Interesting Rule Induction Module: Adding Support for Unknown Attribute Values

By

Theodore Lindsey

Submitted to the Department of Electrical Engineering & Computer Science and the
Graduate Faculty of the University of Kansas
in partial fulfillment of the requirements for the degree of
Master of Science

Prof. Jerzy Grzymala-Busse, Chairperson

Committee members

Prof. Bo Luo

Prof. Prasad Kulkarni

Date defended: December 02, 2016

The Thesis Committee for Theodore Lindsey certifies
that this is the approved version of the following thesis :

Interesting Rule Induction Module: Adding Support for Unknown Attribute Values

Prof. Jerzy Grzymala-Busse, Chairperson

Date approved: December 02, 2016

Abstract

IRIM (Interesting Rule Induction Module) is a rule induction system designed to induce particularly strong, simple rule sets. Additionally, IRIM does not require prior discretization of numerical attribute values. IRIM does not necessarily produce consistent rules that fully describe the target concepts, however, the rules induced by IRIM often lead to novel revelations of hidden relationships in a dataset. In this paper, we attempt to extend the IRIM system to be able to handle missing attribute values (in particular, lost and do-not-care attribute values) more thoroughly than ignoring the cases that they belong to. Further, we include an implementation of IRIM in the modern programming language Python that has been written for easy inclusion in within a Python data mining package or library. The provided implementation makes use of the Pandas module which is built on top of a C back end for quick performance relative to the performance normally found with Python.

Acknowledgements

I would first like to thank my advisor, Prof. Jerzy Grzymala-Busse, for guiding and supporting me through this process and for his data mining classes (EECS 837, 839) which helped me to realize my interest in data science and helped greatly to prepare me for this paper.

I wish to thank Prof. Bo Luo and Prof. Prasad Kulkarni for serving on my thesis committee. Additionally, I would like to thank Prof. Bo Luo for his information retrieval course in which I learned a number of techniques that I was able to apply to more efficiently implementing IRIM.

I would also like to thank the KU EECS faculty as a whole for providing instruction and advice.

I wish to thank my undergraduate advisor Prof. Andrew Parker for his inspiration and mentorship as I worked on my bachelor's degree and since then.

I would like to thank Prof. Mark Yannotta for inspiring me to pursue higher education when I was uncertain of my academic interests and attempting to find an area of study for a 4-year degree.

Finally, I would like to thank my mother, my father, and my wife for their encouragement and support of my education. Both of my parents encouraged me to always explore the world and to discover how it works and they taught me how to push forward through difficulty. My wife has always stood by me and encouraged me to be the best that I can be.

Contents

1. *Introduction*

1.1 Introduction to Data Mining

In our world of data, it is not possible to sift through the vast quantities of data manually. As a description of the scale of the task, many refer to the situation as the data deluge. Further, it is often the case that even experts may not be fully aware of the intricate and unintuitive connections between aspects of a system. Understanding our data is a vital task. Clive Humby (responsible for the Tesco Clubcard loyalty card) coined the phrase "data is the new oil" [2]. On the surface, it is clear that he meant that data is valuable. But he went on to explain that, like oil, data comes in many forms. In the raw form (i.e. crude), there is not much use to be had. It is not until we clean, organize, break down, and refine our data that we may glean value from it.

Data mining aims to address this data deluge by programmatically discovering relationships and connections in data sets. To this end, techniques in data mining draw from machine learning, artificial intelligence, statistics, and data management fields. Data mining spans many types of problems such as classification, clustering, regression analysis, anomaly detection, association rules, supervised learning, and forecasting.

1.2 Introduction to Rule Induction

In this paper, we are interested in the rule induction side of data mining and classification. In rule induction, we would like to establish a set of rules that we can use to classify data. In our data, we have a set of cases or records. Each case is a single observation in which a number of attributes and their values were recorded. In addition, each case also contains an attribute known as a decision for which we would like to classify future. For rule induction, it is necessary that the decision be symbolic (discrete). Other attributes in the data set can be symbolic or numerical (continuous), although, numerical attributes will usually have to be discretized by some aspect of the rule induction algorithm.

If we consider the task of determining if a patient has a flu (in other words, classifying the patient as "Flu" or "No Flu"), then we can imagine the corresponding data set. The attributes in our data set could be symptoms associated with the flu such as the patient's temperature (a numerical attribute), if the patient is experiencing a headache (a binary attribute with values of "yes" and "no", for our purposes, though, it is treated just like any other symbolic attribute), and if the patient has coughed a lot recently (again, another binary attribute with the values "yes" or "no"). The decision in our case would be "Flu" and the possible values would be "yes" or "no." Our task, then, is to establish rules such as "if the patient has a temperature above 37 and is coughing, then they have the flu." We would then write the rule as $(\text{Temp}, > 37)\&(\text{Cough}, \text{yes}) \rightarrow (\text{Flu}, \text{yes})$. The following is an example of what a data set might look like:

	Attributes			Decision
	Temp	Headache	Cough	Flu
0	36	no	no	no
1	36	yes	yes	no
2	38	yes	no	no
3	38	yes	yes	yes
4	40	no	yes	yes
5	40	yes	yes	yes

On each row of the table we find a single case. The cells of that row show the values for each attribute and the value for the decision.

In this paper, we use the following syntax to describe different aspects of the rule induction process:

- (a, v) – This represents an attribute-value pair. In the above example, the attribute might be "Cough" and the value might be "yes" which would give us the attribute-value pair of $(\mathrm{Cough}, \mathrm{yes})$.

- $[(\mathrm{a}, \mathrm{v})]$ – This represents the block for a given attribute-value pair. A block is a set of all of the training cases described by the attribute-value pair. If we knew that in our training set, cases 1, 3, 4, and 5 were all of the cases that matched $(\mathrm{Cough}, \mathrm{yes})$, then we would say that $[(\mathrm{Cough}, \mathrm{yes})] = 1, 3, 4, 5$. This notation can also be applied to the decision and a value of the decision. Formally, this means that $[(\mathrm{a}, \mathrm{v})] = \{x | \mathrm{a}(x) = v\}$ where x is a case in our training data set.‘

- $(\mathrm{a}_1, \mathrm{v}_1) \& \ldots \& (\mathrm{a}_k, \mathrm{v}_k) \rightarrow (\mathrm{d}, \mathrm{d}_i)$ – This represents a rule that can be used for classification. Rules are composed of two components. The first component is shown on the left hand side of the arrow. The left hand side is a list of conditions. The right hand side shows what value

of the decision should be assigned to the case when classifying and the conditions on the left hand side are met.

- $[(a_1, v_1)] \cap \cdots \cap [(a_k, v_k)]$ and $[(d, d_i)]$ – While rules are still being induced, we often consider these two sets. The intersection of all the (a_i, v_i) blocks is the set of training cases described by the left hand side of the rule. The decision block represents the "concept" or set of all training cases that have the value v_k for their decision. These two sets are compared to determine how well our rule performed in training.

We start with a data set called a training set that has already been classified by decision value. Generally, we expect that the classification of the training set has been performed by a domain expert. Occasionally, it is possible that a domain expert is not available to classify the training set so an approach such as cluster analysis could be attempted (this is more likely to lead to an inconsistent training set, however). Once we have a classified training set, we attempt to find sets of attribute-value pairs that lead to our target decision value. To fully replicate the classification of our training set, we need sets of rules that classify each value of the decision.

