# 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

| | |
|---|---|
| Table of Contents | 2 |
| Introduction | 3 |
| Installation | 3 |
| Running the Program | 4 |
| Data Discretization | 5 |
| Algorithm Selection | 5 |
| Algorithm Implementation | 6 |
| Algorithm Validation | 10 |
| GUI | 13 |
| Resources | 14 |
| Group Members | 14 |
| Appendix | 15 |

## 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

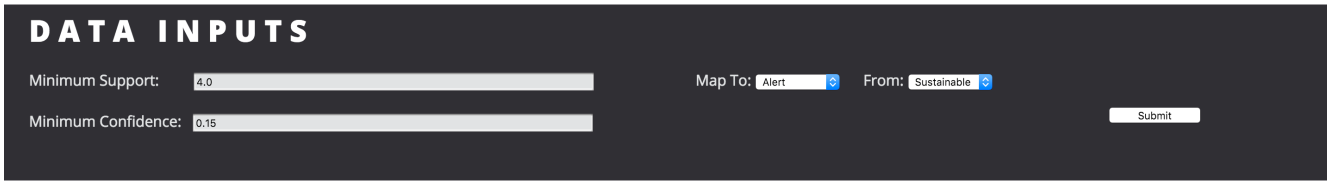
**Decision Attribute:**

- 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

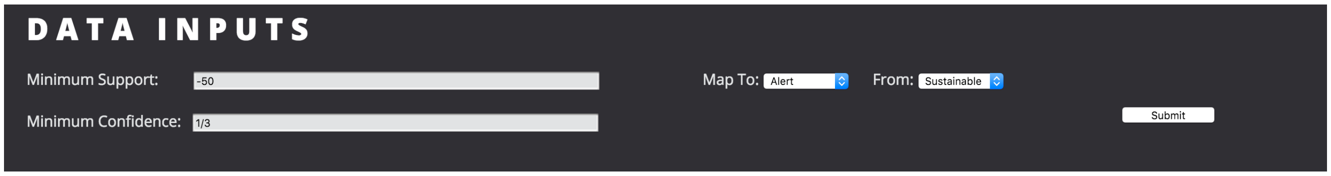
Minimum Support: 4.0

Minimum Confidence: 0.15

Map To: Alert

From: Sustainable

Fig 1. Correct Entry



DATA INPUTS

Minimum Support: -50

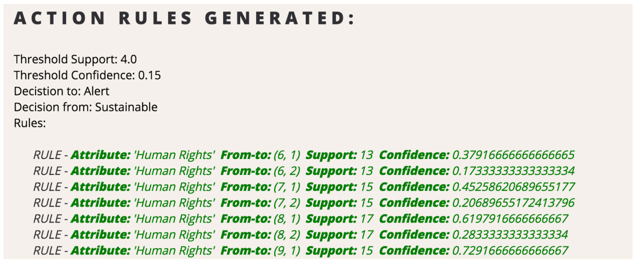
Minimum Confidence: 1/3

Map To: Alert

From: Sustainable

Fig 2. Incorrect Entry

## Example Output

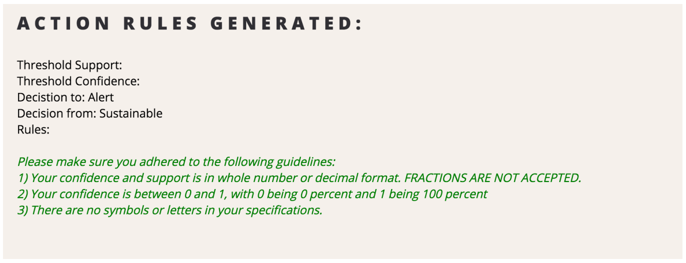


```
ACTION RULES GENERATED:

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

## Algorithm Validation

We conducted a two-step process to validate our implementation of ARED. First, we used the decision system S from (Table 1, see action_rules.ptt, slides 13 – 23), and compared the action rules produced by our implementation with the rules provided in the slides.

| X | a | b | c | d |
|---|---|---|---|---|
| x1 | a1 | b1 | c1 | d1 |
| x2 | a2 | b1 | c2 | d1 |
| x3 | a2 | b2 | c2 | d1 |
| x4 | a2 | b1 | c1 | d1 |
| x5 | a2 | b3 | c2 | d1 |
| x6 | a1 | b1 | c2 | d2 |
| x7 | a1 | b2 | c2 | d1 |
| x8 | a1 | b2 | c1 | d3 |

Table 1: Decision system S.

