User Manuel

KDD Final Project - ARED Implementation for Discovering Action Rules

Omar Eltayeby, Alexander Hohl, Dongdong Li & Brianna Chen - December 3, 2015

Table of Contents

Introduction

The goal of this project is to implement an algorithm that finds action rules, and to apply it to the Failed States Index dataset. The primary focus is the practical implementation of the algorithm. We discarded the possible use of DEAR algorithms [1] in favour of ARED [2], because no prior extraction of classification rules is needed and action rules can be extracted from the database directly.

Installation

Before running the project please make sure your computers environment is equipped with Python (vs 2.7 or greater) and Webpy.

- **Check if you have python 2.7.x installed**
- **Open up CMD window, type “python"**
    - **If nothing show up, please visit https://www.python.org/downloads/windows/**
- **After installation, add directory of python to system "path" for convenience**
- **Visit here https://pypi.python.org/pypi/setuptools**
- **Download Windows (simplified) ez_setup.py**
- **Append "%PYTHON_HOME%\Scripts" , separated by a semicolon ; at the end of path variable in your environment settings as instructed above, as usual enter echo %PATH%**
- **Open cmd "Run as administrator"**
- **Change directory to the folder containing the file, run "python ez_setup.py"**
- **Run "easy_install web.py"**

You are good to go!

Running the Program

- **Download the KDD_Project folder from your email or from Google Drive.**

- **Open your command line as admin**
  - right click command prompt
  - click "Run as administrator"

- **Navigate to the root folder of the KDD_Project.**
  - "cd Users/…./KDD_Project"

- **Type python Processing/action_rules.py**
  - NOTE: If you type cd Users/…/KDD_Project/Processing then python action_rules.py the program will not work. It must be run from the root directory

- **Copy and paste the address given in command line into your browser.**
  - Should be something like "http://0.0.0.0:8080/"
  - Looks best on full screen

- **Input desired values into input section and the rules will be displayed in the main body section**
  - NOTE: The program has been given default values so rules will be generated on first navigation to html page
  - NOTE: The appropriate inputs are in whole number or decimal format. Fractions will not be accepted.
  - You will also see output in your terminal

- **Code can be analyzed in detail in the IDE or text editor of your choice**
  - We used SublimeText for our implementation

Data Discretization

The Fragile States Index (FSI) is an annual ranking of 178 nations based on their levels of stability and the pressures they face. It is published by The Fund For Peace, a non-profit organization that works to prevent violent conflict and promotes sustainable security.

- Fragile States Data set was taken from: http://fsi.fundforpeace.org/rankings-2014

- Data set was discretized in Excel to rules set forth by project guidelines: n.k by n+1 if k>5 and by n if k < 6
    - All values with n.5 were searched and replaced by the rounded down digit
    - The rest were then rounded using Excels build in round functionality
    - Stable attributes added were: Population (in millions), and Neighbors with Equal or Less Stable Values. These attributes were also discretized to previously stated requirements.
    - We replaced the last column by these categories: Alert, Warning, Stable, Sustainable. As they consist of sub-categories (e.g. High Alert, Very High Alert), we merged them to form the final categories.
    - Format is .csv

Algorithm Selection

For our groups algorithm implementation we chose to implement the ARED algorithm. The reasons for this were three-fold.

1) We wanted our algorithm to be able to directly create rules from the dataset without having to find classification-rules first, so we chose to do an Object-Based approach.

2) We determined that given the size of the data-set it would be more efficient to do a single algorithm to discover the action rules rather then having to incorporate a separate algorithm for classification rules.

3) Given the material covered in class and the time spent on the Action-Rules Algorithms, we determined the best Object-Based Algorithm to implement was the ARED algorithm due to availability of information and previous study.

