Action Rules: Counterfactual Explanations in Python *

Lukáš Sýkora and Tomáš Kliegr

Department of Information and Knowledge Engineering, Faculty of Informatics and Statistics, University of Economics, Prague {xsykl04, tomas.kliegr}@vse.cz

Abstract. Action rule mining is an extension of the widely used task of learning classification rules. In addition to information expressed in a standard classification rule, an action rule suggests a course of action. If performed, this action will increase the probability that the class of the instance will change to the desired value. Such rule can either be interpreted as a recommendation for an action, or as a *counterfactual explanation* for the class assigned to the instance. In this paper, we report on a new implementation of action rules discovery that is available in Python (ActionRules package) and on a new experimental method for learning action rules from large datasets (RandomForestRules package), which is based on extraction of classification rules from Random Forests. The paper can serve as a manual for using the created packages or as a guide for researchers who would like to extend them, providing also guide to action rule discovery. The text also includes performance evaluation of reduction trees, which speed up the mining process.

Keywords: Action Rules · Explainable Machine Learning · Classification · Rule Learning · Counterfactual explanation · Benchmark

1 Introduction

An action rule describes how an action (a change in the value in one or more flexible attributes) could impact the classification of a given object. Similarly as standard classification rules [5], action rules can also be learnt from data [9]. In addition to choosing the attribute serving as the predicted class, the required additional settings from the user is a designation of a subset of attributes as 'flexible'; the remaining attributes are considered as 'stable'. Also, the user needs to set which target class values are desired (wanted).

Considering the well-known Titanic dataset[1], an example of a generated action rule is: *"If attribute 'Sex' is 'female', attribute 'Embarked' value 'Southampton' is changed to 'Cherbourg', attribute 'Passenger Class' value '3' is changed to '1', then 'Survived' value '0' is changed to '1' with support: 5%, confidence:*

* Supported by University of Economics, Prague by grant IGA 12/2019.

[1] https://www.kaggle.com/c/titanic

58% and uplift: 6%." This action rule can either serve as recommendation for a course of action to take to increase the probability of survival, or more fittingly on this retrospective dataset, a counterfactual ("what if") explanation for the classification.

To the best of the author's knowledge, the only publicly available implementation of action rules is LISp-Miner[2] developed at the University of Economics in Prague, which is primarily focused on Windows users who use it via a Graphical User Interface. In this paper, we introduce an alternative implementation of action rule mining for Python.

The ActionRules package[3] covered in this paper implements the action rules algorithm as described in [9] with a performance improvement by a reduction tree described in [10]. We also describe its experimental extension available as RandomForestRules package[4], which extracts classification rules from tree ensembles generated by Random Forests. The packages are accompanied by several Jupyter Notebooks, demonstrating different use cases of the software. The packages are available under an open license (MIT).

The structure of the work is the following. In Section 2, we motivate the problem of action rule mining. Section 3 gives the details of the algorithmic approach used. In Section 4, we present a proposal for a new method for extracting association rules from Random forests. Section 5 describes the new implementation of action rules, which is the main contribution of this paper. Section 6 describes the attempted performance improvements: reduction trees and extracting action rules from Random Forests. In the conclusions, we summarize the contributions.

2 Background

Algorithmically, there are two principal approaches to action rule mining [4]: a rule-based approach and the object-based approach.

The *rule-based approach* is divided into two independent steps. In the first step, classification rules are mined by any suitable association rule mining algorithm (for example Apriori [2] modified for Classification Association Rules [7]). In the second step, action rules are generated from the classification rules.

In the *object-based approach*, action rules are discovered directly from the source dataset. Specific algorithms include LERS (Learning from Examples based on Rough Sets) or ERID (Algorithm for Extracting Rules from Incomplete Decision System) with atomic action sets [4].

In the software package introduced in this paper, we adopt the rule-based approach, which is in our opinion more modular. There is a range of highly performing association rule mining algorithms that can be adopted for the first step of classification rule mining.

[2] `https://lispminer.vse.cz/`
[3] `https://github.com/lukassykora/actionrules`
[4] `https://github.com/lukassykora/randomForestRules`

2.1 Phase 1: Classification Rule Mining

In the first step of the rule-based approach, classification rules are mined. A formal representation of a classification rule containing four conditions is:

$$r = [(a_1 \wedge b_1 \wedge c_1 \wedge e_1) \Rightarrow d_1], \tag{1}$$

where a_1, b_1, c_1, e_1 are values of features (attributes) in the *antecedent* of the rule, and d_1 is a value of the target variable predicted by the *consequent* of the rule.

