different purposes and contexts without too much effort, by adding some `Python` lines of code.

Some aspects can be improved yet, pointing directions for future work. On the one hand, even though the definition of a set of text labels for the explanations is extremely simple, it still requires some `Python` programming. The implementation of a graphical user interface to facilitate this task, especially thinking on medical experts, is planned as a future addition. On the other hand, as we discussed in Section 4, the explanations generated from a decision tree do not take into account any potential causal relation between the conditions displayed and the prediction. This may easily lead to a misunderstanding, since an explanation for some prediction may include conditions that are not actually necessary to produce the that prediction. As future work, we plan to tackle this problem by exploring the minimization of the logic program that represents the tree. In that way, the result would no longer have the form of a tree, but a set of rules instead, whereas the logic program would still be equivalent as a classifier, producing the same set of predictions. The advantage of this method would be that we could still generate explanations from the minimal program whereas the conditions of its rules would exclusively contain necessary checks for obtaining the prediction. Another convenient feature to be added to the tool would be to include probabilities extracted from the DT to the explanations. This would be helpful for the final user to increase her level of trust in the system. Lastly, we also plan to continue collecting more transplant cases for the data set.

Acknowledgements. This work was partially supported by Ministerio de Ciencia e Innovación (MICIN) (grant PID2020-116201GB-I00) and Xunta de Galicia (grants GPC ED431B 2019/03 and ED431G 2019/01 for CITIC center).

References

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

[2] K. Connor, E. O'Sullivan, L. Marson, S. Wigmore, E. Harrison, The future role of machine learning in clinical transplantation, Transplantation Publish Ahead of Print.

[3] J. Briceño, M. Cruz-Ramírez, M. Prieto, M. Navasa, J. Urbina, R. Orti, M. A. Bravo, A. Otero, E. Varo, S. Tome, G. Clemente, R. Bañares, R. Bárcena, V. Cuervas-Mons, G. Solorzano, C. Vinaixa, A. Rubín, J. Colmenero, A. Valdivieso, M. García, Use of artificial intelligence as an innovative donor-recipient matching model for liver transplantation: Results from a multicenter spanish study., Journal of hepatology J Hepatol. 2014 Nov; (2014) 1020–8. `doi:10.1016/j.jhep.2014.05.039`.

[4] O. Nitski, A. Azhie, F. A. Qazi Arisar, X. Wang, S. Ma, L. Lilly, K. Watt, J. Levitsky, S. Asrani, D. Lee, B. Rubin, M. Bhat, B. Wang, Long-term mortality risk stratification of liver transplant recipients: real-time application of deep learning algorithms on longitudinal data, The Lancet Digital Health 3. `doi:10.1016/S2589-7500(21)00040-6`.

[5] D. Bertsimas, J. Kung, N. Trichakis, Y. Wang, R. Hirose, P. Vagefi, Development and validation of an optimized prediction of mortality for candidates awaiting liver transplantation, American Journal of Transplantation 19.

[6] P. Cabalar, J. Fandinno, B. Muñiz, A system for explainable Answer Set Programming, in: Proc. of the 36th Intl. Conf. on Logic Programming (ICLP, Technical Communications), Vol. 325 of EPTCS, 2020, pp. 124–136.

[7] A. Niculescu-Mizil, C. Perlich, G. Swirszcz, V. Sindhwani, Y. Liu, P. Melville, D. Wang, J. Xiao, J. Hu, M. Singh, W. Shang, Y. Zhu, Winning the KDD cup orange challenge with ensemble selection, Journal of Machine Learning Research - Proceedings Track 7 (2009) 23–34.

[8] N. Chawla, K. Bowyer, L. Hall, W. Kegelmeyer, Smote: Synthetic minority over-sampling technique, J. Artif. Intell. Res. (JAIR) 16 (2002) 321–357. `doi:10.1613/jair.953`.

[9] I. Mani, I. Zhang, knn approach to unbalanced data distributions: a case study involving information extraction, in: Proceedings of workshop on learning from imbalanced datasets, Vol. 126, ICML United States, 2003.

[10] C. J. V. Rijsbergen, Information Retrieval, 2nd Edition, Butterworth-Heinemann, 1979.

[11] J. Cohen, A coefficient of agreement for nominal scales, Educational and Psychological Measurement 20 (1) (1960) 37–46. `arXiv:https://doi.org/10.1177/001316446002000104`, `doi:10.1177/001316446002000104`.
URL https://doi.org/10.1177/001316446002000104

[12] R. Artstein, M. Poesio, Inter-coder agreement for computational linguistics, Computational Linguistics 34 (2008) 555–596.

[13] M. Gelfond, V. Lifschitz, The Stable Model Semantics For Logic Programming, in: Proc. of the 5th International Conference on Logic Programming (ICLP'88), Seattle, Washington, 1988, p. 1070â 1080.

[14] G. Brewka, T. Eiter, M. Truszczynski, Answer set programming at a glance, Communications of the ACM 54 (12) (2011) 92–103.

[15] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, P. Wanko, Theory solving made easy with clingo 5, in: M. Carro, A. King (Eds.), 32nd Intl. Conf. on Logic Programming (ICLP, Technical Communications), Vol. 52 of OASIcs, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016, pp. 2:1–2:15.

[16] P. Cabalar, J. Fandinno, M. Fink, Causal graph justifications of logic programs, Theory Pract. Log. Program. 14 (4-5) (2014) 603–618. `doi:10.1017/S1471068414000234`.
URL https://doi.org/10.1017/S1471068414000234