Depending on our motivation for performing rule induction, we can place various requirements on the rules we induce. If we start with a consistent training set, it could make sense to require that the induced rules also be consistent (that they correctly classify all training cases). If our training set is not consistent, we would like to at least guarantee that rules we keep meet a minimum quality threshold. For instance, we could require that for all rules we keep, the conditional probability of a case belonging to the concept given that it is covered by the rule is higher than a specified value.

2. *Interesting Rule Induction Module*

2.1 Introduction to IRIM

The Interesting Rule Induction Module (or IRIM) was introduced by Dr. Jerzy Grzymala-Busse, Jay Hamilton, and Dr. Zdzislaw Hippe in [3] and [4]. In [3], the authors compare the results and performance of the LEM2 rule induction algorithm to the results and performance of IRIM. In [4], the authors apply IRIM to the task of detecting Melanoma and comparing the rules induced by IRIM to those induced by LEM2.s

Most rule induction systems aim to ensure that all cases are covered or to ensure that all rules are fully consistent. One common result of these goals is that a complete rule set from a rule induction system will have rules with high specificity (large numbers of attribute-value conditions). Another common result is that many or all of the rules induced will have low strength (low numbers of correct cases covered) relative to the size of the training data set.

In IRIM, low limits are placed on the maximum specificity of rules (the number of attribute-value conditions) in order to produce very short rules (often rules address between two and four attributes). The result of requiring short rules is that many of the induced rules are quite strong when compared to rules produced with more standard rule induction modules. While we aim for strong rules, we do not want to lose sight of quality, either. Towards this end, we require that rules maintain a minimum quality, as measured by the conditional probability that a case is correctly classified in training by the rule (given that it is covered by the rule).

2.2 Pseudocode and an Example

The general approach used by IRIM is broken down into three main parts. First, compute blocks of cases for all attribute-value pairs. Second, determine all possible sets of conditions given the minimum and maximum rule length specified by the user. The third step breaks down into several sub steps but can be boiled down into the general goal: from the set of all possible sets of conditions, choose rules that adhere to the other specifications set for by the user. Make sure all all rules meet the minimum coverage requirement and, implicitly, the minimum strength requirement. Figure out which decision value the rule gives by comparing the intersection of the rule's block of cases with the blocks of cases from all the values of the decision. The decision value corresponding to whichever resulting block is largest is the decision value given by the the rule. Once we know which decision value the rule corresponds to, we can also ensure the rule meets the minimum probability and minimum strength requirements. Any rules meeting the requirements can be returned to the user. Ideally, they should be sorted in order of strength from strongest to least strong.

The following pseudocode illustrates the process

```
for each a_i in attributes:
if a_i is numerical:
a_i.cutpoints = []
sort a_i.values
for v_k, v_(k+1) in a_i.values:
a_i.cutpoints.append((v_k + v_(k_1)) / 2)

for each c_k in a_i.cutpoints:
# note: "<c_k" and ">=c_k" are symbolic values
# representing case of a_i that lie on each side of the cutpoint
compute block [(a_i, "<c_k")]
compute block [(a_i, ">=c_k")]

if a_i is symbolic:
for each v_k in a_i.values:
```

```
    compute block [(a_i, v_k)]

    attributeCombinations = []
    possibleRules = []
    rules = []

    for all l in the interval [minLength, maxLenght]:
    add all possible l-long combinations of attributes to attributeCombinations

    for all c in attributeCombinations:
    add all combinations of attribute-value pairs given the attributes in c to possibleRules

    for all r in possibleRules:
    r.coverage = intersection([(a,v)] for a,v in r)

    conceptCoverages = []
    for all v in decision:
    block = union(r.coverage, [(d,v)])
    conceptCoverages.append((v, block))
    bestDecision = max(conceptCoverages, key=len(block))

    r.rhs = (decision, bestDecision[0])

    r.strength = len(bestDecision[1])

    if r.strength > minStrength and r.strength / len(r.coverage) > minRatio:
    rules.append(r)
```

For our example, consider running IRIM over the following data set:

	Temp	Headache	Cough	Flu
0	36	no	no	no
1	36	yes	yes	no
2	38	yes	no	no
3	38	yes	yes	yes
4	40	no	yes	yes
5	40	yes	yes	yes

We first begin by computing the attribute-value blocks for the attribute Temp. Temp is numerical, so we will compute cut points and then build blocks of cases above and below the cut points. The sorted list of values is

$$[36, 38, 40]$$

Which gives us the cut points 37 and 39. From here, we build two blocks per cut point:

$$[(\text{Temp}, <= 37)] = \{0, 1\}$$
$$[(\text{Temp}, > 37)] = \{2, 3, 4, 5\}$$
$$[(\text{Temp}, <= 39)] = \{0, 1, 2, 3\}$$
$$[(\text{Temp}, > 39)] = \{4, 5\}$$

Now, we can work on blocks for Headache. We do not need to determine cut points since Headache is a symbolic attribute, so we get the following blocks

$$[(\text{Headache}, \text{yes})] = \{1, 2, 3, 5\}$$
$$[(\text{Headache}, \text{no})] = \{0, 4\}$$

Last, we compute the blocks for Cough. Again, Cough is a symbolic attribute, so we do not need

to worry about cut points.

$$[(\text{Cough}, \text{yes})] = \{1,3,4,5\}$$

$$[(\text{Cough}, \text{no})] = \{0,2\}$$

Now, in order to evaluate the strength of our rules, we need to be able to compare them to blocks of decision's values. Those blocks are

$$[(\text{Flu}, \text{no})] = \{0,1,2\}$$

$$[(\text{Flu}, \text{yes})] = \{3,4,5\}$$

With all of the necessary blocks established, we now consider all combinations of attribute-value pairs. In order to limit our scope, we also should apply a few conditions to our calculations:

Min Pr(concept|rule)	0.9
Min Length	2
Max length	2
Min Strength	2

Because we chose a max and min length of 2, we only need to consider rules of length 2. If we choose to involve the attribute Temp in our rule construction, we have four value choices to pick from. If we choose to involve the attributes Headache or Cough in our rule construction, we have two value choices to pick from in either case. We should therefore observe a total of $(4 \times 2) + (4 \times 2) + (2 \times 2) = 20$ possible rules. For now, we will only consider the left hand side. At the same time, we will compute the composite blocks:

