

# Tertius extension to Weka

Amélie Deltour

## 1 Introduction

This project was done in the framework of a training period of 8 weeks.

The aim of this report is to present the main features of the Tertius extension to Weka that has been developed.

## 2 Presentation

### 2.1 Installation

There are currently two versions of Tertius: a public version (tertius 2.0) which is a stable version of the Tertius algorithm, and a development version (tertius 2.1) which has additional options and can be used to test and further develop the program (see section 2.3).

The Tertius package is included in the jar archive `tertius-2-0.jar` (`tertius-2-1.jar`). The archive can be found in `/home/compsci/guests/adeltour/unix/Projet/Tertius`. This archive can be extracted with the `jar` utility by typing:

```
jar xvf tertiary-2-0.jar
```

The `README` file explains how to install and run Tertius (see Appendix A).

Tertius can be run from the command line, or by using the Weka GUI.

### 2.2 Options

Tertius can be run with the following options: (command line option / GUI explorer option)

#### General options

**-t <name of training file>**

Set the training file.

#### Pruning

Several options can be used to bias the search and allow more or less pruning:

**-K <number of values in result> / confirmationValues**

This option specifies the number of best confirmation values to be returned by the algorithm.

The number of values must be greater than 1.

The default setting is 10.

**-F <frequency threshold> / frequencyThreshold**

This option is to set a threshold on the minimal frequency of the instances satisfying the body and the instances satisfying the negations of the head.

The threshold must be between 0 and 1.

The default setting is 0 (no threshold).

**-C <confirmation threshold> / confirmationThreshold**

This option sets a threshold on the minimal confirmation of the results. This option can be used alternatively to the `-K` option: all the rules over the threshold will be returned.

The threshold must be between 0 and 1.

The default setting is 0 (no threshold).

**-N <noise threshold> / noiseFrequency**

This option sets a threshold on the maximal frequency of counter-instances of the results. With a threshold of 0, only satisfied rules (rules with no counter-instances) will be returned.

The threshold must be between 0 and 1.

The default setting is 1 (all rules are kept).

## Search space and language bias

The following options are syntactic constraints put on the hypothesis language.

### **-R / repeat**

This allows to repeat an attribute in a rule (with different values of course). With this option, rules can express the notion of a set of values for an attribute.

The default is that attributes are not repeated.

### **-L <number of literals> / numLiterals**

This option sets the maximum number of literals (attribute-value pairs) in a rule.

The number of literals must be greater than 1.

The default setting is 4.

### **-G <0=no negation | 1=body | 2=head | 3=body and head> / negation**

This option sets the kind of negation that can be used in a rule.

The default setting is 0 (no negation will be used).

### **-S / classification**

This option allows to do classification: the head of the rules will consist of the target attribute.

This option can be used with the `-G` option to get positive or negative classification.

### **-c <class index> / classIndex**

This option sets the target (class) attribute.

The class index must be between 0 and the number of attributes of the data file.

The default is 0, which means that the last attribute will be used.

### **-H / horn**

This option can be used to consider horn clauses only.

## Subsumption tests

### **-E / equivalent**

What we call equivalent here are rules like  $A \implies B$  and  $\text{not } B \implies \text{not } A$  (sometimes the negation is implicit, since for boolean values the default is that  $A = \text{true} \implies \text{not } A = \text{false}$ ).

Such rules have the same confirmation and number of counter-instances, so they are strictly equivalent from the point of view of the Tertius algorithm. Consequently, the default is that only one rule is kept.

The `-E` option allows to keep both rules.

### **-M / sameClause**

What we call here same clauses are for example rules like  $A \implies B \text{ or } C$  and  $A \text{ and not } C \implies B$ .

These rules correspond to the same clause  $(\neg A \vee B \vee C)$ , nevertheless their confirmation can be different.

The default is that only the “best” rules are kept. A rule is said to be better than another if it has a higher confirmation value.

With the `-M` option, this test is not performed and all the rules are kept.

### **-T / subsumption**

The default concerning cases of subsumption other than equivalent rules or same clauses is that only the “best” rules are kept. Here a rule is better than another if it has a better confirmation value, a lower number of counter-instances and it subsumes the other rule (it is simpler). Two rules that cannot be compared (for example one subsumes the other but has more counter-instances) are both kept.

With the `-T` option, this subsumption test is not performed and all rules are kept.

## Missing values

### **-I <0=always match | 1=never match | 2=significant> / missingValues**

This option sets the way to handle missing values (see section 4.2). Missing values can be set to match all values, to match no value, or to be significant and be considered as a new value.

The default setting is 0.

## ROC analysis

### **-O / rocAnalysis**

This option allows to perform ROC analysis on the rules found.

The default is that rules are given with their confirmation value and the frequency of counter-instances in the data. With this option, instead of the frequency of counter-instances, the TP-rate and FP-rate are given.

## Multi-table learning

**-p <name of file> / partFile**

This option allows to use a second file for individual-based learning. The file given with the `-t` option must contain the data about the individuals, and the file given with the `-p` option must contain the data about the parts (see sections 5 and 6).

## Output of current values

**-P <0=no output | 1=on stdout | 2=in separate window> / valuesOutPut**

This option can be used to get an output during the search of the best and worst current values found (confirmation values and number of counter-instances). If the window is used, a button allows to stop the search, and then the best rules found so far are given (see section 4.3).

Getting the values on the standard output only works if you run Tertius from the command line, if you run it from the GUI (Explorer or SimpleCLI) a separate window should be used.

The default setting is no output.

## 2.3 Development version

The development version has a few more options. These options allow to test different possible settings for the program, and can be used for development purposes to improve these settings. In the public version, the setting which seems to be the best one in most cases is used. More details about the settings and the choices that have been made in the public version can be found further, and a few test results can be found in Appendix B and Appendix C.

**-A <algorithm> / algorithm**

This option allows to choose the search algorithm to be used, so as to compare the performance. Three algorithms are available (see section 3.1).

The default setting (which is used in the public version) is 2.

**-B <0=best first | 1=breadth first | 2=depth first> / searchApproach**

This option sets the search approach, which can be a best first search, a breadth first search or a depth first search. This allows to compare the performance obtained with different approaches, on different search spaces.