Each discovered rule is accompanied by values of support and confidence that express its quality.

The support of a rule $ant \Rightarrow d_1$ (denoted as $sup(ant \Rightarrow d_1)$) is a number of transactions (rows in training data set) that match both the antecedent and consequent of the given rule.

The confidence of a rule $ant \Rightarrow d_1$ (denoted as $conf(ant \Rightarrow d_1)$) is expressed as $sup(ant \Rightarrow d_1)/sup(ant)$. It corresponds to a ratio between the number of transactions that satisfy the antecedent as well as the consequent to the number of transactions that satisfy the antecedent of the rule (denoted as $sup(ant)$).

2.2 Phase 2: Generation of Action Rules

In the second step, action rules are formed from a subset of discovered classification rules.

An action rule can be represented as:

$$r = [(\omega) \wedge (\alpha \rightarrow \beta)] \Rightarrow [\phi \rightarrow \psi], \tag{2}$$

where ω is a fixed condition (set of stable attributes), which describes the object, $(\alpha \rightarrow \beta)$ is the proposed to change to a subset of user-designated flexible attributes and $(\phi \rightarrow \psi)$ represents the implied change to the target attribute.

The quality of action rules can be represented using the following measures. Let r be an action rule, which was generated from two classification rules r_1 and r_2. We compute the support of the action rule r as $sup(r) = min(sup(r_1), sup(r_2))$. The confidence of the action rule r is computed as $conf(r) = conf(r_1) * conf(r_2)$. These definitions were adopted from [4] and simplified. Note that other somewhat different definitions can also be found in the literature [11]. Uplift is a measure used to predict an incremental response to an action [8]. We propose to adapt this measure for the purpose of evaluating an action rule r as follows: $uplift = P(decision|treatment) - P(decision|no\ treatment)$. The instances are divided into two groups : control and exposed. Exposed group ($treatment$) is exposed to the recommended action whereas control group ($no\ treatment$) is suppressed from the recommended action. The *decision* expresses the classification of the instances.

Note that the definitions of support and confidence for action rules are different from these definitions for classification rules.

2.3 Example

Below is an example of two classification rules discovered from the Titanic dataset.

$$r_1 = [(Sex : female \wedge Embarked : S \wedge Pclass : 3) \Rightarrow Survived : 0], \\ with\ support\ 10\%\ and\ confidence\ 55\%. \quad (3)$$

$$r_2 = [(Sex : female \wedge Embarked : C \wedge Pclass : 1) \Rightarrow Survived : 1], \\ with\ support\ 6\%\ and\ confidence\ 86\%. \quad (4)$$

Rule r_1 can be interpreted as follows: women who embarked on a voyage in Southampton in the third class did not survive with probability of 55%. Rule r_2 can be interpreted as follows: women who embarked on a voyage in Cherbourg in the first class survived with probability of 86%.

Based on these two classification rules, the following action rule can be generated:

$$r = [(Sex : f) \wedge (Embarked : S \rightarrow C) \wedge (Pclass : 3 \rightarrow 1)] \Rightarrow [Surv. : 0 \rightarrow 1], \\ with\ support\ 5\%,\ confidence\ 59\%\ and\ uplift\ 5.7\%. \quad (5)$$

This action rule can be interpreted as follows: women who travelled from Southampton in the third class would have increased their chances of survival if they had changed their boarding place to Cherbourg and had paid extra money for the first class.

3 Action Rule Generation with Rule-based Approach

In the following, we will outline the phase 2 of the rule-based approach for action rule generation, in which the algorithm attempts to form action rules from all pairs of classification rules.

The process is the following:

- Discovered classification rules are (conceptually) represented as a table where each condition in the antecedent is a column. There is also one column for consequent (target variable).
- Conditions in the antecedent are divided into stable and flexible. Stable conditions do not allow any change of their state (no action can be taken). Flexible conditions allow changing their state (the action can be taken).
- The table of classification rules is divided into two tables. The first table X_β contains classification rules without the desired state. In the second table X_α, there are classification rules with the desired state.