Therefore, we had an instance of a valid solution to the problem of generating action rules using ARED. We wanted to see whether our methodology produced the correct output, provided with identical input. The rules produced by our implementation matched the rules provided in the slides:

- [[(b,b1→b1)*(c,c1→c2)] → (d, d1→d2)]
- [[(a,a2→a1] → (d, d1→d2)]

In a second step, we wanted to see whether our implementation creates valid results using the Fragile States dataset. Since we did not have access to a solution, we conducted logical tests by back-tracking the resulting action rules through the levels of positively and unmarked notations and therefore, getting a better understanding of their genesis. We randomly chose 4 rules to be tested for changing the decision attribute from “Alert” to “Sustainable” (Table 1). We used threshold support of 5 and confidence 0f 0.15. The analysis showed that we could reconstruct all chosen action rules down the levels, therefore the implementation is valid.

| Action Rule 1 | Support | Confidence |
|---|---|---|
| Human Flight >> (5,2), Public Services >> (6,2), Refugees and IDPs >>(6,2) | 6 | 0.2500 |
| Level 3 notation positive mark (no positive marked notations in levels 4 and 5) | | |
| Human Flight >> (5,2), Public Services >> (6,2), Refugees and IDPs>>(6,2) | 6 | 0.2500 |
| Level 2 unmarked | | |
| Human Flight >> (5,2), Public Services >> (6,2) | 9 | 0.0804 |
| Refugees and IDPs (6,2) >>Public Services >> (6,2) | 8 | 0.0816 |
| Level 1 unmarked | | |
| Human Flight >> (5,2) | 10 | 0.0958 |