The default setting (used in the public version) is a best first search.

**-U / storeInstances**

This options allows to store the counter-instances in the rules, instead of counting the counter-instances from the data file (see section 3.3). Using this option makes the program run faster with a large number of instances, but it requires more memory.

**-W / twoValuesAsBoolean**

The default in the program is to consider attributes with two values as boolean attributes. That is to say that for example `not play = true` is equivalent to `play = false`, or if we have an attribute humidity that can take only two values, normal or high, `not humidity = normal` is equivalent to `humidity = high`.

Consequently, both rules `play = true ==> humidity = normal` and `humidity = high ==> play = false` are considered as equivalent and only one rule will be given (unless you use the `-E` option).

Another consequence is that if you allow negation in the rule with the `-G` option, you will never find `not play = yes`, you will find `play = no` instead.

With the `-W` option, you do not have this bias, and attributes with two values are just considered as any other attribute. This option allows in particular to compare results obtained with the C implementation, where this bias does not exist, and the Java implementation.

**-V <level> / verbosityLevel**

This option can be used to get an output explaining the main steps of the search. There are four levels of verbosity:

- level 0: no output
- level 1: outputs which rule is refined and the children that are then considered.
- level 2: outputs for each rule considered whether it can be refined, explored, added to the results, added to the nodes...
- level 3: outputs the explanation for each choice (why a rule cannot be refined, added to the results...)

All the tests implied by this option make the program slower, that is why the option is only available in the development version. The main use of this option is for debugging purposes.

Another difference between the two versions is that in the development version the `printStackTrace()` method is called when an exception is raised, so as to see where this exception comes from.

### 3 Differences with the C implementation

There are slight differences between this implementation of Tertius in Java and the C implementation made by Nicolas Lachiche. Some options are slightly different, but in general both programs allow to do the same things. The main differences are differences that aim at making the program run faster, since the Java implementation tends to be slow.

#### 3.1 Algorithm

##### First algorithm

In the C implementation, the algorithm is the following (assuming a best first search is used):

```
agenda ← {empty rule}
while agenda is not empty
  rule ← first rule of the agenda
  if rule can be stored in results
    add rule to results
  if rule can be refined
    refine rule
    for each child
      calculate optimistic estimate and confirmation
      if child can be stored in agenda
        add child to agenda
    sort agenda according to optimistic estimate
```

The drawback of this algorithm is that if a child has a good confirmation, but an optimistic estimate which is not so good, it will be at the end of the agenda. So it will be added to the results only late in the search, which is bad for pruning. Of course, the confirmation is linked with the optimistic estimate, since the optimistic estimate is an upper bound for the confirmation. But a rule can have an optimistic estimate of 0.7, which is not very good since a lot of rules often have a high optimistic estimate, and a confirmation of 0.65, which is quite good and would probably allow it to be in the results.

##### Second algorithm

Consequently, another algorithm can be considered that considers adding the rules to the results before putting them to the agenda. The agenda is only used for the rules to be refined. Rules with a good optimistic estimate are refined first, so it is likely that one of the children will have a good confirmation. The children are considered as soon as they are generated, so they are added to the results as soon as possible, and this allows more pruning (the better the rules in the results are, the more rules are pruned).

The algorithm obtained is the following:

```
agenda ← {empty rule}
while agenda is not empty
  rule ← first rule of the agenda
  if rule can be refined
    refine rule
    for each child
      calculate optimistic estimate
      if child can be explored
        if child can be stored in agenda
          add child to agenda
        calculate confirmation
        if child can be stored in results
          add child to results
    sort agenda according to optimistic estimate
```

If we compare the performances of both algorithms, we can observe that the same number of rules are generated (the same number of rules are refined), but a lot less rules are explored in the second algorithm. What we call here explore a rule is consider it for refinement and for adding to the results (this corresponds to the rules stored in the agenda in the first algorithm, and to the rules passing the 'can be explored' test in the second algorithm). Consequently the second algorithm allows to know much sooner that a rule is not worth keeping and can be pruned away, and this leads to a gain in terms of time used for the search.

Furthermore, in the second algorithm rules in the agenda are only considered for refining. Since the 'can be refined' test is based on the optimistic estimate, and the agenda is sorted according to the optimistic estimate, if a rule can not be refined, then it is not worth considering all the following rules in the agenda, and the search can be stopped:

```

agenda ← {empty rule}
while agenda is not empty
  rule ← first rule of the agenda
  if rule can be refined
    refine rule
    for each child
      ...
    sort agenda according to optimistic estimate
  else break

```

This makes the search a little quicker.

Another way to make the search quicker is to split the tests, so as not to calculate the confirmation and the optimistic estimate of the children when you can know in advance that they can be pruned away. For example, you can prune a rule if it is not over the frequency threshold, without having to calculate the optimistic estimate. This leads to the following algorithm:

```

agenda ← {empty rule}
while agenda is not empty
  rule ← first rule of the agenda
  if canRefine(rule)
    refine rule
    for each child
      if canCalculateOptimistic(child)
        calculate optimistic estimate
      if canExplore(child)
        if canStoreInNodes(child)
          add child to agenda
        if canCalculateConfirmation(child)
          calculate confirmation
          if canStoreInResults(child)
            add child to results
    sort agenda according to optimistic estimate
  else break

```

The tests are the following:

- `canRefine`: returns `true` if the rule is empty or there are not enough confirmation values in the results or the optimistic estimate of the rule is greater than the confirmation value of the last rule in the results; else returns `false`.
- `canCalculateOptimistic`: returns `false` if the rule has a false head or a true body, or it is not over the frequency threshold; else returns `true`.
- `canExplore`: returns `false` if the confirmation is under the confirmation threshold, or there are enough confirmation values in the results and the optimistic estimate of the rule is lower than the confirmation of the last rule in the results; returns `true` if there are not enough confirmation values in the results or the optimistic estimate of the rule is greater than the confirmation value of the last rule in the results.
- `canStoreInNodes`: returns `false` if the rule has no counter-instances; else returns `true`.
- `canCalculateConfirmation`: returns `false` if the number of counter-instances of the rule is over the noise threshold; else returns `true`.
- `canStoreInResults`: returns `false` if the confirmation is under the confirmation threshold, or there are enough confirmation values in the results and the confirmation value of the rule is lower than the confirmation of the last rule in the results; returns `true` if there are not enough confirmation values in the results or the confirmation value of the rule is greater than the confirmation value of the last rule in the results.