Algorithm Implementation

We used Python programming language for algorithm implementation and HTML to create the GUI. It enables the user to specify stable, flexible and decision attributes, as well as minimum support and confidence. The input is restricted to positive integers for support and floating-point numbers between 0.0 and 1.0 for confidence. The algorithm will find action rules according to the user-specified parameters and display them within the browser, together with support and confidence. We make heavy use of built-in data structures, such as dictionaries and lists, for storing the FSI dataset, as well as notations and rules.

Stable Attributes:

- Population
- Number of Neighbors with Equal or Lesser Stability

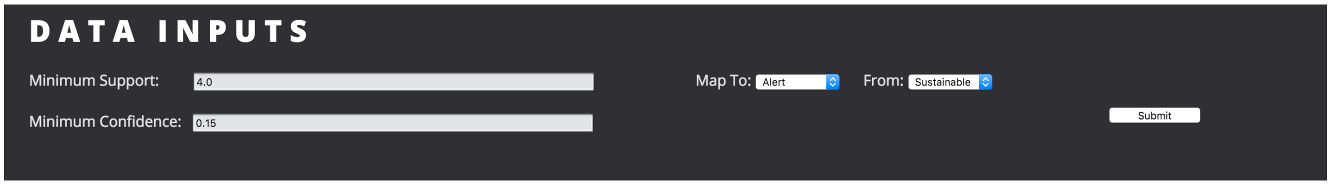
Decision Attribute:

- Stability Status

User Inputs:

- Minimum Support Threshold *(Integer or Float, if given a float will just give the support of the next higher integer)*
- Minimum Confidence Threshold *(0 to 1, with 0 being 0% and 1 being 100%)*
- Attribute Map To *(Alert, Warning, Stable, Sustainable)*
- Attribute Map From *(Alert, Warning, Stable, Sustainable)*