$$[(\text{Temp}, <= 37)] \cap [(\text{Headache}, \text{yes})] = \{1\}$$

$$[(\text{Temp}, > 37)] \cap [(\text{Headache}, \text{yes})] = \{2,3,5\}$$

$$[(\text{Temp}, <= 39)] \cap [(\text{Headache}, \text{yes})] = \{1,2,3\}$$

$$[(\text{Temp}, > 39)] \cap [(\text{Headache}, \text{yes})] = \{5\}$$

$$[(\text{Temp}, <= 37)] \cap [(\text{Headache}, \text{no})] = \{0\}$$

$$[(\text{Temp}, > 37)] \cap [(\text{Headache}, \text{no})] = \{4\}$$

$$[(\text{Temp}, <= 39)] \cap [(\text{Headache}, \text{no})] = \{0\}$$

$$[(\text{Temp}, > 39)] \cap [(\text{Headache}, \text{no})] = \{4\}$$

$$[(\text{Temp}, <= 37)] \cap [(\text{Cough}, \text{yes})] = \{1\}$$

$$[(\text{Temp}, > 37)] \cap [(\text{Cough}, \text{yes})] = \{3,4,5\}$$

$$[(\text{Temp}, <= 39)] \cap [(\text{Cough}, \text{yes})] = \{1,3\}$$

$$[(\text{Temp}, > 39)] \cap [(\text{Cough}, \text{yes})] = \{4,5\}$$

$$[(\text{Temp}, <= 37)] \cap [(\text{Cough}, \text{no})] = \{0\}$$

$$[(\text{Temp}, > 37)] \cap [(\text{Cough}, \text{no})] = \{2\}$$

$$[(\text{Temp}, <= 39)] \cap [(\text{Cough}, \text{no})] = \{0,2\}$$

$$[(\text{Temp}, > 39)] \cap [(\text{Cough}, \text{no})] = \{\}$$

$$[(\text{Headache}, \text{yes})] \cap [(\text{Cough}, \text{yes})] = \{1,3,5\}$$

$$[(\text{Headache}, \text{no})] \cap [(\text{Cough}, \text{yes})] = \{4\}$$

$$[(\text{Headache}, \text{yes})] \cap [(\text{Cough}, \text{no})] = \{2\}$$

$$[(\text{Headache}, \text{no})] \cap [(\text{Cough}, \text{no})] = \{0\}$$

Of the above rules, we want a minimum support of two so we know that we can throw out all

rules that cover a single case or fewer. That leaves us with the following left hand sides

$$[(\text{Temp}, > 37)] \cap [(\text{Headache}, \text{yes})] = \{2,3,5\}$$

$$[(\text{Temp}, <= 39)] \cap [(\text{Headache}, \text{yes})] = \{1,2,3\}$$

$$[(\text{Temp}, > 37)] \cap [(\text{Cough}, \text{yes})] = \{3,4,5\}$$

$$[(\text{Temp}, <= 39)] \cap [(\text{Cough}, \text{yes})] = \{1,3\}$$

$$[(\text{Temp}, > 39)] \cap [(\text{Cough}, \text{yes})] = \{4,5\}$$

$$[(\text{Temp}, <= 39)] \cap [(\text{Cough}, \text{no})] = \{0,2\}$$

$$[(\text{Headache}, \text{yes})] \cap [(\text{Cough}, \text{yes})] = \{1,3,5\}$$

Now, in order to calculate the actual strength, we need to know which decision concept the rule gives. To determine this, we consider the intersection of the block given by the rule with each of the concept blocks of the decision. Whichever intersection produces the largest resulting set will be the decision value that corresponds with the rule. After applying that process to our data, we are left with the following. Note that the intersection of the blocks of the left hand side and the right hand side is shown. In other words, we see the sets of correctly covered cases for each rules.

$$(\text{Temp}, > 37)\&(\text{Headache}, \text{yes}) \rightarrow (\text{Flu}, \text{yes}) : \{3,5\}$$

$$(\text{Temp}, <= 39)\&(\text{Headache}, \text{yes}) \rightarrow (\text{Flu}, \text{no}) : \{1,2\}$$

$$(\text{Temp}, > 37)\&(\text{Cough}, \text{yes}) \rightarrow (\text{Flu}, \text{yes}) : \{3,4,5\}$$

$$(\text{Temp}, <= 39)\&(\text{Cough}, \text{yes}) \rightarrow (\text{Flu}, \text{no}) : \{1\}$$

$$(\text{Temp}, > 39)\&(\text{Cough}, \text{yes}) \rightarrow (\text{Flu}, \text{yes}) : \{4,5\}$$

$$(\text{Temp}, <= 39)\&(\text{Cough}, \text{no}) \rightarrow (\text{Flu}, \text{no}) : \{0,2\}$$

$$(\text{Headache}, \text{yes})\&(\text{Cough}, \text{yes}) \rightarrow (\text{Flu}, \text{yes}) : \{3,5\}$$

To now apply the strength requirement, we can ignore any rules that correctly cover only one case, leaving us with the following rules (again, we show the sets of correctly covered cases):

$$(\text{Temp}, > 37)\&(\text{Headache}, \text{yes}) \rightarrow (\text{Flu}, \text{yes}) : \{3,5\}$$

$$(\text{Temp}, <= 39)\&(\text{Headache}, \text{yes}) \rightarrow (\text{Flu}, \text{no}) : \{1,2\}$$

$$(\text{Temp}, > 37)\&(\text{Cough}, \text{yes}) \rightarrow (\text{Flu}, \text{yes}) : \{3,4,5\}$$

$$(\text{Temp}, > 39)\&(\text{Cough}, \text{yes}) \rightarrow (\text{Flu}, \text{yes}) : \{4,5\}$$

$$(\text{Temp}, <= 39)\&(\text{Cough}, \text{no}) \rightarrow (\text{Flu}, \text{no}) : \{0,2\}$$

$$(\text{Headache}, \text{yes})\&(\text{Cough}, \text{yes}) \rightarrow (\text{Flu}, \text{yes}) : \{3,5\}$$

The second restriction we placed on our rules is that we only want rules in which the probability of covering the specified decision given that the left hand side is satisfied is above 90%. To calculate this, we just consider the size of the correctly covered sets divided by the size of the set of

all cases covered by the rule. In the process, we eliminate all rules that do not meet the minimum probability requirement to get the following three rules.

$$(\text{Temp}, > 37)\&(\text{Cough}, \text{yes}) \rightarrow (\text{Flu}, \text{yes}): \quad \Pr(\text{rhs}|\text{lhs}) = 1$$

Covers: $\{3,4,5\}$

Strength: 3

Specicifity: 2

$$(\text{Temp}, > 39)\&(\text{Cough}, \text{yes}) \rightarrow (\text{Flu}, \text{yes}): \quad \Pr(\text{rhs}|\text{lhs}) = 1$$

Covers: $\{4,5\}$

Strength: 2

Specicifity: 2

$$(\text{Temp}, <= 39)\&(\text{Cough}, \text{no}) \rightarrow (\text{Flu}, \text{no}): \quad \Pr(\text{rhs}|\text{lhs}) = 1$$

Covers: $\{0,2\}$

Strength: 2

Specicifity: 2

2.3 Dealing with Missing Attribute Values

Sometimes in data mining we encounter data sets with incomplete information. [5, Ch. 3]

Attribute values may be missing because they were deemed irrelevant when they were recorded. This might occur if a conclusion were reached and the case was classified by a domain expert prior to reaching a certain test, in which case the result of the test does not impact the classification of the case. This would be classified as a "do not care" missing value and are designated with a "*" symbol.

Attribute values may be missing because they were not recorded or got lost even though they are still important. This could occur if the value were accidentally removed or the data was corrupted or a respondent refused to answer that question. Attributes falling into this category are referred to as "lost" values and are designated with a "?" symbol.

A third missing value type which we will not address in this paper is the "concept" missing attribute value. This missing value is designated with a "-" and usually takes on the value most common value of the attribute when restricting our consideration to cases sharing the same decision value (belonging to the same concept) [1].