### Third algorithm

Now some tests only take the optimistic estimate and the confirmation into account. Following the idea of breaking the search, we can consider breaking the search in the children as well, by sorting the children at different stages. This leads to a third algorithm:

```

agenda ← {empty rule}
while agenda is not empty
  rule ← first rule of the agenda
  if canRefine(rule)
    refine rule
    for each child
      if canCalculateOptimistic(child)
        calculate optimistic estimate
      else remove child
    sort remaining children according to optimistic estimate
    for each remaining child
      if canExplore(child)
        if canStoreInNodes(child)
          add child to the agenda
        if canCalculateConfirmation(child)
          calculate confirmation
          add child to remaining
      else break
    sort remaining children according to confirmation
    for each remaining child
      if canStoreInResults(child)
        add child to the results
      else break
    sort agenda according to optimistic estimate
  else break

```

However, this algorithm does not seem to be quicker than the previous one. Indeed, the cost of sorting the children compensates the breaks.

### 3.2 Rules refinement

The search starts with the empty rule and the refinement operation is just adding an attribute-value pair either in the head or in the body of the rule. The same kind of non-redundant refinement operator as in the C implementation of Tertius is used: attributes are ordered following the order in the dataset, and values for an attribute are ordered in the same way (the program can only deal with nominal attributes).

However, it can be noticed that rules with a false head or a true body (rules of the form  $\dots \implies \text{FALSE}$  and  $\text{TRUE} \implies \dots$ ) always have a confirmation value of 0 and often have an optimistic estimate of 1 (in the first case, the optimistic estimate is 1 provided that the frequency of the body is more than 0.5, and in the second case it is 1 provided that the frequency of the negation of the head is more than 0.5, which happens quite often). Therefore, that kind of rules are kept in the agenda because of their high optimistic estimate. When such rules are refined by adding a predicate to the empty head or body, it decreases the optimistic estimate and often allows pruning. But if they are refined by keeping an empty head or body, they often still keep that high optimistic estimate. So that kind of refinement is done, adding a high number of rules to the agenda with an empty part that are not really useful, until the rules are refined enough to decrease the optimistic estimate.

This can be observed in particular with the  $-C\ 1$  option : even with a confirmation threshold of 1, a lot of rules are generated, to finally give a result with no rules at all!

A way to avoid this is to never leave a rule with an empty head or an empty body, by forcing the refinement operator to add predicates to the head only when the head is empty, and the same for the body. This leads to a refinement operator which is a bit more complicated. Let's take an example of four predicates (or attribute-value pairs) A, B, C, D ordered in this order.

The  $\text{TRUE} \implies A$  rule can be refined to  $B \implies A$  or  $C \implies A$  or  $D \implies A$ . But then, if we want to get all the possible rules,  $C \implies A$  must be allowed to be refined to  $C \implies A$  or  $B$ , even if B is before C in the order. That means that while there is a gap between the predicates in the head and the predicate in the body, and there is only one predicate in the body (otherwise we get the same rule several times), predicates in this gap can be added to the head. Predicates being after all the predicates in the rule can always be added to the body or the head.

With this refinement policy, the optimistic estimate of the rules is decreased much sooner, and it allows more pruning, which makes the program run quicker.

However, the limit to this change of the refinement policy is that it assumes that rules with a confirmation of 0 are not interesting. But sometimes, rules like  $\text{outlook} = \text{sunny}$  and  $\text{humidity} = \text{high} \implies \text{FALSE}$  can be interesting even if they have a confirmation of 0, because they are true rules and express an interesting fact. With the new refinement policy, such rules can not be obtained. Nevertheless, rules corresponding to the same clause and with a higher confirmation can be obtained if negation is used, for example:  $\text{outlook} = \text{sunny} \implies \text{not humidity} = \text{high}$ . So the program offers a way to express these interesting clauses with another rule.

### 3.3 Counting instances

In the original version of Tertius, when you want to calculate the optimistic estimate and confirmation of the rule, you first have to count in the dataset the number of counter-instances of the rule, and the number of instances satisfying the body and satisfying the negation of the head.

However, we can notice that a rule is always obtained by refining another rule, and the refinements considered can only decrease these numbers. So, if we associate with each rule its counter-instances, instances satisfying the head and instances satisfying the body, then you only have to do an update when a rule is refined, by considering these instances and checking if they are still counter-instances, or still satisfy the body, or still satisfy the negation of the head. This implies considering only a small part of the instances of the dataset instead of considering the whole dataset.

Experiments run on the mushroom dataset, which contains 8124 instances, have shown that the time needed to find the results could drop significantly from almost 2 minutes to 1 minute in some cases. However, in other cases (with other options), storing instances requires more time than counting (this may be due to a problem with data structures, see section 7.3).

Moreover, experiments run on the soybean dataset, which contains 683 instances and 36 attributes, with 2 to 19 values each, have shown that storing instances makes the program run out of memory as soon as we ask for rules with more than 2 literals, whereas without counting instances we can get rules with 4 literals.

Some results of the experiments can be found in Appendix B.

Consequently, it has been decided that storing instances would be optional. Since it makes the program run out of memory too often, and it could not be proved to make the program run faster in all cases, this option is available in the development version only, and in the public version instances are never stored.

### 3.4 Negation

In the C implementation of Tertius, rules are based on the notion of clauses. That means that negated predicates will be in the head and other predicates will be in the body of the rule, unless a classification bias is used.

This version of Tertius is based on the notion of head and body, and thus negative predicates can be considered in the head or body of the rules. Of course, considering negative predicates increases the search space and is often too time-consuming. But in cases where the search space is not too big (less than 10 predicates with only 2 or 3 values each) it can be interesting to consider negated predicates as well and see the results that can be obtained. This allows getting rules with a higher confirmation, because for example if the rule  $A \implies B \text{ or } C$  is not good, it is possible that the rule  $A \text{ and not } C \implies B$ , which corresponds to the same clause, is much better.