– Classification rule pairs are created as a Cartesian product of the rows in these two tables $X_\beta \times X_\alpha$. Each of these $rowcount(X_\beta) * rowcount(X_\alpha)$ classification rule pairs is a candidate for an action rule. The action rule candidate is therefore formed from a rule predicting other than a desired class *state before* and a rule predicting a desired class *state after*.
– The algorithm loops through all candidates. If there are no pairs left, the algorithm finishes, and returns the saved action rules.
– On each loop, all the stable antecedent pairs and flexible antecedent pairs are checked. If the conditions given in Section 3.1 are satisfied, the pair is changed to an action rule and saved.

3.1 Conditions for Validation of Classification Rules Pairs

There are several approaches to check whether a pair of classification rules satisfies the conditions and can be used for generation of an action rule.

Baseline approach The approach, which we consider as a baseline, is described in [4].The same stable attribute needs to be present in both rules and their values must be the same. Flexible attributes of both classification rules must be present in both rules and their values must be different.

Values of the target attribute in the consequent must comply with the user setting. For example, when generating action rules for the Titanic dataset, we would like to change the target state from Survival: No to Survival: Yes, but not in the opposite way.

Extended action rules An alternative approach described in [9] may generate more action rules, because it allows to generate candidates from the input classification rules even if they have missing values. For example, this approach would generate rules such as: *"If **any** 'Pclass' value is changed to '1', then 'Survived' value '0' is changed to '1'."*

3.2 Speed Up by Reduction Tree

A pre-processing step for speeding up the phase 2 of action rule mining is described in [10]. This step is implemented in the software package introduced in this paper.

This approach is based on the splitting of the table of all classification rules to multiple small tables, which allows pruning the search space. For each stable attribute, one table with rules is generated, since the values of stable attributes must always be the same in the state before and the state after in the action rule candidate pair. Some tables can be eliminated because they do not contain the desired target value, or they do not have enough variability, see Figure 1.

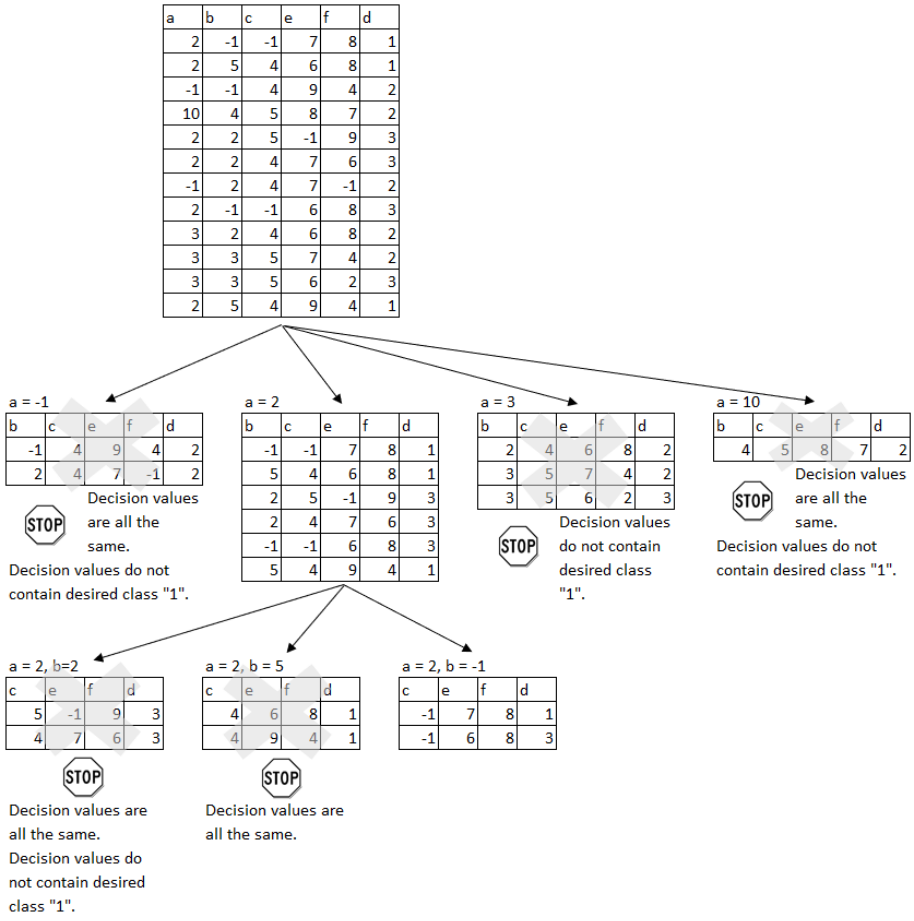

Fig. 1: Reduction tree for baseline action rules mining. Desired class d is '1', stable attributes are a and b, and flexible attributes are c, e and f. Each row represents one classification rule.