For cases that have some missing attribute values, we assume that they have a least one attribute which does not have a missing value. Cases missing all attribute values provide us with no information and can therefore be discarded [6].

As was discussed in [1] as an approach for addressing missing attribute values, we will consider, independently, what we would like to do with "*" and "?":

- For "do not care", it was decided that for this case, the missing attribute value did not matter. Our approach, then, will be to assign all possible attribute values for these sorts of missing values. In [5], this is approached by removing the case in question and replacing it with a set of cases in which, for each element of the set, the "*" attribute value has been replaced with one of the other attribute values observed in the data set.

- For lost attribute values, the original values was probably important to the case. In this event, rather than replacing the value with all possible values, we will instead not consider this record when building blocks with the attribute that had the lost value. We do not want to incorrectly influence the rule induction process when it comes to this attribute and value since we assume that it was important.

In order to implement the "do not care" portion, it is sufficient to approach building our attribute-value blocks slightly differently. Normally, the attribute-value blocks are simply the set of all training cases for which all records share the same attribute value. We will start with this base,

but then, we will add the block $[(\mathrm{a}, \text{"*"})]$. In other words, for any attributes with a "do not care" value, blocks for that attribute will be defined as $[(\mathrm{a}, \mathrm{v}_k)] = \{x | \mathrm{a}(x) = \mathrm{v}_k\} \cup \{x | \mathrm{a}(x) = \text{"*"}\}$. Note that this means that if the attribute a has no "do not care" attribute values, $[(\mathrm{a}, \mathrm{v}_k)] = \{x | \mathrm{a}(x) = \mathrm{v}_k\}$ which is the traditional definition.

Consider the previously used data set in which a few attribute values are missing:

	Temp	Headache	Cough	Flu
0	36	no	no	no
1	36	yes	*	no
2	38	-	no	no
3	38	yes	?	yes
4	40	no	yes	yes
5	40	*	yes	yes

Again, we go through the same process of computing cut points for Temp and then building

the blocks for each attribute-value pair. This gives us:

$$[(\text{Temp}, <= 37)] = \{0,1\}$$

$$[(\text{Temp}, > 37)] = \{2,3,4,5\}$$

$$[(\text{Temp}, <= 39)] = \{0,1,2,3\}$$

$$[(\text{Temp}, > 39)] = \{4,5\}$$

$$[(\text{Headache}, \text{yes})] = \{1,3\}$$

$$[(\text{Headache}, \text{no})] = \{0,4\}$$

$$[(\text{Cough}, \text{yes})] = \{4,5\}$$

$$[(\text{Cough}, \text{no})] = \{0,2\}$$

and

$$[(\text{Flu}, \text{no})] = \{0,1,2\}$$

$$[(\text{Flu}, \text{yes})] = \{3,4,5\}$$

Now, we must also compute the blocks for the value "*" for each attribute:

$$[(\text{Temp}, \text{"*"})] = \{\}$$

$$[(\text{Headache}, \text{"*"})] = \{5\}$$

$$[(\text{Cough}, \text{"*"})] = \{1\}$$

Now, for each attribute-value pair's block $[(a_i, v_k)]$, we will add the set $[(a_i, "*")]$ to it to get:

$$[(\text{Temp}, <= 37)] = \{0,1\}$$

$$[(\text{Temp}, > 37)] = \{2,3,4,5\}$$

$$[(\text{Temp}, <= 39)] = \{0,1,2,3\}$$

$$[(\text{Temp}, > 39)] = \{4,5\}$$

$$[(\text{Headache}, \text{yes})] = \{1,3,5\}$$

$$[(\text{Headache}, \text{no})] = \{0,4,5\}$$

$$[(\text{Cough}, \text{yes})] = \{1,4,5\}$$

$$[(\text{Cough}, \text{no})] = \{0,1,2\}$$

With the blocks established, we now consider the parameters we will use for IRIM. We will re-use the same parameters as with the last example:

Min Pr(concept|rule)	0.9
Min Length	2
Max length	2
Min Strength	2

For now, we will only consider the left hand side. At the same time, we will compute the composite

blocks:

$$[(\text{Temp}, \text{<=37})] \cap [(\text{Headache}, \text{yes})] = \{1\}$$

$$[(\text{Temp}, \text{>37})] \cap [(\text{Headache}, \text{yes})] = \{3,5\}$$

$$[(\text{Temp}, \text{<=39})] \cap [(\text{Headache}, \text{yes})] = \{1,3\}$$

$$[(\text{Temp}, \text{>39})] \cap [(\text{Headache}, \text{yes})] = \{5\}$$

$$[(\text{Temp}, \text{<=37})] \cap [(\text{Headache}, \text{no})] = \{0\}$$

$$[(\text{Temp}, \text{>37})] \cap [(\text{Headache}, \text{no})] = \{4,5\}$$

$$[(\text{Temp}, \text{<=39})] \cap [(\text{Headache}, \text{no})] = \{0\}$$

$$[(\text{Temp}, \text{>39})] \cap [(\text{Headache}, \text{no})] = \{4,5\}$$

$$[(\text{Temp}, \text{<=37})] \cap [(\text{Cough}, \text{yes})] = \{1\}$$

$$[(\text{Temp}, \text{>37})] \cap [(\text{Cough}, \text{yes})] = \{4,5\}$$

$$[(\text{Temp}, \text{<=39})] \cap [(\text{Cough}, \text{yes})] = \{1\}$$

$$[(\text{Temp}, \text{>39})] \cap [(\text{Cough}, \text{yes})] = \{4,5\}$$

$$[(\text{Temp}, \text{<=37})] \cap [(\text{Cough}, \text{no})] = \{0,1\}$$

$$[(\text{Temp}, \text{>37})] \cap [(\text{Cough}, \text{no})] = \{2\}$$

$$[(\text{Temp}, \text{<=39})] \cap [(\text{Cough}, \text{no})] = \{0,1,2\}$$

$$[(\text{Temp}, \text{>39})] \cap [(\text{Cough}, \text{no})] = \{\}$$

$$[(\text{Headache}, \text{yes})] \cap [(\text{Cough}, \text{yes})] = \{1,5\}$$

$$[(\text{Headache}, \text{no})] \cap [(\text{Cough}, \text{yes})] = \{4,5\}$$

$$[(\text{Headache}, \text{yes})] \cap [(\text{Cough}, \text{no})] = \{1\}$$

$$[(\text{Headache}, \text{no})] \cap [(\text{Cough}, \text{no})] = \{0\}$$

Again, we remove any blocks that contain fewer than 2 elements as they can not give a rule with

strength of 2.