| Public Services >> (6,2) | 12 | 0.0962 |
| Refugees and IDPs >>(6,2) | 8 | 0.1134 |

| Action Rule 2 | Support | Confidence |
|---|---|---|
| Human Rights >> (6,1) | 13 | 0.3792 |
| Level 1 positive | | |
| Human Rights >> (6,1) | 13 | 0.3792 |

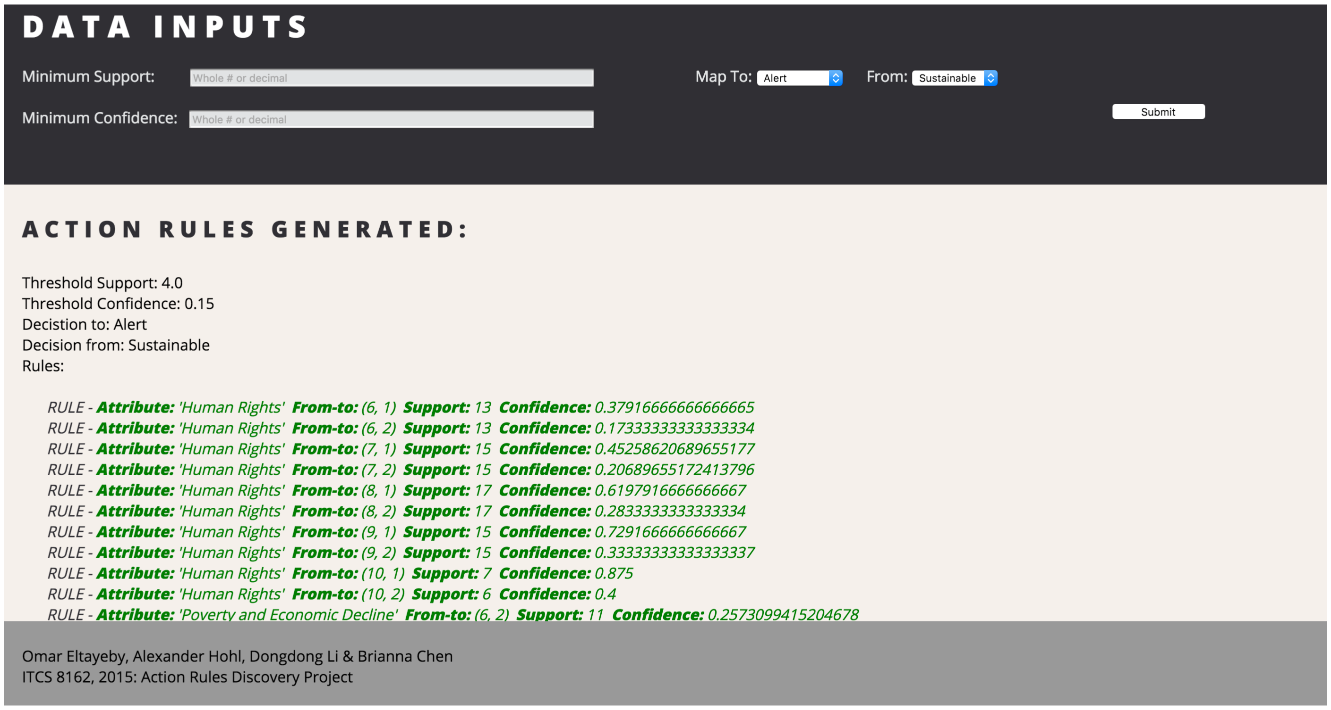
| Action Rule 3 | Support | Confidence |
|---|---|---|
| Poverty and Economic Decline >> (8,4), Group Grievance >> (6,2) | 6 | 1.0000 |
| Level 2 notation positive | | |
| Poverty and Economic Decline >> (8,4), Group Grievance >> (6,2) | 6 | 1.0000 |
| Level 1 unmarked | | |
| Poverty and Economic Decline >> (8,4) | 22 | 0.1214 |
| Group Grievance >> (6,2) | 10 | 0.1429 |

| Action Rule 4 | Support | Confidence |
|---|---|---|
| Human Flight >> (5,2), Public Services >> (6,2), Demographic Pressures >>(7,3) | 5 | 0.2778 |
| Level 3 notation positive | | |
| Human Flight >> (5,2), Public Services >> (6,2), Demographic Pressures >>(7,3) | 5 | 0.2778 |
| Level 2 unmarked | | |
| Human Flight >> (5,2), Public Services >> (6,2) | 9 | 0.0804 |
| Demographic Pressures >> (7,3) >>Public Services >> (6,2) | 6 | 0.1026 |
| Level 1 unmarked | | |
| Human Flight >> (5,2) | 10 | 0.0958 |
| Public Services >> (6,2) | 12 | 0.0962 |
| Demographic Pressures >> (7,3) | 8 | 0.1091 |

Table 2-6: Validation of our implementation of ARED.

## 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: 0.45258620689655177
    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
    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

# Appendix

## ACTION_RULES.PY