## 4 Other features

### 4.1 Handling numeric attributes

Nothing particular has been done so far for the program to handle numeric attributes. It can only use nominal attributes, and datasets with numeric attributes have to be first discretized using the Weka toolkit. When discretization is done, the number of nominal values should be kept low, otherwise the search space might be too wide for the search to work effectively.

### 4.2 Handling missing values

There are many ways of handling missing values.

A first way is to consider that missing values can be significant, and should be considered like other values.

If we do not consider this, then the main problem is, considering an instance  $\text{outlook} = ? - \text{play} = \text{yes}$ , whether it should be counted as a counter-instance for the rule  $\text{play} = \text{yes} \implies \text{outlook} = \text{sunny}$ .

This leads to three different approaches concerning missing values.

#### “Significant” approach

In this approach, missing values are considered as a new value for the attribute. Consequently, missing values can appear in rules like any other value. This approach should be used if missing values are supposed to be significant.

One of the side effects is that  $\text{play} = \text{no}$  is no longer considered as the negation of  $\text{play} = \text{yes}$  (see section 2.3), since  $\text{not play} = \text{yes}$  can be either  $\text{play} = \text{no}$  or  $\text{play} = ?$ .

#### “Always match” approach

From the point of view of counter-instances, this approach minimizes the number of counter-instances for a rule. Indeed, it assumes that  $\text{outlook} = ? - \text{play} = \text{yes}$  is not a counter-instance for the rule  $\text{play} = \text{yes} \implies \text{outlook} = \text{sunny}$ , nor for the rule  $\text{not outlook} = \text{sunny} \implies \text{play} = \text{no}$ .

From the point of view of matches, this is a strong assumption since it means that a missing value (?) matches any value. Indeed, in the example above, if `outlook = ?` does not match `not outlook = sunny` (this is what is considered when we count counter-instances), this means implicitly that it matches `outlook = sunny`. In a way, this is similar to an “explicit negation” in the first-order version of Tertius. Indeed, if we have a predicate `outlook` with explicit negation, then a day with no data on the outlook will not be counted as a counter-instance of the rules given above.

### “Never match” approach

From the point of view of counter-instances, this is a greedy approach. It considers that `outlook = ? - play = yes` is a counter-instance for the rules `play = yes ==> outlook = sunny`, and `not outlook = sunny ==> play = no`.

From the point of view of matches, this means implicitly that a missing value never matches other values in a rule.

In this approach, it can no longer be considered that `play = no` is the negation of `play = yes` (see section 2.3), because an instance matching `not play = yes` can be either `play = no` or `play = ?`.

In a way, this is similar to a “Closed-World Assumption” in the first-order version of Tertius. Indeed, if we have a predicate `outlook` with closed-world assumption, then a day with no data on the outlook will be counted as a counter-instance of the rules give above.

## 4.3 Interrupting the search

The search can be interrupted in two cases: when the program runs out of memory, or when the user decides to interrupt it because it takes too much time.

### Memory interruption

When the program runs out of memory, the best rules found so far are printed, and a message indicates that the search was interrupted.

### User interruption

Several ways can be used to interrupt the search, according from where the program is run.

If the program is run from the command line, then it can be interrupted with Control C. In that case however the program just stops and no result is given. An alternative is to use the `-P 2` option, which provides a button to interrupt the search. Then the best results found so far are printed. It is to note that the search is not stopped immediately, sometimes it takes a few seconds before it stops.

If the program is run from the Weka SimpleCLI, the program can also be interrupted with this `-P 2` option. But it is recommended to use the `break` command instead, because it makes the program stop quicker. In both cases, the best rules are printed.

If the program is run from the Weka Explorer, it can be interrupted by using the Stop button provided by the interface. Nevertheless, there is a bug in Weka 3.2 and this button does not work. The bug should be corrected in the next version. But it is all the same recommended to use the `-P 2` option instead, since the button provided by the Weka interface (if the bug is corrected and it calls the right method) stops the search quicker but not in a good way (this may imply some problems, see section 7.3 to know why).

## 5 Multi-table learning

This version of Tertius can also use several tables for individual-based learning, but in a quite restrictive case.

It can deal with two dataset files, one containing individuals, and the other containing the parts of the individuals. Then it can find rules using properties of the individual and from its parts. The properties of the individual can only be found in the body of the rule.

It is just an extension of attribute-value literals, but with attribute coming from different files. A SQL-like syntax is used, with a “.” between the name of the relation and the name of the attribute.

An existential quantifier is used for the parts. For example, a rule like `key.color = green ==> keyring.class = good` must be interpreted as “if a keyring contains a key with the color green, then the class of the keyring is good”.

The link between the two files is made with a particular attribute. There must be an attribute called `id` in the individual file, being a sort of key for the individuals, and an attribute called `id` as well in the parts file, referring to the individual each instance is a part of.

This can be used for example on the “mutagenic” problem, to find rules identifying mutagenic molecules from the properties of the atoms (see section 6).

However, this version of individual-based learning is very limited. First of all, it can only deal with two files, not more. There must be a one-to-many relation between the individuals and the parts. And only one part of each individual can appear in the rules, that is to say that the rules can not express for example “if a keyring contains a blue key and a green key then it is good”.

## 6 Examples

Here are a few example on the `weather.nominal.arff` dataset, which is a small dataset, with only 6 attributes with 1 or 2 values each, and 14 instances.

If we only want 6 confirmation values, we get the following result:

```
java tertius.Tertius -t weather.arff -K 6
```

```
Tertius
=====
```

```
1. /* 0.633754 0.071429 */ play = yes ==> outlook = overcast or humidity = normal
2. /* 0.607625 0.000000 */ humidity = normal ==> temperature = cool or play = yes
3. /* 0.607625 0.000000 */ temperature = cool ==> humidity = normal
4. /* 0.594071 0.214286 */ humidity = normal ==> temperature = cool
5. /* 0.590214 0.000000 */ outlook = sunny and humidity = high ==> play = no
6. /* 0.555556 0.000000 */ play = no ==> outlook = sunny or windy = TRUE
7. /* 0.486606 0.000000 */ humidity = normal ==> outlook = rainy or play = yes
8. /* 0.486606 0.000000 */ outlook = sunny and play = no ==> humidity = high
```

```
Number of hypotheses considered: 1353
Number of hypotheses explored: 481
Time: 00 min 01 s 680 ms
```

The first number given with the rules is the confirmation value, and the second number is the frequency of counter-instances.