Example Input


DATA INPUTS

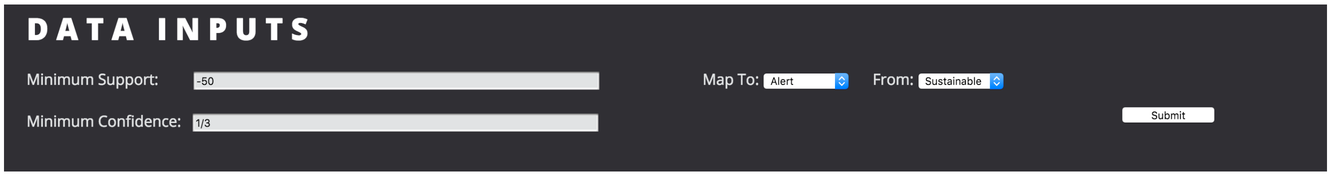
Minimum Support: 4.0

Minimum Confidence: 0.15

Map To: Alert

From: Sustainable

Fig 1. Correct Entry


DATA INPUTS

Minimum Support: -50

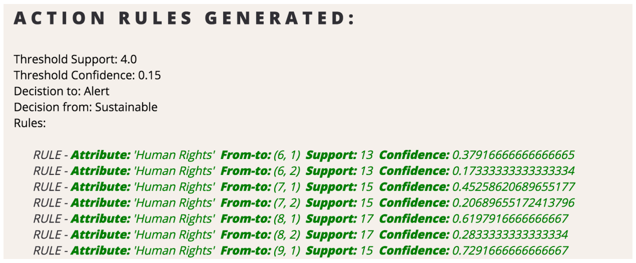
Minimum Confidence: 1/3

Map To: Alert

From: Sustainable

Fig 2. Incorrect Entry

Example Output

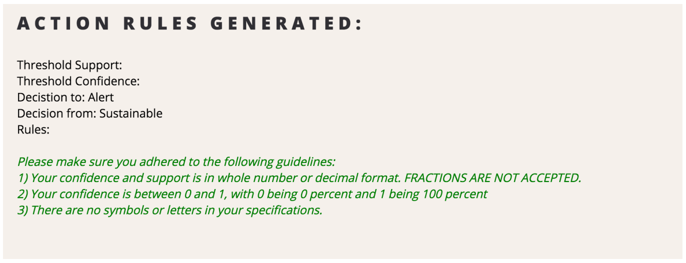


```
ACTION RULES GENERATED:

Threshold Support: 4.0
Threshold Confidence: 0.15
Decistion to: Alert
Decision from: Sustainable
Rules:

    RULE - Attribute: 'Human Rights'  From-to: (6, 1)  Support: 13  Confidence: 0.37916666666666665
    RULE - Attribute: 'Human Rights'  From-to: (6, 2)  Support: 13  Confidence: 0.17333333333333334
    RULE - Attribute: 'Human Rights'  From-to: (7, 1)  Support: 15  Confidence: 0.45258620689655177
    RULE - Attribute: 'Human Rights'  From-to: (7, 2)  Support: 15  Confidence: 0.20689655172413796
    RULE - Attribute: 'Human Rights'  From-to: (8, 1)  Support: 17  Confidence: 0.6197916666666667
    RULE - Attribute: 'Human Rights'  From-to: (8, 2)  Support: 17  Confidence: 0.2833333333333334
    RULE - Attribute: 'Human Rights'  From-to: (9, 1)  Support: 15  Confidence: 0.7291666666666667
```

Fig 3. Correct Entry Output



```
ACTION RULES GENERATED:

Threshold Support:
Threshold Confidence:
Decistion to: Alert
Decision from: Sustainable
Rules:

Please make sure you adhered to the following guidelines:
1) Your confidence and support is in whole number or decimal format. FRACTIONS ARE NOT ACCEPTED.
2) Your confidence is between 0 and 1, with 0 being 0 percent and 1 being 100 percent
3) There are no symbols or letters in your specifications.
```

Fig 4. Incorrect Entry Output

Algorithm Process:

- **Functions definitions:** there are a couple of functions that needs definition that will be used in this implementation description:
    - representsInt:
        - input: string
        - output: True is an integer in the string, False if not
    - notation_flexible:
        - input: list of unique values of a flexible attributes
        - output: list of tuples as the combination of these values between each other and themselves. This is used for generating the notations, which express the change of value of the classification attribute that could possibly be part of an action rule
    - notation_stable:
        - the same as notation_flexible but for stable attributes, so it generates notations, which are the transition (change of value) from and to the value itself.
    - notation2examples:
        - input: the notation and the attribute
        - output: the example indexes that match those notations at the specified attribute
    - findIntersection:
        - input: two lists
        - output: the intersection between those two lists

- testMark:
    - input: both sets of a notation
    - output: calculate the support and confidence and return whether the notation is positively, negatively or unmarked
- levelOne:
    - input: None
    - output: extract the positively, negatively or unmarked
- nextLevel:
    - input: unmarked notations of the previous level
    - output: extract the positively, negatively or unmarked