$$[(\text{Temp}, \text{>37})] \cap [(\text{Headache}, \text{yes})] = \{3,5\}$$

$$[(\text{Temp}, \text{<=39})] \cap [(\text{Headache}, \text{yes})] = \{1,3\}$$

$$[(\text{Temp}, \text{>37})] \cap [(\text{Headache}, \text{no})] = \{4,5\}$$

$$[(\text{Temp}, \text{>39})] \cap [(\text{Headache}, \text{no})] = \{4,5\}$$

$$[(\text{Temp}, \text{>37})] \cap [(\text{Cough}, \text{yes})] = \{4,5\}$$

$$[(\text{Temp}, \text{>39})] \cap [(\text{Cough}, \text{yes})] = \{4,5\}$$

$$[(\text{Temp}, \text{<=37})] \cap [(\text{Cough}, \text{no})] = \{0,1\}$$

$$[(\text{Temp}, \text{<=39})] \cap [(\text{Cough}, \text{no})] = \{0,1,2\}$$

$$[(\text{Headache}, \text{yes})] \cap [(\text{Cough}, \text{yes})] = \{1,5\}$$

$$[(\text{Headache}, \text{no})] \cap [(\text{Cough}, \text{yes})] = \{4,5\}$$

And now, we determine what the decision should be given the overlap of the blocks above and

blocks of decision values. At the same time, we will only keeps rules with a support of 2 or better:

$$(\text{Temp}, >37)\&(\text{Headache}, \text{yes}) \rightarrow (\text{Flu}, \text{yes}) : \{3,5\}$$

$$(\text{Temp}, >37)\&(\text{Headache}, \text{no}) \rightarrow (\text{Flu}, \text{yes}) : \{4,5\}$$

$$(\text{Temp}, >39)\&(\text{Headache}, \text{no}) \rightarrow (\text{Flu}, \text{yes}) : \{4,5\}$$

$$(\text{Temp}, >37)\&(\text{Cough}, \text{yes}) \rightarrow (\text{Flu}, \text{yes}) : \{4,5\}$$

$$(\text{Temp}, >39)\&(\text{Cough}, \text{yes}) \rightarrow (\text{Flu}, \text{yes}) : \{4,5\}$$

$$(\text{Temp}, <=37)\&(\text{Cough}, \text{no}) \rightarrow (\text{Flu}, \text{no}) : \{0,1\}$$

$$(\text{Temp}, <=39)\&(\text{Cough}, \text{no}) \rightarrow (\text{Flu}, \text{no}) : \{0,1,2\}$$

$$(\text{Headache}, \text{no})\&(\text{Cough}, \text{yes}) \rightarrow (\text{Flu}, \text{yes}) : \{4,5\}$$

Next, we will discard all rules that don't meet the minimum ratio requirement, leaving us with:

$$(\text{Temp}, >37)\&(\text{Headache}, \text{yes}) \rightarrow (\text{Flu}, \text{yes}): \quad \Pr(\text{rhs}|\text{lhs}) = 1$$

Covers: $\{3,5\}$

Strength: 2

Specicifity: 2

$$(\text{Temp}, >37)\&(\text{Headache}, \text{no}) \rightarrow (\text{Flu}, \text{yes}): \quad \Pr(\text{rhs}|\text{lhs}) = 1$$

Covers: $\{4,5\}$

Strength: 2

Specicifity: 2

$$(\text{Temp}, >39)\&(\text{Headache}, \text{no}) \rightarrow (\text{Flu}, \text{yes}): \quad \Pr(\text{rhs}|\text{lhs}) = 1$$

Covers: $\{4,5\}$

Strength: 2

Specicifity: 2

$$(\text{Temp}, >37)\&(\text{Cough}, \text{yes}) \rightarrow (\text{Flu}, \text{yes}): \quad \Pr(\text{rhs}|\text{lhs}) = 1$$

Covers: $\{4,5\}$

Strength: 2

Specicifity: 2

$$(\text{Temp}, >39)\&(\text{Cough}, \text{yes}) \rightarrow (\text{Flu}, \text{yes}): \quad \Pr(\text{rhs}|\text{lhs}) = 1$$

Covers: $\{4,5\}$

Strength: 2

Specicifity: 2

$$(\text{Temp}, <=37)\&(\text{Cough}, \text{no}) \rightarrow (\text{Flu}, \text{no}): \quad \Pr(\text{rhs}|\text{lhs}) = 1$$

Covers: $\{0,1\}$

Strength: 2

Specicifity: 2

$$(\text{Temp}, <=39)\&(\text{Cough}, \text{no}) \rightarrow (\text{Flu}, \text{no}): \quad \Pr(\text{rhs}|\text{lhs}) = 1$$

Covers: $\{0,1,2\}$

Strength: 3

Specicifity: 2

$$(\text{Headache}, \text{no})\&(\text{Cough}, \text{yes}) \rightarrow (\text{Flu}, \text{yes}): \quad \Pr(\text{rhs}|\text{lhs}) = 1$$

Covers: $\{4,5\}$

Strength: 2

Specicifity: 2

2.4 Performance Analysis

IRIM has the potential to perform very poorly. The main factors that contribute to this are the number of attributes and, to a lesser degree, the number of numerical attributes. The performance hit from large numbers of attributes is because the algorithm calls for finding all possible combinations for rules of specified lengths. The performance hit from many numerical attributes is because numerical attributes tend to have very many unique values and thus, very many cut points

which will then produce two discretized values. This leaves many possible combinations for rules. As a result, IRIM's worst case time complexity is exponential with respect to n for inputs with n attributes.

The modifications made to IRIM to add support for missing attribute values does not increase the complexity per attribute since no deeper looping is needed per attribute. Additionally, for data sets that do not have any missing attribute values, the logic for the unmodified IRIM and the modified IRIM are completely identical. This modification has adds no significant impact to IRIM's performance while adding support for a wider variety of input datasets.

2.5 Future Investigations

Several modification to the included implementation of IRIM would add a degree of user-friendliness and utility to the algorithm. As it stands, IRIM returns all rules satisfying the minimum strength, minimum and maximum specificity, and minimum ratio requirements. As a result, IRIM ends up generating rules which overlap. As an example, in the example with no missing values, we generated the rules $(\text{Temp}, > 37)\&(\text{Cough}, \text{yes}) \rightarrow (\text{Flu}, \text{yes})$ and $(\text{Temp}, > 39)\&(\text{Cough}, \text{yes}) \rightarrow (\text{Flu}, \text{yes})$. If it is determined that the first rule is satisfactory, then the second rule is both redundant and also more limiting than necessary given that $[(\text{Temp}, > 39)] \subseteq [(\text{Temp}, > 37)]$. As such, it would be prudent to modify IRIM to not return rules that are more restrictive versions of other rules that were generated at the same time. Similarly, adding support for ignoring rules that are identical to other returned rules but have added extra conditions would follow nicely.

Additionally, using the "most common value of attribute restricted to concept" rather than disregarding that case for cases that have concept-based missing attribute values in the next logical step after adding support for lost and "do not care" attribute values. With support for concept-based missing attribute values, all three types of missing attribute values will be handled.

2.6 Concluding Remarks

The work reported in this thesis has introduced the reader to rule induction and, in particular, to the process of inducing rules using the Interesting Rule Induction Model. Further, we have explored an approach for incorporating missing attribute values when inducing rules rather than simply discarding them. Next, we discussed the impact that accommodating missing attribute values would have on the performance of the IRIM algorithm. Finally, we considered a few additional extensions to make to IRIM in order to clean up the induced rules and to possibly improve the induced rules by incorporating a recovery scheme for concept missing attribute values.

References

[1] Grzymala-Busse, J. (2015). EECS 839: Mining special data. Lecture Notes.