The “number of hypotheses considered” is the number of rules generated with the refinement operator.

The “number of hypotheses explored” is the number of rules that were “potentially interesting” and were considered for adding to the results or refining. This corresponds to the number of rules taken from the agenda in the first algorithm (see section 3.1). It can be compared with the number given by the C implementation, since they count the same thing.

If we use ROC analysis, we get the following result:

```
java tertius.Tertius -t weather.arff -K 6 -O
```

```
Tertius
=====
```

```
1. /* 0.633754 0.888889 0.200000 */ play = yes ==> outlook = overcast or humidity = normal
2. /* 0.607625 0.700000 0.000000 */ humidity = normal ==> temperature = cool or play = yes
3. /* 0.607625 0.571429 0.000000 */ temperature = cool ==> humidity = normal
4. /* 0.594071 1.000000 0.300000 */ humidity = normal ==> temperature = cool
5. /* 0.590214 0.600000 0.000000 */ outlook = sunny and humidity = high ==> play = no
6. /* 0.555556 0.555556 0.000000 */ play = no ==> outlook = sunny or windy = TRUE
7. /* 0.486606 0.636364 0.000000 */ humidity = normal ==> outlook = rainy or play = yes
8. /* 0.486606 0.428571 0.000000 */ outlook = sunny and play = no ==> humidity = high
```

If we try to find rules with negations, we get one more rule which turns out to be more interesting than the two last rules in the previous example:

```
java tertius.Tertius -t weather.arff -G 2 -K 6
```

```
Tertius
=====
```

```
1. /* 0.633754 0.071429 */ play = yes ==> outlook = overcast or humidity = normal
2. /* 0.607625 0.000000 */ humidity = normal ==> temperature = cool or play = yes
3. /* 0.607625 0.000000 */ temperature = cool ==> humidity = normal
4. /* 0.594071 0.214286 */ humidity = normal ==> temperature = cool
5. /* 0.590214 0.000000 */ outlook = sunny and humidity = high ==> play = no
6. /* 0.555556 0.000000 */ play = no ==> outlook = sunny or windy = TRUE
7. /* 0.538289 0.000000 */ windy = FALSE and play = yes ==> not outlook = sunny or temperature = cool
```

```
Number of hypotheses considered: 1615
Number of hypotheses explored: 571
Time: 00 min 01 s 937 ms
```

Here is now an example of the performance on a larger dataset: `soybean.arff`, which has 36 attributes and 683 instances. The search space is much bigger, and here we must restrain to three literals in each rule otherwise the search is too long.

```
java weka.associations.tertius.Tertiuss -t soybean.arff -L 3
```

```
Tertiuss  
=====
```

```
1. /* 0,801047 0,043924 */ stem-cankers = absent and fruit-spots = absent ==> stem = norm  
2. /* 0,797104 0,051245 */ stem-cankers = absent and int-discolor = none ==> stem = norm  
3. /* 0,796114 0,048316 */ fruiting-bodies = absent and fruit-spots = absent ==> stem = norm  
4. /* 0,792484 0,035139 */ canker-lesion = dna and fruit-spots = absent ==> stem = norm  
5. /* 0,791883 0,048316 */ fruit-spots = absent ==> stem = norm or class = brown-spot  
6. /* 0,789718 0,036603 */ canker-lesion = dna and fruiting-bodies = absent ==> stem = norm  
7. /* 0,784201 0,039531 */ canker-lesion = dna and int-discolor = none ==> stem = norm  
8. /* 0,777930 0,020498 */ stem-cankers = absent and fruit-spots = absent ==> canker-lesion =  
dna  
9. /* 0,775079 0,010249 */ leaves = abnorm and stem = norm ==> canker-lesion = dna  
10. /* 0,773579 0,002928 */ stem = norm ==> canker-lesion = dna or class = purple-seed-stain
```

```
Number of hypotheses considered: 244429  
Number of hypotheses explored: 77571  
Time: 27 min 36 s 403 ms
```

Here is now an example of individual-based learning applied to the problem of classifying mutagenic molecules. Here the individuals are molecules, which properties are in the `molecule.arff` file, and the parts of the molecules are atoms, which properties are in the `atom.arff` file. The results are the same as those obtained by the C version of `tertius` with the `-b class` option.

```
java weka.associations.tertius.Tertiuss -t molecule.arff -p atom.arff -S
```

```
Tertiuss  
=====
```

```
1. /* 0,364484 0,058511 */ atom.type = 27 ==> molecule.class = active  
2. /* 0,231743 0,063830 */ atom.type = 29 ==> molecule.class = active  
3. /* 0,210764 0,000000 */ atom.type = 28 ==> molecule.class = active  
4. /* 0,198029 0,005319 */ atom.charge = 0.142 ==> molecule.class = active  
5. /* 0,198029 0,005319 */ atom.charge = -0.118 ==> molecule.class = active  
6. /* 0,194423 0,010638 */ atom.type = 50 ==> molecule.class = inactive  
7. /* 0,187607 0,000000 */ atom.charge = 0.812 ==> molecule.class = active  
8. /* 0,182032 0,005319 */ atom.charge = 0.012 ==> molecule.class = active  
9. /* 0,182032 0,005319 */ atom.charge = 0.145 ==> molecule.class = active  
10. /* 0,179557 0,000000 */ atom.charge = 0.141 ==> molecule.class = active  
11. /* 0,175114 0,090426 */ atom.type = 1 ==> molecule.class = inactive  
12. /* 0,173767 0,005319 */ atom.charge = -0.388 ==> molecule.class = active
```

```
Number of hypotheses considered: 14814  
Number of hypotheses explored: 248  
Time: 09 min 50 s 690 ms
```

## 7 Java implementation

### 7.1 Short description of the classes

The Java implementation uses a class for the rules defined in `Rule.java`. Each rule has a body and a head, which are defined in `Body.java` and `Head.java`. These two classes extend the `LiteralSet` class, which is an abstract class. With a `Rule` and with a `LiteralSet` can be associated a set of instances (see section 3.3). For a `Rule`, these are the counter-instances of the rule. For a `LiteralSet`, they are called counter-instances as well and represent in fact the instances satisfying the `LiteralSet` if it is a `Body`, and satisfying the negation of the `LiteralSet` if it is a `Head`.