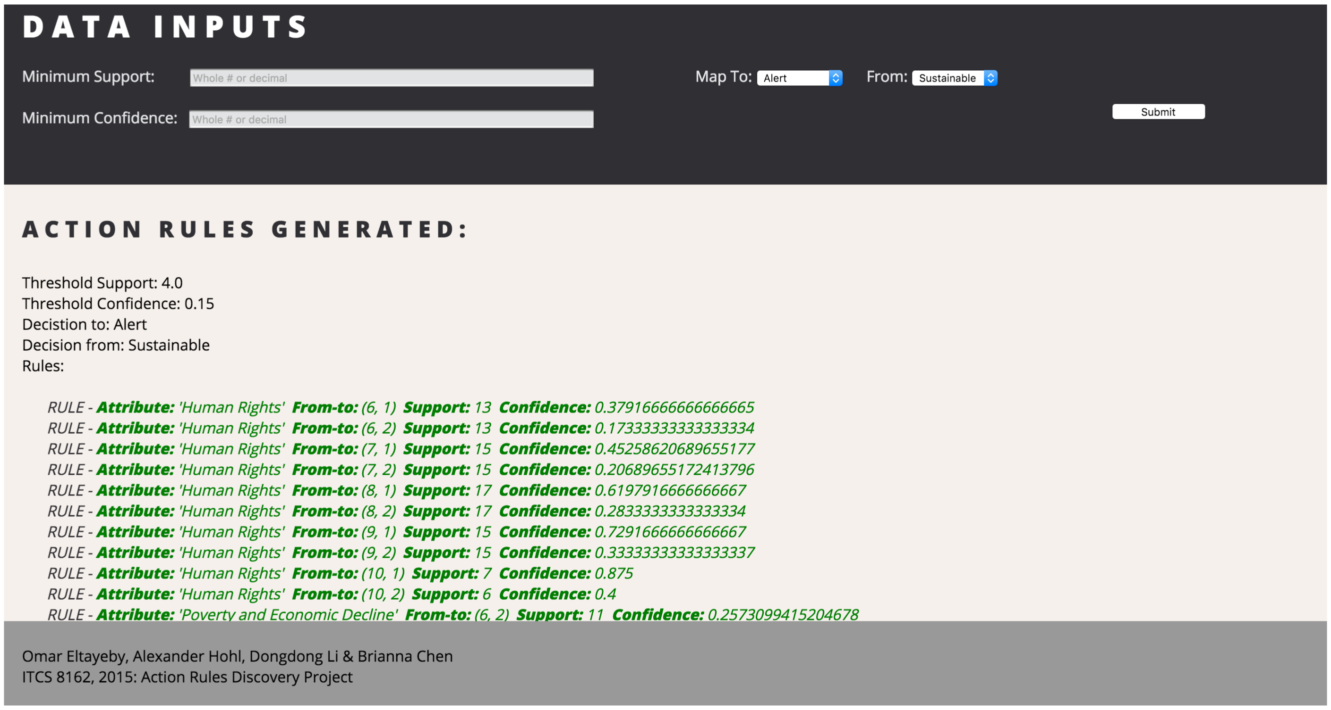
- **The procedure:** Take in user inputs: Threshold support, confidence and decision attribute to and from
    - Store stable attributes in a list
    - Read the csv file in ../Data/Fragile States Index 2014.csv and create a dictionary to store for each attribute all of its values (i.e. {"attribute1": [array of values], "attribute2": [array of values], and so on})
    - Get decision attribute's set of examples notation2examples
    - Get the positively and negatively marked, and unmarked for the first level using levelOne function
    - Find unique values of each attribute and store them in a dictionary
    - If stable attribute generate notations using notation_stable, by passing the unique values of the attribute
    - If flexible attribute generate notations using notation_flexible
    - All notations are a list of combinations of values changes. So for each notation:
    - Find examples' indexes for the notations using notation2examples
    - Get the mark for the notation using testMark by passing the examples' indexes
    - The return from the testMark are the mark (positive, negative, unmarked), support and confidence
    - Store the notation examples, the name of the attribute for that notation, the value change of the notation, support and confidence according to the mark returned
    - The positively, negatively and unmarked are separate lists. Each containing list of tuples. The tuples contain 6 arrays, the first two are for the example's indexes, the

third is for the attribute names, the fourth for the value change, the fifth for the support and the sixth for the confidence