[2] Humby, C. (2006). The "New Oil" of the 21st century. Panel Discussion, ANA Marketing Maestros, Kellogg School. http://ana.blogs.com/maestros/2006/11/data_is_the_new.html.

[3] Jerzy Grzymala-Busse, Witold J. Grzymala-Busse, J. H. (2005). *Discriminant versus Strong Rule Sets*, (pp. 67–76). Springer Berlin Heidelberg: Berlin, Heidelberg.

[4] Jerzy Grzymala-Busse, Jay Hamilton, Z. H. (2004). *Diagnosis of Melanoma Using IRIM, a Data Mining System*, (pp. 996–1001). Springer Berlin Heidelberg: Berlin, Heidelberg.

[5] Oded Maimon, L. R. (2005). *Data Mining and Knowledge Discovery Handbook*. Springer US.

[6] W. Grzymala-Busse, J. (2006). *Rough Set Strategies to Data with Missing Attribute Values*, (pp. 197–212). Springer Berlin Heidelberg: Berlin, Heidelberg.

A. *Python Code*

A.1 LERS Parsing

../IRIM/LERSdat.py

```
# Code written by Theodore Lindsey for his master's thesis.
# Fall 2016
# Liscensed under the GNU Lesser General Public License v3
import pandas as pd
from utilities import isnum, progressBar, fileLineCount

class LERSdat:
    def __init__(self, filename=None):
        self.attributes = []
        self.attributeTypes = []
        self.records = {}
        self.index = {}
        self.decision = ""
        self.decisions = {}
        self.blocks = {}

        if filename is not None:
            self.open(filename)


    def open(self, filename):
        self.attributes = []
        self.attributeTypes = []
        self.records = {}
        self.index = {}
        self.decision = ""
```

```
        self.decisions = {}
        self.blocks = {}

        self.parse(filename)
        self.computeAllBlocks()


    def parse(self, filename):
        # parses the file to extract the dataset
        # file must be in LERS format
        linecount = fileLineCount(filename)
        phase = "find attributes"

        file = open(filename, 'r')
        case = 0
        attribute = 0
        caseValues = []

        print("Loading from file {}".format(filename))
        for linecurrent, line in enumerate(file):
            print(progressBar((linecurrent+1)/linecount), end="\r")

            line = line.split()
            if not line:
                continue

            elif phase == "find attributes" and line[0].startswith("["):
                # this mode finds the attributes
                # attributes should be the second line in the file
                if line[-1].endswith("]"):
                    # if the line also ends the attributes, move to next step
                    self.attributes = line[1:-2]
                    self.decision = line[-2]

                    self.records = pd.DataFrame(columns = self.attributes + [self.decision])

                    # set up voting for numerical attribute types
                    self.attributeTypes = [0 for x in self.attributes]
                    phase = "read records"
                else:
                    # otherwise, keep reading attributes
```

```
68                  phase = "reading attributes"
69                  line = line[1:]
70
71              elif phase == "reading attributes":
72                  # keep reading attributes
73                  if line[-1].endswith("]"):
74                      # we found the end of the attribute list
75                      self.attributes += line[:-2]
76                      self.decision = line[-2]
77
78                      self.records = pd.DataFrame(columns = self.attributes + [self.decision])
79
80                      # set up voting for numerical attribute types
81                      self.attributeTypes = [0 for x in self.attributes]
82                      phase = "read records"
83                  else:
84                      # we're still looking for attributes
85                      self.attributes += line
86
87              elif phase == "read records":
88                  # we finished reading attributes and now we're reading values for cases
89                  for element in line:
90                      # TODO: (optimization) if isnum(element), element = float(element)
91
92                      if element.startswith("!"):
93                          # ingore the rest of the line.  it's a comment
94                          break
95
96
97                      if attribute + 1 > len(self.attributes):
98                          # we've found all the attributes for this case
99
100                         if self.decision not in self.index:
101                             self.index[self.decision] = {}
102                         if element not in self.index[self.decision]:
103                             self.index[self.decision][element] = []
104
105                         self.index[self.decision][element].append(case)
106
107                         self.decisions[case] = element
108
```

```
                        caseValues.append(element)
                        # vote on if the attribute is a number or not
                        if len(self.attributeTypes) < len(caseValues):
                            self.attributeTypes.append(int(isnum(element)))
                        else:
                            self.attributeTypes[len(caseValues) - 1] += int(isnum(element))

                        self.records.loc[case] = caseValues

                        case += 1
                        attribute = 0
                        caseValues = []
                        continue

                    caseValues.append(element)
                    # vote on if the attribute is a number or not
                    self.attributeTypes[len(caseValues) - 1] += int(isnum(element))

                    if self.attributes[attribute] not in self.index:
                        self.index[self.attributes[attribute]] = {}
                    if element not in self.index[self.attributes[attribute]]:
                        self.index[self.attributes[attribute]][element] = set([])
                    self.index[self.attributes[attribute]][element].add(case)

                    attribute += 1

        # determine if attribute was numerical
        # todo: (optimization) base numerical decision on what values are, not how many can convert to float

        numcases = len(self.records.columns)
        tempHeader = self.attributes + [self.decision]
        tempAttributeTypes = self.attributeTypes
        self.attributeTypes = {}

        for i in range(len(tempAttributeTypes)):
            if len(set(self.records[(self.attributes + [self.decision])[i]])) == 2:
                self.attributeTypes[tempHeader[i]] = "Binary"
            elif tempAttributeTypes[i] > numcases / 2:
                self.attributeTypes[tempHeader[i]] = "Numerical"
            else:
```

```
149                 self.attributeTypes[tempHeader[i]] = "Discrete"
150         print("\n")
151 
152 
153 
154     def computeAllBlocks(self):
155         for attr in (self.attributes + [self.decision]):
156             self.blocks[attr] = {}
157 
158         print("computing numerical blocks")
159         self.computeNumericalBlocks()
160         print("computing discrete blocks")
161         self.computeDiscreteBlocks()
162 
163         # add all the "do-not-care" cases to the blocks for that attribute
164         for attr in self.attributes:
165             if "*" in self.blocks[attr]:
166                 for val in self.blocks[attr]:
167                     if val not in ["*", "?", "-"]:
168                         self.blocks[attr][val] = self.blocks[attr][val].union(self.blocks[attr]["
    *"])
169 
170 
171     def computeNumericalBlocks(self):
172         total = len(self.attributes + [self.decision])
173         for idx, attribute in enumerate(self.attributes + [self.decision]):
174             print(progressBar((idx+1) / total), end="\r")
175             if self.attributeTypes[attribute] == "Numerical":
176                 # find all unique values
177                 values = sorted(set(self.records[attribute]).difference(set(["*", "?", "-"])),
    key=lambda item: float(item))
178 
179                 # compute the cutpoints for this attribute
180                 cutpoints = []
181                 for i in range(len(values)-1):
182                     cutpoints.append((float(values[i])+float(values[i+1])) / 2)
183 
184                 # for each cut point, compute the <= and the > blocks
185                 for cutpoint in cutpoints:
186                     self.blocks[attribute][">{}".format(cutpoint)] = set()
187                     self.blocks[attribute]["<={}".format(cutpoint)] = set()
```

```
                    for case in self.records.index:
                        if isnum(self.records[attribute][case]):
                            if float(self.records[attribute][case]) > cutpoint:
                                self.blocks[attribute][">{}".format(cutpoint)].add(case)
                            else:
                                self.blocks[attribute]["<={}".format(cutpoint)].add(case)

                # find the "do-not-care" block for numerical attributes, too!
                if "*" in self.index[attribute]:
                    self.blocks[attribute]["*"] = set(self.index[attribute]["*"])
        print()

    def computeDiscreteBlocks(self):
        total = len(self.attributes + [self.decision])
        for (idx, attribute) in enumerate(self.attributes + [self.decision]):
            print(progressBar((idx+1) / total), end="\r")
            if self.attributeTypes[attribute] != "Numerical":
                for value in self.index[attribute]:
                    self.blocks[attribute][value] = set(self.index[attribute][value])
        print()

    def __str__(self):
        # this class will display the index when it is printed
        retStr = ""
        retStr += "Decision:    '{}'\n".format(self.decision)
        retStr += "Attributes: {}\n".format(self.attributes)
        for attribute in self.attributes:
            retStr += str(attribute) + "\n"
            for value in self.index[attribute]:
                retStr += "\t{}: {}\n".format(value, self.index[attribute][value])

        retStr += "{}\n".format(self.decision)
        for value in self.index[self.decision]:
            retStr += "\t{}: {}\n".format(value,self.index[self.decision][value])

        return retStr


if __name__ == "__main__":
    o = LERSdat("../Inputs/jerzy1.txt")
    print(o.records)
```

```
229     print(o.attributes + [o.decision])
230     print(o.attributeTypes)
231     o.computeAllBlocks()
232 
233     for attr in o.attributes + [o.decision]:
234         print("{}:{}".format(attr,o.blocks[attr]))
235     #print(o)
```