`LiteralSet` has three abstract methods, which are different if you consider a `Head` or a `Body`:

- `canKeep` is a method saying whether an instance can be kept in the set of counter-instances associated with this `LiteralSet`.
- `isIncludedIn` is a method saying if this `LiteralSet` is included in another rule.
- `toString` is a method returning a `String` for this `LiteralSet`.

A `Predicate` class is used to describe the predicates used in the rules. Here, in the case of attribute-value, a `Predicate` represents an attribute. A `Predicate` consists mainly of a name and a set of literals. The literals are described by a `Literal` abstract class. Each `Literal` has a sign and is associated with its negation (which can be null if no negation is used in the search), and the corresponding `Predicate.Literal` has three abstract methods:

- `satisfies` is a method saying if this `Literal` satisfies an instance.
- `negationSatisfies` is a method saying if the negation of this `Literal` satisfies an instance.
- `toString` is a method returning a `String` for this `Literal`.

Two sorts of `Literal` have been implemented: `AttributeValueLiteral`, which represents an attribute-value pair, and `IndividualLiteral`, which extends `AttributeValueLiteral` and represents a literal used in individual-based learning (see section 5).

All these classes are included in a package called `weka.associations.tertius`.

Further information can be found in the documentation generated with `javadoc`.

## 7.2 Data structures

Many lists or sets are used in the program: to store the results, to store the nodes to explore, to store the counter-instances...

Concerning the nodes to explore (the agenda), the choice of a good data structure was critical since a bad data structure can make the search slower. The operation we need on this list of nodes is adding a node, and getting the first node. The children generated from a node must be stored in the same structure, and what we need with them is iterating and sorting. Then we need to merge the agenda with the sorted children.

It seemed quite obvious that the best data structure to optimise the time needed for these operations was a linked list. Experiments have been run with other data structures, confirming that it was actually quicker. The problem with the `LinkedList` class provided by Java is that when we want to implement a sort and a merge on it, it is not so easy. So a `SimpleLinkedList` class was implemented. It is not more efficient than the `LinkedList` class, but it implements a quicksort and a merge and is therefore easier to use for the purpose of the search.

This `SimpleLinkedList` class was also used to store the results. It provides a `LinkedListIterator`, and also a `LinkedListInverseIterator`, to iterate the list from the end.

For the counter-instances, since the set has to be iterated and instances have to be removed, using a linked list also seemed convenient. But it turned out that this required too much memory. Therefore, an `ArrayList` was used. This uses less memory, but it is probably slower to remove an element from an `ArrayList` than from a `LinkedList`, so this solution is not optimal.

For other sets (sets of literals for example), `ArrayList` has been used. `Vector` is said to be slow because it is synchronized, whereas `ArrayList` is not synchronized.

## 7.3 Interruption of the search

The search can be interrupted by the user in two ways: the ways provided by the Weka interface (break command in `SimpleCLI` or “Stop” button in `Explorer`), or the button provided by the `-P 2` option.

The Weka interface uses the `interrupt()` method to interrupt the thread running `Tertius`. However, interrupting a thread does not seem to really stop it, it only has an effect in some particular cases, if the thread is waiting for example. So the solution to stop the search when the thread is interrupted is the following: the thread launched by the Weka interface, when it comes to the `buildAssociation` method, launches another thread making the search, while itself waits for the end of this other thread. Executing the `wait()` method allows to catch an `InterruptedException` if the thread was interrupted. After the exception is caught, the program continues and it has for effect to print the results, which are the results found until the search was interrupted. However, the thread making the search has to be stopped as well, otherwise it continues running. The `interrupt()` method has no effect on it, and the `stop()` method is deprecated. As recommended by the Java tutorial, the best approach to stop a thread without using the `stop()` method is to use a variable to indicate that the thread should stop running. So a variable called `m_status` is used in `Tertius`, and before taking a new node from the agenda, the search method checks this variable.

The problem with the button provided by the `Explorer` is that it uses the `stop()` method, just after calling the `interrupt()` method. This method is deprecated and should not be used. The result is that the thread does not end correctly and some problems may appear (the “Start” button is not enabled again).

The button provided by the `-P 2` option does not have access to the thread running `Tertius`, since this thread is launched by the Weka interface. Consequently, the only thing it can do is changing the `m_status` variable, so that the search stops. That is why in this case the search does not stop immediately: if the search is interrupted just after a new node was taken from the agenda, the program first ends refining this node, before it realizes that the search was interrupted.

## 7.4 Individual-based learning

For individual-based learning, a new class for instances, called `IndividualInstances`, was created, extending the `Instances` class. Instead of containing just one set of instances, `IndividualInstances` contains two sets of instances: the individuals and the parts. Actually, since the program is restricted to a one-to-many relation, the individual instances themselves contain the instances corresponding to their parts. An `IndividualInstance` class was created, extending the `Instance` class. Each `IndividualInstance` contains an `Instances` object, being the parts of this individual.

Then, the `counterInstance` method in `LiteralSet` is different if the `Instance` it considers is an `IndividualInstance`: in that case, it has to consider all the parts of this individual.

An `IndividualLiteral` class was created as well. An `IndividualLiteral` can have two types: it can be either an individual property, or a part property. This is taken into account for the refinement of rules: part properties can only be added to the body of the rules, and once individual properties have already been added.

## 7.5 How to extend Weka

The purpose of this `Tertius` package is to be used as an extension to the Weka toolkit. Here is a short explanation of how to do it.

### Interface

First of all, as explained in Chapter 8.5 of the *Data Mining* book by Ian H. Witten and Eibe Frank, if you want to add a new algorithm to Weka, it should implement one of the interfaces provided with Weka. For example `Tertius` implements the `Associator` interface (we consider `Tertius` as an associator even if the rules are not really association rules, since it does unsupervised learning too).

### Options

The best way to deal with options is that the algorithm implements the `OptionHandler` interface provided with Weka. This implies implementing the `getOptions`, `listOptions` and `setOptions` methods (see the Weka documentation).