- The notation examples' indexes are stored to be used in the next levels
- All of the positive notations are appended into one array for the rules
- Get the positively, negatively and unmarked for all next levels. Keep looping (while loop) until there are no more unmarked notations or the unmarked notations are saturated. Saturated means they are not being reduced (negatively marked) or rules coming out of them (positively marked)
- Call next level function by passing the unmarked notations, and return the new combinations of notations (positively, negatively and unmarked)
- nextLevel tests all unmarked notations by generating combinations between all notations
- The intersections of the examples' indexes of the previous level is the new notation
- We find the mark of the new notation using testMark
- The return of the testMark decides which set should the new notation be included in (positive, negative, unmarked)
- Also, the list of attributes of the old notations make up the new list of attributes of the new notations to be stored for the use of next levels. The same for the value change.
- The support and confidence are stored from the value returned testMark
- nextLevel returns positive, negative and unmarked
- Positive notations might have repeated attributes and values change when storing. Thus, filter them out (this does not change the rule)
- All of the filtered positive notations are appended into the same rules array used in first and previous levels
- Test whether the unmarked notations are saturated or not, if saturated break the while loop. If not test the while loop condition, which is testing unmarked notation array. If the array is empty exit the while loop, if not repeat from calling the next level part

GUI

Made with html and css along with webpy for integration into python code. Single page interface with straightforward intuitive form for user input entry. The parameters are set by typing numbers for support and confidence into the corresponding input boxes (e.g. Minimum Support: 2, Minimum Confidence: 0.5). The mapping of decision attribute values is specified by choosing one of the options in both boxes ("Map To:", "From"). The algorithm is executed once all parameters are specified and the "Submit" button is clicked



```
Threshold Support: 4.0
Threshold Confidence: 0.15
Decistion to: Alert
Decision from: Sustainable
Rules:

    RULE - Attribute: 'Human Rights'  From-to: (6, 1)  Support: 13  Confidence: 0.37916666666666665
    RULE - Attribute: 'Human Rights'  From-to: (6, 2)  Support: 13  Confidence: 0.17333333333333334
    RULE - Attribute: 'Human Rights'  From-to: (7, 1)  Support: 15  Confidence: 0.4525862068965517
    RULE - Attribute: 'Human Rights'  From-to: (7, 2)  Support: 15  Confidence: 0.20689655172413796
    RULE - Attribute: 'Human Rights'  From-to: (8, 1)  Support: 17  Confidence: 0.6197916666666667
    RULE - Attribute: 'Human Rights'  From-to: (8, 2)  Support: 17  Confidence: 0.2833333333333334
    RULE - Attribute: 'Human Rights'  From-to: (9, 1)  Support: 15  Confidence: 0.7291666666666667
    RULE - Attribute: 'Human Rights'  From-to: (9, 2)  Support: 15  Confidence: 0.33333333333333337
    RULE - Attribute: 'Human Rights'  From-to: (10, 1)  Support: 7  Confidence: 0.875
    RULE - Attribute: 'Human Rights'  From-to: (10, 2)  Support: 6  Confidence: 0.4
    RULE - Attribute: 'Poverty and Economic Decline'  From-to: (6, 2)  Support: 11  Confidence: 0.2573099415204678
```

Omar Eltayeby, Alexander Hohl, Dongdong Li & Brianna Chen
ITCS 8162, 2015: Action Rules Discovery Project

Figure 5: GUI for data inputs and execution of ARED. The interface consists of the following parts: A: Minimum support and confidence, B: Mapping of decision attribute values, C: Execution button, D: Action rules display.

Resources

- Code Language: Python 2.7, Webpy, html, css
- Text Editor: SublimeText 2
- Browsers Tested: Chrome, Firefox, Safari
- Additional Software Used: Excel, Terminal, Drive, Gmail
- Notes: http://webpages.uncc.edu/ras/KDD-Fall.html; Action Rules ppt 1 & 2
- References:

  - [1] Tsay*, L. S., & Raś, Z. W. (2005). Action rules discovery: system DEAR2, method and experiments. *Journal of Experimental & Theoretical Artificial Intelligence*, *17*(1-2), 119-128.
  - [2] Raś, Z., Dardzińska, A., Tsay, L. S., & Wasyluk, H. (2008, December). Association action rules. In *Data Mining Workshops, 2008. ICDMW'08. IEEE International Conference on* (pp. 283-290). IEEE.

Group Members

Omar Eltayeby, Alexander Hohl, Dongdong Li & Brianna Chen