A.2 Rule Class

../IRIM/Rule.py

```
# Code written by Theodore Lindsey for his master's thesis.
# Fall 2016
# Liscensed under the GNU Lesser General Public License v3
class Rule:
    def __init__(self, lhs):
        self.lhs = lhs
        self.rhs = None
        self.correct = None
        self.incorrect = None

    def __str__(self):
        return " & ".join([str(c) for c in self.lhs]) + " -> {}".format(self.rhs)

    def strength(self):
        return len(self.correct)

    def specificity(self):
        return len(self.lhs)

    def coverage(self):
        return self.correct.union(self.incorrect)

    def score(self):
        return self.strength() * self.specificity()

    def stats(self, concept):
        st = {}

        st["cov"] = self.coverage()
        st["cov#"] = len(st["cov"])

        st["cor"] = self.correct
        st["cor#"] = len(st["cor"])

        st["icor"] = self.incorrect
        st["icor#"] = len(st["icor"])

```

```
38          st["pr"] = st["cor#"] / st["cov#"]
39          st["sp"] = self.specificity()
40          st["sc"] = self.score()
41          return st
```

A.3 IRIM Implementation

../IRIM/IRIM.py

```
# Code written by Theodore Lindsey for his master's thesis.
# Fall 2016
# Liscensed under the GNU Lesser General Public License v3
from LERSdat import LERSdat
from Rule import Rule
import itertools
from utilities import union, intersection, isnum, progressBar, nCr

class IRIM:
    def __init__(self, filename=None):
        self.rules = None
        self.lers = LERSdat()
        if filename is not None:
            self.open(filename)


    def open(self, filename):
        self.rules = None
        self.lers.open(filename)

    def computeAllRules(self, **options):
        # establish option parameters
        minLen = options.get("minLen")
        if minLen is None:
            minLen = 2
        maxLen = options.get("maxLen")
        if maxLen is None:
            maxLen = minLen
        ratio = options.get("ratio")
        if ratio is None:
            ratio = 0.8
        minStr = options.get("minStr")
        if minStr is None:
            minStr = 1
        minCov = options.get("minCov")
        if minCov is None:
            minCov = 1
```

```
38
39          # don't let the max length get out of hand
40          maxLen = min(maxLen, len(self.lers.attributes))
41
42          # find all possible conditions (rules)
43          possibleRules = []
44          self.rules = []
45          for length in range(minLen, maxLen+1):
46              print("computing possible rules of length {}".format(length))
47              count = nCr(len(self.lers.attributes), length)
48              for idx, attributeSet in enumerate(itertools.combinations(self.lers.attributes,
    length)):
49                  print(progressBar((idx+1) / count), end="\r")
50                  #valueSet = [[val for val in self.blocks[attr].keys() if (len(self.blocks[attr][
    val])/len(self.lers.records) > ratio)] for attr in attributeSet]
51                  valueSet = [[val for val in self.lers.blocks[attr].keys() if val not in ["*", "?"
    , "-"]] for attr in attributeSet]
52
53                  for combination in itertools.product(*valueSet):
54                      possibleRules.append(Rule(list(zip(attributeSet, combination))))
55
56          print("\n\nThere are a total of {} candidate rules".format(len(possibleRules)))
57
58          # trim the list down to just actual rules
59          concepts = self.lers.index[self.lers.decision]
60          print("\nremoving out-of-spec rules")
61          for idx, rule in enumerate(possibleRules):
62              #print()
63              print(progressBar((idx+1) / len(possibleRules)), end="\r")
64
65              coverage = intersection([self.lers.blocks[attr][val] for attr,val in rule.lhs])
66
67              if len(coverage) < minCov and len(coverage) < minStr:
68                  continue
69
70              coverageRatios = self.coverageRatioCheck(coverage)
71              bestDecision = max(coverageRatios, key=lambda item: item[1])
72
73              concept = self.lers.index[self.lers.decision][bestDecision[0]]
74              rule.correct = bestDecision[2]
75
```

```
            if bestDecision[1] > ratio and rule.strength() >= minStr:
                rule.incorrect = coverage.difference(bestDecision[2])
                rule.rhs = (self.lers.decision, bestDecision[0])
                self.rules.append(rule)

        # sort the rules by strength
        self.rules.sort(key=lambda item: item.strength(), reverse=True)

        print()

        return self.rules

    def coverageRatioCheck(self, covered):
        coverageRatios = []

        #covered = intersection([self.lers.blocks[attr][val] for attr,val in lhs])

        for d in self.decisionValues():
            correct = intersection([covered, self.lers.blocks[self.lers.decision][d]])
            if len(covered) > 0:
                coverageRatios.append((d, len(correct) / len(covered), correct))
            else:
                coverageRatios.append((d, 0, set()))

        return coverageRatios

    def decisionValues(self):
        return [v for v in self.lers.index[self.lers.decision]]

    def evaluateRules(self, top=None, rules=None):
        if rules is None:
            if self.rules is not None:
                rules = self.rules
            else:
                return

        temp = []
        if top is None or top > len(rules)+1:
            top = len(rules)+1

        for r in rules[:top]:
```

```
            lhs, rhs = r.lhs, r.rhs
            lhsConcept = r.coverage

            rhsConcept = self.lers.blocks[rhs[0]][rhs[1]]

            stats = r.stats(rhsConcept)

            for s in stats:
                if s is set and len(s) > 10:
                    s = "too large to display"

            print(r)
            print("\tCoverage         ({}): {}".format(stats["cov#"], stats["cov"]))
            print("\tCorrect/Strength ({}): {}".format(stats["cor#"], stats["cor"]))
            print("\tIncorrect        ({}): {}".format(stats["icor#"], stats["icor"] if stats["
    icor#"] > 0 else "{}"))
            print("\tP(rhs | lhs) : {}".format(stats["pr"]))
            print("\tSpecificity  : {}".format(stats["sp"]))
            print("\tScore        : {}".format(stats["sc"]))

    def __str__(self):
        if self.rules is None:
            return "No rules computed"
        else:
            s = ""

            for r in self.rules:
                s += "{}\n".format(r)

            return s

    def __getitem__(self, key):
        return self.rules[key]

    def __iter__(self):
        return self.rules.__iter__()

if __name__ == "__main__":
    i = IRIM("./Inputs/jerzy1.txt")
    rules = i.computeAllRules(minLen=2, maxLen=2, ratio=0.8, minStr=1)
    i.evaluateRules(rules)
```