```
import csv
import itertools
import web
import cgi
import HTMLParser
import collections

#connect to GUI
urls = (
  '/', 'index'
)

#call render
app = web.application(urls, globals())
render = web.template.render('GUI/')

#global variables with default attributes
global supp
global conf
global decisionTo
global decisionFrom
global rules
global h
supp = 6.0
conf = 0.15
decisionTo = "Alert"
decisionFrom = "Sustainable"
rules = "Please make sure you adhered to the following guidelines: <br/> 1) Your confidence and
support is in whole number or decimal format. FRACTIONS ARE NOT ACCEPTED. <br/> 2) Your
confidence is between 0 and 1, with 0 being 0 percent and 1 being 100 percent <br/> 3) There are no
symbols or letters in your specifications."
# Stable attributes
stable_attr = ["Fragile States Index 2014", "Neighbors with Equal or Lesser Stability", "Population
(millions)"]
# Open the file and read
reader = csv.reader(open("Data/Fragile States Index 2014.csv", 'rU'), dialect=csv.excel_tab)
d1set = []
d2Set = []
attrDict = {}
uniqueValuesDict={}
h = HTMLParser.HTMLParser()
#class for GUI page
class index:
    def GET(self):
```

```
		form = web.input(minSup="6.0", minConf="0.15", mapTo="Alert",
mapFrom="Sustainable", rules = "none")
		sup  = "%s" % (form.minSup)
		con =  "%s" % (form.minConf)
		to  = "%s" % (form.mapTo)
		mapfrom = "%s" % (form.mapFrom)
		rules = "Please make sure you adhered to the following guidelines: <br/> 1) Your
confidence and support is in whole number or decimal format. FRACTIONS ARE NOT ACCEPTED.
<br/> 2) Your confidence is between 0 and 1, with 0 being 0 percent and 1 being 100 percent <br/> 3)
There are no symbols or letters in your specifications."
		#rules="hello <br> world"

		ruleSet = []

		try:
			sup = float(sup)
			con = float(con)
			if(con > 1):
				return render.index(sup, con, to, mapfrom, rules)
				#return(rules)
			if(con < 0):
				return render.index(sup, con, to, mapfrom, rules)
				#return(rules)
			if(sup < 0):
				return render.index(sup, con, to, mapfrom, rules)
				#return(rules)
		except ValueError:
			return render.index(sup, con, to, mapfrom, rules)
			#return(rules)

		setSupport(float(sup))
		setConfidence(float(con))
		setTo(to)
		setFrom(mapfrom)

		ruleSet = start()
		rul3 =  str(ruleSet)[1:-1].strip('()')#rul = ' '.join(ruleSet)
		rul3 = rul3.replace("'", '').replace(',,','')
		rules = rul3


		#return render.index(rules)
		return render.index(sup, con, to, mapfrom, rules)

#runs app
if __name__ == "__main__":
	app.run()

#sets support threshold from user entry
```

```
def setSupport(sup1):
        global supp
        supp = sup1

#sets confidence threshold from user entry
def setConfidence(con1):
        global conf
        conf = con1

#sets decision map-to from user entry
def setTo(to1):
        global decisionTo
        decisionTo = to1

#sets decision map-from from user entry
def setFrom(mfrom1):
        global decisionFrom
        decisionFrom = mfrom1

# Function to test if the string is an integer
def RepresentsInt(s):
    try:
        int(s)
        return True
    except ValueError:
        return False

# Function to get all combinations for the notations of a flexible attribute
def notation_flexible(inList):
        outList = []
        for i in inList:
            for j in inList:
                inter = []
                outList.append((i,j))
        return outList

# Function to get all combinations for the notations of a stable attribute
def notation_stable(inList):
        outList = []
        for i in inList:
                inter = []
                outList.append((i,i))
        return outList

# Function to find the example numbers of notations for a specific attribute
# Output: list of examples indexes
def notation2examples(notation, attr):
        examples=[]
        global attrDict
        for idx, eachValue in enumerate(attrDict[attr]):
```

```
        if eachValue==notation:
            examples.append(idx+1)
    return examples

# Function to find intersections between two lists
def findIntersection(l1,l2):
    return list(set(l1)&set(l2))

# Function to test whether the
def testMark(exampleNote1, exampleNote2):
    # Find intersections between the flexible attribute and the action attribute
    intersectionLeft = findIntersection(exampleNote1, d1Set)
    intersectionRight = findIntersection(exampleNote2, d2Set)
    # Calculate the support using the intersection
    supportLeft = len(intersectionLeft)
    supportRight = len(intersectionRight)
    if supportRight==0 or supportLeft==0:
        return (-1, 0, 0)
    support = max(supportLeft, supportRight)
    # Calculate the confidence
    confidenceLeft = float(supportLeft/float(len(exampleNote1)))
    confidenceRight = float(supportRight/float(len(exampleNote2)))
    confidence = float(confidenceLeft*confidenceRight)
    if support>=THsup:
        if confidence>=THconf:
            return (1, support, confidence)
        elif confidence==0:
            return (-1, support, confidence)
        else:
            return (0, support, confidence)
    else:
        return (-1, support, confidence)

def levelOne():
    positiveNotations=[]
    unmarkedNotations=[]
    negativeNotations=[]
    for attrValues in attrDict:
        # Unique values of each atttribute
        global uniqueValuesDict
        uniqueValuesDict[attrValues] = list(set(attrDict[attrValues]))
        # Generate notations for stable attributes, which are between each value and itself
        if attrValues in stable_attr:
            attrNotations = notation_stable(uniqueValuesDict[attrValues])
        # Generate notations for flexible attributes, which are between all values and each
other
        else:
            attrNotations = notation_flexible(uniqueValuesDict[attrValues])
        # For each combination
        for note in attrNotations:
```