### Weka Explorer

Specific methods must be provided, that can be recognized by the Weka Explorer to get and set the options, and provide some help.

For each option with the name `nameOfOption` you should have the following methods:

- `public String nameOfOptionTipText()` returns a `String` describing this option.
- `public typeOfOption getNameOfOption()` returns the value of the option.
- `public void setNameOfOption(typeOfOption v)` sets the option to the value `v`.

Finally, a `public String globalInfo()` method should return a `String` describing the algorithm.

### GenericObjectEditor.props file

To add a package to the Weka GUI is quite simple. Indeed, all the algorithm-classes that can be used by Weka are specified in a single file, called `GenericEditor.props`.

The default is that the file included in the `weka.jar` archive is used. But you can override these settings by putting a similar file either in your home directory, or in the directory from where you want to start Weka (then the new settings will only be taken into account if you run Weka from this directory).

In the case of `Tertius`, we can for example use a file containing the following lines:

```
weka.associations.Associator=\
  weka.associations.Apriori, \
  weka.associations.tertius.Tertius
```

This overrides the settings for associators only, and specifies that the associators available will be `Apriori` and `Tertius`.

### Classpath

If Weka is run from the command-line by typing for example `java weka.gui.GUIChooser.java`, then the `$CLASSPATH` environment variable must be set to specify where the `Tertius` classes can be found.

If Weka is run from the `weka.jar` archive by typing for example `java -jar weka.jar`, then the `weka.jar` archive itself must specify where the `tertius` classes are to be found. This can be done by adding the following line to the manifest of the `weka.jar` archive:

```
Class-Path: tertius.jar
```

(with of course the correct path to find the `tertius.jar` archive).

The `README` file (see Appendix A) explains how to update the `weka.jar` archive.

## 8 Further work

First of all, the program is still very slow, so ways of making it run faster can still be considered. Several ways have already been considered (see section 3), and many changes have been made on the data structures during the development to try and find the best ones. However, this could probably still be improved.

Experiments should be run to compare performance of the three algorithms on several datasets with different sizes (in number of attribute-values), and to compare the performance of the different search approaches. For the moment, the second algorithm and the best-first approach seem to be the best ones, but not enough experiments were run. If enough experiments are run, a learning algorithm could be used to know what is the best approach considering the features of the dataset (number of attributes, of values...), and the program itself could maybe choose the approach to use.

It could also be worked on the memory problem, so that for example storing the counter-instances does not make the program run out of memory so quickly. But anyway, there will always be a choice to make between speed and memory!

Finally, with more time, what would be very interesting would be to think about a way to use data from several files, but in a more general way, without the limits of this version. This would imply a deep reflection on the way to express relations between the different files (the notion of keys and foreign keys in a relational model, or of entities and associations between entities in an entity-association model).



**Appendix A  
README file**



```
=====
          =====
          README
          =====
```

Installation of the Tertius package in Weka

- ```
=====
```
- 1) If you use a unix command-line with environment variables and you are familiar with it, set TERTIUSHOME to be the directory which contains this README. We assume WEKAHOME is set to be the directory which contains the weka.jar file.  
Else you should substitute the name of the directory where this README is where you see \$TERTIUSHOME, and the name of the directory where weka.jar lives where you see \$WEKAHOME.
  - 2) Put the GenericObjectEditor.props file in your home directory, or in the directory from which you wish to start Weka.  
If you are using other associators than Apriori and Tertius, you should edit the file and add those associators.

To run Tertius with the Weka GUIChooser

- ```
-----
```
- 3) If your tertius directory (TERTIUSHOME) is in the same root directory as the weka directory (WEKAHOME), update the class-path for weka by typing:

```
jar umvf $TERTIUSHOME/manifest.mf $WEKAHOME/weka.jar
```

(the manifest.mf file can then be removed).

If your tertius directory is somewhere else, you should first edit the manifest.mf file and correct the path to find the tertius.jar package from the weka directory.

You can then run your Weka GUIChooser as usual (java -jar \$WEKAHOME/weka.jar).

To run Tertius from the command line

- ```
-----
```
- 3) Add \$TERTIUSHOME/tertius.jar to your CLASSPATH environment variable.

You can then run Tertius by typing :

```
java weka.associations.tertius.Tertius <options>
```



**Appendix B**  
**Test results for -U option**



=== Run information ===

Scheme: weka.associations.tertius.Tertius -K 10 -F 0.0 -C 0.0 -N 1.0 -L 4 -G 0 -c 0 -I 0 -A 2 -B 0 -  
V 0 -P 0 -p  
Relation: mushroom-weka.filters.AttributeFilter-V-R6-9  
Instances: 8124  
Attributes: 4  
gill-attachment  
gill-spacing  
gill-size  
gill-color

=== Associator model (full training set) ===

Tertius

=====

1. /\* 0,621585 0,000000 \*/ gill-color = b ==> gill-size = n
2. /\* 0,617140 0,064993 \*/ gill-spacing = c and gill-size = n ==> gill-color = b
3. /\* 0,587838 0,096504 \*/ gill-size = n ==> gill-color = b
4. /\* 0,521576 0,064993 \*/ gill-size = n ==> gill-spacing = w or gill-color = b
5. /\* 0,252476 0,408912 \*/ gill-size = b ==> gill-attachment = a or gill-spacing = w or gill-color = p
6. /\* 0,251664 0,408912 \*/ gill-attachment = f and gill-size = b ==> gill-spacing = w or gill-color = p
7. /\* 0,248940 0,599951 \*/ gill-attachment = f and gill-spacing = c ==> gill-color = b
8. /\* 0,240124 0,424545 \*/ gill-size = b ==> gill-attachment = a or gill-spacing = w or gill-color = w
9. /\* 0,239928 0,599951 \*/ gill-spacing = c ==> gill-attachment = a or gill-color = b
10. /\* 0,233375 0,424545 \*/ gill-attachment = f and gill-size = b ==> gill-spacing = w or gill-color = w

Number of hypotheses considered: 1412

Number of hypotheses explored: 932

Time: 01 min 59 s 012 ms



=== Run information ===

Scheme: weka.associations.tertius.Tertius -K 10 -F 0.0 -C 0.0 -N 1.0 -L 4 -G 0 -c 0 -I 0 -A 2 -B 0 -  
V 0 -P 0 -U -p  
Relation: mushroom-weka.filters.AttributeFilter-V-R6-9  
Instances: 8124  
Attributes: 4  
gill-attachment  
gill-spacing  
gill-size  
gill-color