A.4 Interface and Runner

../IRIM/run.py

```
# Code written by Theodore Lindsey for his master's thesis.
# Fall 2016
# Liscensed under the GNU Lesser General Public License v3
from os import listdir
from os.path import isfile, join
from utilities import isnum
from math import ceil

# Python version check
from sys import version_info
if version_info < (3,3):
    print("\n\nWARNING: You need to use python version 3.4 or newer.\n\n")
    quit()

# Check for required packages
from pip import get_installed_distributions
# build list of installed packages
installed_packages = [package.project_name for package in get_installed_distributions()]
# list of required_packages
required_packages = ["pandas"]
# check if each required package is installed
for package in required_packages:
    if package not in installed_packages:
        print("\n\nWARNING: You need to install the '{}' package.".format(package))
        print("Try running:\n\tpip install {}\nfrom a privileged terminal".format(package))
        quit()

from IRIM import IRIM

path = "./Inputs"
irim = IRIM()
choice = ""
file = None

# establish default IRIM parameters
minLen = 2
maxLen = 2
```

```
ratio = 0.8
minStr = 1
minCov = 1

menu = ["Open file", "View blocks", "Compute rules", "View rules", "Evaluate rules", "Exit"]

while choice != str(1+menu.index("Exit")):
    print("====MENU====")
    # print out the currently open file and path
    if file is not None:
        print("".join(["=" for x in "file=''" + str(join(path, file))]))
        print("file='{}'".format(join(path, file)))
        print("".join(["=" for x in "file=''" + str(join(path, file))]))

    # print out the menu options
    for i, text in enumerate(menu):
        print("{}. {}".format(i+1, text))
    choice = input("> ")
    ##############
    # Open file #
    ##############
    if choice == str(1+menu.index("Open file")):
        files = [f for f in listdir(path) if isfile(join(path, f))]

        subchoice = ""
        while subchoice not in [str(i) for i in range(1,len(files)+1)]:
            for i, file in enumerate(files):
                print("{}. '{}'".format(i+1, file))
            subchoice = input("> ")
        irim.open(join(path, files[int(subchoice) - 1]))

        minStr = ceil(0.05 * len(irim.lers.records))
    ################
    # View blocks #
    ################
    elif choice == str(1+menu.index("View blocks")):
        for attr in irim.lers.blocks:
            for val in irim.lers.blocks[attr]:
                print("[({}, {})] = {}".format(attr, val, irim.lers.blocks[attr][val]))
    ##################
    # Compute rules #
```

```
    ################
    elif choice == str(1+menu.index("Compute rules")):
        if file is None:
            print("Please first open a file")
        else:
            subchoice = None
            while subchoice not in ["6", ""]:
                print("Data set size: {}".format(len(irim.lers.records)))
                print("Current parameters")
                print("1. Min rule length  : {}".format(minLen))
                print("2. Max rule length  : {}".format(maxLen))
                print("3. Ratio            : {}".format(ratio))
                print("4. Min rule strength: {}".format(minStr))
                print("5. Min rule coverage: {}".format(minCov))
                print("6. accept: Compute rules")
                subchoice = input("> ")
                if subchoice == "1":
                    minLen = int(input("New min rule length: "))
                elif subchoice == "2":
                    maxLen = int(input("New max rule length: "))
                elif subchoice == "3":
                    ratio = float(input("New ratio: "))
                elif subchoice == "4":
                    minStr = int(input("New min rule strength: "))
                elif subchoice == "5":
                    minCov = int(input("New min rule coverage: "))

            irim.computeAllRules(minLen=minLen, maxLen=maxLen, ratio=ratio, minStr=minStr, minCov
    =minCov)
    ###############
    # View rules #
    ###############
    elif choice == str(1+menu.index("View rules")):
        if irim.rules is None:
            print("Please first compute rules")
        else:
            subchoice = None
            while (not isnum(subchoice)) and (subchoice not in ["a", "A", ""]):
                subchoice = input("Would you like to see all rules (enter 'a')?\nor the top n
    rules where n=(enter a number)?\n> ")

```

```
118             if subchoice in ["a", "A", ""]:
119                 print(irim)
120             else:
121                 stop = int(subchoice)
122                 for rule in irim[:stop]:
123                     print(rule)
124     ##################
125     # Evaluate rules #
126     ##################
127     elif choice == str(1+menu.index("Evaluate rules")):
128         if irim.rules is None:
129             print("Please first compute rules")
130         else:
131             subchoice = None
132             while (not isnum(subchoice)) and (subchoice not in ["a", "A", ""]):
133                 subchoice = input("Would you like to evaluate all rules (enter 'a')?\nor the top
    n rules where n=(enter a number)?\n> ")
134 
135             if subchoice in ["a", "A", ""]:
136                 irim.evaluateRules()
137             else:
138                 stop = int(subchoice)
139                 irim.evaluateRules(stop)
140     print()
```

A.5 Utility Functions

../IRIM/utilities.py

```
# Code written by Theodore Lindsey for his master's thesis.
# Fall 2016
# Liscensed under the GNU Lesser General Public License v3
from math import floor, factorial

def union(sets):
    if len(sets) == 0:
        return set()
    elif len(sets) == 1:
        return sets[0]
    elif len(sets) == 2:
        return sets[0].union(sets[1])
    else:
        return union([sets[0].union(sets[1])] + sets[2:])

def intersection(sets):
    if len(sets) == 0:
        return set()
    elif len(sets) == 1:
        return sets[0]
    elif len(sets) == 2:
        return sets[0].intersection(sets[1])
    else:
        return intersection([sets[0].intersection(sets[1])] + sets[2:])

def isnum(s):
    if s is None:
        return False
    try:
        float(s)
        return True
    except ValueError:
        return False

def progressBar(percnt):
    width = 50
    width = width / 100
```

```
38      percnt = floor(100*percnt)
39      retstr = "\r{:3d}% [".format(floor(percnt))
40      retstr += "".join(["=" if x < floor(percnt*width) else " " for x in range(floor(100*width))])
41      retstr += "]"
42      return retstr
43
44  def fileLineCount(filename):
45      with open(filename) as f:
46          num_lines = sum(1 for line in f)
47      return num_lines
48
49  def nCr(n,r):
50      f = factorial
51      return (f(n) / f(n-r)) / f(r)
```