4 Extraction of Action Rules from Random Forest

A challenge for classification rule mining are datasets containing many features and many rows. In this case, association rule classification algorithms used in phase 1 of action rule generation can fail due to excessive combinatorial complexity.

Random Forest [3] is a classification learning method that produces an ensemble of decision trees. The target class is selected by voting. This method is known for its ability to process very large multidimensional datasets.

Classification rules can be extracted from a Random Forest model and used as an input to phase 2 of action rule generation. The proposed algorithm for generation of action rules from Random Forests based on this principle is outlined in the following:

- Features in the input data are subject to one-hot encoding. This replaces each multinominal column with n distinct values with $n-1$ binary columns.
- Random Forest classifier is learnt on the recoded dataset.
- All decision trees are extracted from the Random Forest Classifier.
- Extracted decision trees are transformed to classification rules.
- Classification rules not meeting minimum support and minimum confidence thresholds are removed.

According to preliminary experiments, extraction of classification rules from a Random Forest is much faster on datasets, which are difficult to process for association rule mining algorithms, such as Apriori. The disadvantage of this approach is that the the standard algorithm for inducing Random Forests is neither deterministic nor exhaustive. Association rule mining algorithms guarantee that a complete set of rules valid with respect to user-set minimum confidence and minimum support thresholds is found. Furthermore, there is no random element involved in association rule mining.

5 Implementation

The ActionRules package allows two ways of operation. Either the complete mining workflow can be executed, using Apriori for phase 1, or only action rule generation (phase 2) can be initiated from classification rules supplied as a data frame.

The ActionRules package and RandomForestRules package are available in PyPi (Python Package Index). They can be installed by the following commands:

```
pip install actionrules-lukassykora
pip install randomForestRules-lukassykora
```

5.1 ActionRules package

The package for action rules mining uses PyFIM[5] library as a default option for classification rule discovery. The PyFIM package returns an exhaustive list of classification rules used as input for action rule generation.

The ActionRules package uses the same general machine learning workflow as Scikit-Learn [1], a popular data mining library.

Instantiate model object

Firstly, the model is instantiated.

Code 1 *Instantiate model object*

```
from actionrules.actionRulesDiscovery import ActionRulesDiscovery

actionRDiscovery = ActionRulesDiscovery()
```

Fit model to training data

In the next step, the model is fit to training data. In this case, Titanic dataset is used. All features are pre-processed to be nominal.

In the example listing below, the stable attribute is *Age*, flexible attributes are *Embarked*, *Fare* and *Pclass*, the consequent is set to *Survived*. Minimum confidence is set to 55% and minimum support to 3%. These thresholds are used only for generating classification rules (phase 1). Desired class for *Survived* is *1*. Mining of extended action rules is not enabled (*is_nan=False*) and reduction trees are used to speed up phase 2 (*is_reduction=True*). Both the minimum number of stable attributes and the minimum number of flexible attributes in action rules is set to *1*.

Code 2 *Fit model to training data*

```
actionRDiscovery.read_csv("data/titanic.csv", sep="\t")
actionRDiscovery.fit(stable_attributes = ["Age"],
                     flexible_attributes =
                      ["Embarked",
                      "Fare",
                      "Pclass"],
                     consequent = "Survived",
                     conf=55,
                     supp=3,
                     desired_classes = ["1"],
                     is_nan=False,
                     is_reduction=True,
                     min_stable_attributes=1,
                     min_flexible_attributes=1)
```

[5] `http://www.borgelt.net/pyfim.html`

Predict & Evaluate

This is the last step of the action rule mining workflow.

Code 3 *Predict & Evaluate*

```
new_data = [['32-48', 'S', 'very high' ,'3.0']]
df_new_data = pd.DataFrame(new_data, columns = ["Age",
                                                "Embarked",
                                                "Fare",
                                                "Pclass"])
actionRDiscovery.predict(df_new_data)
```

As a result, the application returns a table with recommended actions (marked with *).

Table 1: Titanic - prediction. Recommended actions are marked with *.