```
                        # Find example indeces for the combination of the right and left sets, which are
the same
                        exampleNote1 = notation2examples(note[0], attrValues)
                        # If the attribute is stable the transition of attribute is between iteself
                        if attrValues in stable_attr:
                                exampleNote2 = exampleNote1
                        # If the attribute is flexible the transition of attribute is between the combinations
of different values
                        else:
                                exampleNote2 = notation2examples(note[1], attrValues)
                        # Store the attribute in order to extract it for listing all rules at the end
                        exampleNote3 = [attrValues]
                        # Store the transition (change) of values in order to extract it for listing all rules
at the end
                        exampleNote4 = [note]
                        (noteMark, support, confidence) = testMark(exampleNote1, exampleNote2)
                        exampleNote5 = support
                        exampleNote6 = confidence
                        if noteMark==0:
                                unmarkedNotations.append((exampleNote1, exampleNote2,
exampleNote3, exampleNote4, exampleNote5, exampleNote6))
                        elif noteMark>0:
                                positiveNotations.append((exampleNote1, exampleNote2,
exampleNote3, exampleNote4, exampleNote5, exampleNote6))
                        elif noteMark<0:
                                negativeNotations.append((exampleNote1, exampleNote2,
exampleNote3, exampleNote4, exampleNote5, exampleNote6))
        return (positiveNotations, unmarkedNotations, negativeNotations)

def nextLevel(unmarkedNotations):
        combUnmarkedNotations = list(itertools.combinations(unmarkedNotations, 2))
        positiveNotations=[]
        unmarkedNotations=[]
        negativeNotations=[]
        for eachComb in combUnmarkedNotations:
                exampleNote1 = list(set(eachComb[0][0]) & set(eachComb[1][0]))
                exampleNote2 = list(set(eachComb[0][1]) & set(eachComb[1][1]))
                # str3 = "Attribute: " +  str(eachComb[0][2]) + str(eachComb[1][2])
                # str4 = "From " + str(eachComb[0][3]) + " to " + str(eachComb[1][3])
                exampleNote3 = eachComb[0][2] + eachComb[1][2]
                exampleNote4 = eachComb[0][3] + eachComb[1][3]
                # exampleNote3 = str3
                # exampleNote4 = str4
                (noteMark, support, confidence) = testMark(exampleNote1, exampleNote2)
                exampleNote5 = support
                exampleNote6 = confidence
                if noteMark==0:
                        unmarkedNotations.append((exampleNote1, exampleNote2, exampleNote3,
exampleNote4, exampleNote5, exampleNote6))
                elif noteMark>0:
```

```
                              # str3 = "Attribute " + str(exampleNote1)
                              # str4 = "From " + str(exampleNote2)
                              # str5 = "To " + str(exampleNote3)
                              #positiveNotations.append((str3, str4, str5, exampleNote4))
                              positiveNotations.append((exampleNote1, exampleNote2, exampleNote3,
exampleNote4, exampleNote5, exampleNote6))
                    elif noteMark<0:
                              negativeNotations.append((exampleNote1, exampleNote2, exampleNote3,
exampleNote4, exampleNote5, exampleNote6))
          return (positiveNotations, unmarkedNotations, negativeNotations)

def start():
          num_row=0

          #Read data into array
          for row in reader:
                    values = row[0].split(",")
                    if num_row==0:
                              attributes = values
                              del attributes[0]
                              for attr in attributes:
                                        attrDict[attr] = []
                    else:
                              del values[0]
                              for idx, attr in enumerate(attributes):
                                        # Test if the string is an integer
                                        if RepresentsInt(values[idx]):
                                                  val = int(values[idx])
                                        else:
                                                  val = values[idx]
                                        attrDict[attr].append(val)
                    num_row+=1