Age	**Embarked**	**Embarked***	**Fare**	**Pclass**	**Pclass***	**Sex**	**Target**	**Uplift**
32-48	S	C	very high	3	1	female	Survived	6%
32-48	S	-	very high	3	1	female	Survived	3%

5.2 ActionRules package combined with RandomForestRules package

The example below shows how to use the package RandomForestRules together with the ActionRules package. The dataset Audiology [6] is used for demonstration. Mining with Apriori using PyFIM fails on this dataset with the following setting: minimum support: 0.5%, minimum confidence: 50% and unlimited length of final itemsets.

[6] `http://archive.ics.uci.edu/ml/datasets/audiology+(standardized)`

Code 4 *ActionRules package combined with RandomForestRulespackage*

```
from randomForestRules import RandomForest
from actionrules.actionRulesDiscovery import ActionRulesDiscovery
import pandas as pd

df = pd.read_csv("data/audiology.csv")

randomForest = RandomForest() # Initialize
randomForest.load_pandas(df) # Load
randomForest.fit(antecedent = cols, consequent = 'binaryClass', supp=0.005, conf=50) # Fit
frame = randomForest.get_frame() # Get dataframe with classification rules

actionRulesDiscovery = ActionRulesDiscovery() # Initialize
actionRulesDiscovery.load_pandas(frame) # Load classification rules
actionRulesDiscovery.fit_classification_rules(stable_attributes = stable_col_names, # Fit
                                flexible_attributes = flex_col_names,
                                consequent = 'target',
                                conf_col = 'confidence',
                                supp_col = 'support',
                                desired_classes = [1])
print(actionRulesDiscovery.get_action_rules_representation()) # Print the result
```

Output:
`rule 1.` [(age-gt-60: t) ∧ (bone: unmeasured) ∧ (history-noise: t → f)] ⇒ [target: 0 → 1] with support: 0.018 and confidence: 0.561
`rule 2.` [(age-gt-60: t) ∧ (bone: mild) ∧ (history-noise: t → f)] ⇒ [target: 0 → 1] with support: 0.018 and confidence: 0.561
`rule 3.` [(age-gt-60: t) ∧ (bone: ?) ∧ (history-noise: t → f)] ⇒ [target: 0 → 1] with support: 0.018 and confidence: 0.561
`rule 4.` [(age-gt-60: t) ∧ (ar-u: absent) ∧ (history-noise: f → t) ∧ (o-ar-c: absent → normal)] ⇒ [target: 0 → 1] with support: 0.001 and confidence: 0.666
`rule 5.` [(age-gt-60: t) ∧ (ar-u: absent) ∧ (history-noise: f → t) ∧ (o-ar-c: elevated → normal)] ⇒ [target: 0 → 1] with support: 0.001 and confidence: 0.666
`rule 6.` [(age-gt-60: t) ∧ (ar-u: absent) ∧ (history-noise: f → t) ∧ (o-ar-c: ? → normal)] ⇒ [target: 0 → 1] with support: 0.001 and confidence: 0.666

6 Evaluation

The evaluation was run on a notebook Dell Latitude E7470 with configuration Intel Core i5-6300U 2.40GHz with 8GB RAM. The Titanic dataset is used for fitting the models.

6.1 ActionRules Package Performance

In this section, we evaluate the effect of the use of extended action rules and of reduction trees on the performance. We use the ActionRules package for the whole process of action rules discovery.

Setup The minimum confidence was fixed to 50%, and the minimum support threshold was varied. The evaluation was performed in four configurations: reduction trees enabled/disabled, extended association rules enabled/disabled. All configurations are always tested three times and the median result is used for comparison.

Results Results are reported in Figure 2. The left graph shows the performance for the baseline approach to action rules mining (certainty). The right graph shows the performance for extended action rules mining (uncertainty). The runtime of calculation with 'uncertainty' is more than 100x higher, but also the number of found action rules is higher (2443 extended action rules vs. 105 action rules). The reduction trees have a consistent positive effect on performance.

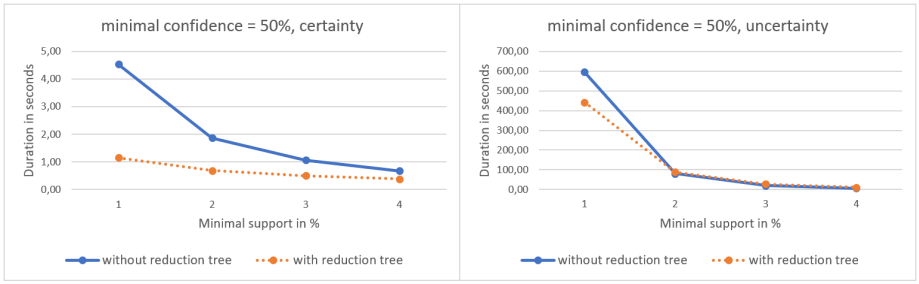

Fig. 2: Evaluation of performance of ActionRules package on Titanic dataset.

6.2 RandomForestRules Package Performance

Setup The setting for classification rules discovery was the following: minimum support 3% and minimum confidence 55% (these thresholds are used just for filtering of rules extracted from the Random Forest). The package uses the RandomForestClassifier[7] from Scikit-learn library. All settings are kept as default, just for n_estimators several values (10,20,30) were tested. The author also separately evaluates generation of action rules from a passed list classification rules with the following setting: stable_attributes = ["Age", "Sex"], flexible_attributes = ["Embarked", "Fare", "Pclass"], consequent = "Survived", desired_classes = ["1.0"].

Results Results are reported in Table 2. As can be seen, only very few unique classification rules were extracted from the Random Forest models, which resulted in no discovered action rules. The probable reason is that the Random Forest algorithm creates many duplicate trees (or their parts), which are largely

[7] `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`

Table 2: RandomForestRules vs. APRIORI. on Titanic dataset

Package	n_estimators	Duration	Count classif. r.	Count action r.
PyFIM (Apriori)		0.5 s	189	13
RandomForestRules	30	225.8 s	11	0
RandomForestRules	20	112.0 s	9	0
RandomForestRules	10	62.6 s	7	0

overlapping. If the PyFIM package that uses the Apriori algorithm was used for the same task, 13 action rules were discovered.

On the other hand, generating classification rules using RandomForestRules was successful for the Audiology dataset, where application of association rule mining to generate candidates was not successful (cf. Section 5.2).

7 Conclusion

In this paper, we described two new Python packages for action rule mining. The first package ActionRulesDiscovery is primarily intended for mining of action rules from classification rules generated with association rule mining (from class association rules). The second experimental package RandomForestRules extracts classification rules from Random Forest models. This package can be combined with ActionRulesDiscovery to facilitate discovery of action rules from very large multidimensional datasets.

In terms of performance, the approach taken in ActionRulesDiscovery benefits from the use of PyFIM package, which wraps the high performing low-level implementation of Apriori [6] for generating candidate classification rules. The ActionRulesDiscovery package is written in Python. The package could be extended by rewriting the action rule generation algorithm to C or C++, which would make the mining process faster. The direction of using random forests to generate action rules is promising, but needs further investigation.

References

1. An introduction to machine learning with scikit-learn. scikit-learn (2019), `https://scikit-learn.org/stable/tutorial/basic/tutorial.html`
2. Agrawal, R., Srikant, R., et al.: Fast algorithms for mining association rules. In: Proc. 20th int. conf. very large data bases, VLDB. vol. 1215, pp. 487–499 (1994)
3. Breiman, L.: Random forests. Machine learning **45**(1), 5–32 (2001)
4. Dardzinska, A.: Action rules mining, vol. 468. Springer (2012)
5. Fürnkranz, J., Gamberger, D., Lavrač, N.: Foundations of rule learning. Springer Science & Business Media (2012)
6. Hahsler, M., Chelluboina, S., Hornik, K., Buchta, C.: The arules r-package ecosystem: analyzing interesting patterns from large transaction data sets. Journal of Machine Learning Research **12**(Jun), 2021–2025 (2011)
7. Jovanoski, V., Lavrač, N.: Classification rule learning with apriori-c. In: Portuguese Conference on Artificial Intelligence. pp. 44–51. Springer (2001)

8. Kumar, A., Kumar, R.: Uplift modeling: Predicting incremental gains (2018)
9. Ras, Z.W., Wieczorkowska, A.: Action-rules: How to increase profit of a company. In: European Conference on Principles of Data Mining and Knowledge Discovery. pp. 587–592. Springer (2000)
10. Raś, Z.W., Wyrzykowska, E., Wasyluk, H.: Aras: Action rules discovery based on agglomerative strategy. In: International Workshop on Mining Complex Data. pp. 196–208. Springer (2007)
11. Tzacheva, A.A., Raś, Z.W.: Action rules mining. International Journal of Intelligent Systems **20**(7), 719–736 (2005)