          # Get decision attribute's set of examples first
          global decision1
          global decision2
          decision1=decisionTo
          decision2=decisionFrom
          decisionAttribute="Status"
          global d1Set
          global d2Set
          d1Set = notation2examples(decision1, decisionAttribute)
          d2Set = notation2examples(decision2, decisionAttribute)

          # Threshold support and confidence

          global THsup
          global THconf

          THsup=supp
```

```
        THconf=conf
        print "Sup:", THsup
        print "Con:", THconf

        level=1
        # Get the positively and negatively marked, and unmarked for the first level
        (positiveNotations, unmarkedNotations, negativeNotations) = levelOne()
        print "level", level

        rules=[]
        newRules=[]
        # Gather rules in one array
        for eachrule in positiveNotations:
                str1 = "<span>RULE - </span>" + "<b>Attribute: </b>" +
str(eachrule[2]).strip('[]').strip('()').strip(',')
                str2 = "<b>From-to: </b>" + str(eachrule[3]).strip('[]')
                str8 = "<b>Support: </b>" + str(eachrule[4]).strip('[]').strip('()').strip(',')
                str9 = "<b>Confidence: </b>" + str(eachrule[5]).strip('[]').strip('()').strip(',')
                str13 = str1 + '&nbsp&nbsp' + str2 + '&nbsp&nbsp' + str8 + '&nbsp&nbsp' + str9 + '<br />' + ","

                rules.append(str13)

        prevNumNotations = len(unmarkedNotations)
        saturationFlag=0
        # Get the positively and negatively marked, and unmarked for all next levels
        # Keep looping until there are no more unmarked notations or the unmarked notations are
saturated
        while (unmarkedNotations!=[]):
                level+=1
                print "level", level
                (positiveNotations, unmarkedNotations, negativeNotations) =
nextLevel(unmarkedNotations)
                # Gather rules in one array
                for eachrule in positiveNotations:
                        attributesRule = eachrule[2]
                        valueRule = eachrule[3]
                        supportRule = eachrule[4]
                        confidenceRule = eachrule[5]

                        # Filter repeating attributes which also has the same value change
                        repeatedAttr = [item for item, count in
collections.Counter(attributesRule).items() if count > 1]
                        for eachRepeated in repeatedAttr:
                                repeatedIndex = attributesRule.index(eachRepeated)
                                del attributesRule[repeatedIndex]
                                del valueRule[repeatedIndex]
                        str5 = "<span>RULE - </span>"+ "<b>Attribute: </b>" +
str(attributesRule).strip('[]').strip('()').strip(',')
                        str6 = "<b>From-to: </b>" + str(valueRule).strip('[]')
                        str10 = "<b>Support: </b>" + str(supportRule).strip('[]').strip('()').strip(',')
                        str11 = "<b>Confidence:</b> " + str(confidenceRule).strip('[]').strip('()').strip(',')
                        str12 = str5 + '&nbsp&nbsp' + str6 + '&nbsp&nbsp' + str10 + '&nbsp&nbsp' +
str11 + '<br />' + ","
                        rules.append(str12)

                if prevNumNotations<=len(unmarkedNotations):
                        if saturationFlag==0:
                                saturationFlag=1
                        else:
                                print "Saturated: the unmarked notations will stay the same forever"
                                break
                prevNumNotations = len(unmarkedNotations)


        for rule in rules:
                rul2 = str(rule)
                print rul2.strip('[]').strip('()')

        print "len(rules)", len(rules)

        return rules
        # The unmarked notations saturate and the while loop keeps repeating till infinity >> address
this case in the report
        # This happens when the support & confidence thresholds are too low like supth=5 &
confth=0.25, therefore, we made a saturation check
        # This case is avoided by stopping the loop when saturation happens using saturation flag and
the comparing the previous number of unmarked notations with the current one
```

**INDEX.HTML**

```
$def with (supp, conf, decisionTo, decisionFrom, rules)

<!--style attributes-->
<style>
        html, body {
                margin:0;
                padding:0;
        }

        body{
                background-color: #F5F0EB;
        }

        h1{
                color: white;
                padding-left: 30px;
                padding-top: 10px;
                font-family: 'Open Sans', sans-serif;
        }

        h2{
                color:  #302F34;
                padding-left: 30px;
                padding-top: 10px;
                font-family: 'Open Sans', sans-serif;
        }

        .h3{
                color: #E1E3E4;
                padding-left: 20px;
                font-family: 'Open Sans', sans-serif;
        }

        .h2{
                color: #E1E3E4;
                padding-left: 30px;
                width: 100%;
                font-family: 'Open Sans', sans-serif;
        }

        .top{
                background-color: #302F34;
                margin: 0 auto;
                width: 100%;
                height: 200px;
        }

        .leftHeader{
                width: 50%;
                float: left;
        }

        .rightHeader{
                width: 50%;
                float:left;
        }

        p{
                padding-left: 30px;
                padding-top: 10px;
                padding-bottom: 120px;
                font-family: 'Open Sans', sans-serif;
                white-space:normal;
        }
        span{
                color: #302F34;
                padding-left: 26px;
                font-family: 'Open Sans', sans-serif;
        }
        #footer{
                position:fixed;
                left:0px;
                bottom:0px;
                height:100px;
                width:100%;
                background:#999;
        }
        #wrapper{
                width:100%;
        }

</style>

<!--linking google fonts-->
<link href='https://fonts.googleapis.com/css?family=Open+Sans:400,800' rel='stylesheet' type='text/
css'>

<html>
        <head>
                <title>ITCS 8162</title>
                <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>
        </head>

<body >

<div class="top">
        <h1>D A T A &nbsp; I N P U T S </h1>
        <!--form for input values-->
```

```
	<form>
		<div class ="leftHeader">
			<div class = "h2">	<div class id ="wrapper">Minimum Support:
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<input type="text" name="minSup"
style="width: 60%; background-color: #E1E3E4;" placeholder="Whole # or decimal" > </
div>&nbsp;&nbsp;&nbsp;&nbsp;
								<div class id ="wrapper">Minimum
Confidence: &nbsp;&nbsp;<input type="text" name="minConf" style="width: 60%;background-color:
#E1E3E4; " placeholder="Whole # or decimal"> </div>
			</div>
		</div>
		<div class="rightHeader">
			<div class = "h2">Map To:
			<select name="mapTo">
				<option value="Alert" >Alert</option>
				<option value="Warning" >Warning</option>
				<option value="Stable">Stable</option>
				<option value="Sustainable">Sustainable</option>
			</select>
			&nbsp;&nbsp;&nbsp;&nbsp;
			From:
			<select name="mapFrom">
				<option value="Sustainable" >Sustainable</option>
				<option value="Alert" >Alert</option>
				<option value="Warning" >Warning</option>
				<option value="Stable" >Stable</option>

			</select></div><br>
			<input onclick = "string_sanitize($rules)" type="submit" value="Submit" style=
"float:right; margin-right: 20%;width: 100px;">
		</div>
	</form>
</div>
<h2>A C T I O N  &nbsp; R U L E S &nbsp; G E N E R A T E D :</h2>
<p id="P_action_rules">
	Threshold Support: $supp <br>
	Threshold Confidence: $conf <br>
	Decistion to: $decisionTo <br>
	Decision from: $decisionFrom <br>
$if rules:
	Rules: <br>
	<em id="action_rules" style="color: green; font-size: 1em;">  <br> $:rules<br></em><br>
	<!--Rules Generated to go here-->
	<!--  <button onclick="myFunction()">Try it</button> -->
$else:
	<em style="color: green; font-size: 1em;">No Rules matching your desired requirements.
Please make sure you adhered to the following guidelines: <br>
		1) Your confidence and support is in whole number or decimal format. FRACTIONS
ARE NOT ACCEPTED. <br>
```

```
                    2) Your confidence is between 0 and 1, with 0 being 0 percent and 1 being 100 percent
<br>
                    3) There are no symbols or letters in your specifications. </em>



</p>
<div id ="footer"> <p> Omar Eltayeby, Alexander Hohl, Dongdong Li & Brianna Chen <br>
                              ITCS 8162, 2015: Action Rules Discovery Project</p>
</div>


</body>
</html>
```