=== Associator model (full training set) ===

Tertius

=====

1. /\* 0,621585 0,000000 \*/ gill-color = b ==> gill-size = n
2. /\* 0,617140 0,064993 \*/ gill-spacing = c and gill-size = n ==> gill-color = b
3. /\* 0,587838 0,096504 \*/ gill-size = n ==> gill-color = b
4. /\* 0,521576 0,064993 \*/ gill-size = n ==> gill-spacing = w or gill-color = b
5. /\* 0,252476 0,408912 \*/ gill-size = b ==> gill-attachment = a or gill-spacing = w or gill-color = p
6. /\* 0,251664 0,408912 \*/ gill-attachment = f and gill-size = b ==> gill-spacing = w or gill-color = p
7. /\* 0,248940 0,599951 \*/ gill-attachment = f and gill-spacing = c ==> gill-color = b
8. /\* 0,240124 0,424545 \*/ gill-size = b ==> gill-attachment = a or gill-spacing = w or gill-color = w
9. /\* 0,239928 0,599951 \*/ gill-spacing = c ==> gill-attachment = a or gill-color = b
10. /\* 0,233375 0,424545 \*/ gill-attachment = f and gill-size = b ==> gill-spacing = w or gill-color = w

Number of hypotheses considered: 1412

Number of hypotheses explored: 932

Time: 01 min 09 s 659 ms



=== Run information ===

Scheme: weka.associations.tertius.Tertius -K 10 -F 0.0 -C 0.0 -N 1.0 -L 2 -G 0 -c 0 -I 0 -A 2 -B 0 -  
V 0 -P 0 -p  
Relation: soybean  
Instances: 683  
Attributes: 36  
date  
plant-stand  
precip  
temp  
hail  
crop-hist  
area-damaged  
severity  
seed-tmt  
germination  
plant-growth  
leaves  
leafspots-halo  
leafspots-marg  
leafspot-size  
leaf-shread  
leaf-malf  
leaf-mild  
stem  
lodging  
stem-cankers  
canker-lesion  
fruiting-bodies  
external-decay  
mycelium  
int-discolor  
sclerotia  
fruit-pods  
fruit-spots  
seed  
mold-growth  
seed-discolor  
seed-size  
shriveling  
roots  
class

=== Associator model (full training set) ===

Tertius  
=====

1. /\* 0,755110 0,019034 \*/ stem = norm ==> canker-lesion = dna
2. /\* 0,733632 0,000000 \*/ leafspots-marg = dna ==> leafspots-halo = absent
3. /\* 0,733632 0,000000 \*/ leafspots-halo = absent ==> leafspots-marg = dna
4. /\* 0,727634 0,001464 \*/ leafspot-size = dna ==> leafspots-halo = absent
5. /\* 0,727634 0,001464 \*/ leafspot-size = dna ==> leafspots-marg = dna
6. /\* 0,727634 0,001464 \*/ leafspots-halo = absent ==> leafspot-size = dna
7. /\* 0,727634 0,001464 \*/ leafspots-marg = dna ==> leafspot-size = dna
8. /\* 0,719363 0,074671 \*/ canker-lesion = dna ==> stem = norm
9. /\* 0,714597 0,052709 \*/ plant-growth = abnorm ==> fruit-spots = dna
10. /\* 0,706734 0,098097 \*/ fruit-spots = absent ==> stem = norm
11. /\* 0,685707 0,010249 \*/ leafspots-halo = no-yellow-halos ==> leafspots-marg = w-s-marg
12. /\* 0,685216 0,002928 \*/ stem = norm ==> stem-cankers = absent
13. /\* 0,673900 0,017570 \*/ canker-lesion = dna ==> stem-cankers = absent
14. /\* 0,671521 0,016105 \*/ leafspot-size = gt-1/8 ==> leafspots-halo = no-yellow-halos

Number of hypotheses considered: 9890  
Number of hypotheses explored: 4528  
Time: 01 min 05 s 440 ms



=== Run information ===

Scheme: weka.associations.tertius.Tertius -K 10 -F 0.0 -C 0.0 -N 1.0 -L 2 -G 0 -c 0 -I 0 -A 2 -B 0 -  
V 0 -P 0 -U -p  
Relation: soybean  
Instances: 683  
Attributes: 36  
date  
plant-stand  
precip  
temp  
hail  
crop-hist  
area-damaged  
severity  
seed-tmt  
germination  
plant-growth  
leaves  
leafspots-halo  
leafspots-marg  
leafspot-size  
leaf-shread  
leaf-malf  
leaf-mild  
stem  
lodging  
stem-cankers  
canker-lesion  
fruiting-bodies  
external-decay  
mycelium  
int-discolor  
sclerotia  
fruit-pods  
fruit-spots  
seed  
mold-growth  
seed-discolor  
seed-size  
shriveling  
roots  
class

=== Associator model (full training set) ===

Tertius  
=====

1. /\* 0,755110 0,019034 \*/ stem = norm ==> canker-lesion = dna
2. /\* 0,733632 0,000000 \*/ leafspots-marg = dna ==> leafspots-halo = absent
3. /\* 0,733632 0,000000 \*/ leafspots-halo = absent ==> leafspots-marg = dna
4. /\* 0,727634 0,001464 \*/ leafspot-size = dna ==> leafspots-halo = absent
5. /\* 0,727634 0,001464 \*/ leafspot-size = dna ==> leafspots-marg = dna
6. /\* 0,727634 0,001464 \*/ leafspots-halo = absent ==> leafspot-size = dna
7. /\* 0,727634 0,001464 \*/ leafspots-marg = dna ==> leafspot-size = dna
8. /\* 0,719363 0,074671 \*/ canker-lesion = dna ==> stem = norm
9. /\* 0,714597 0,052709 \*/ plant-growth = abnorm ==> fruit-spots = dna
10. /\* 0,706734 0,098097 \*/ fruit-spots = absent ==> stem = norm
11. /\* 0,685707 0,010249 \*/ leafspots-halo = no-yellow-halos ==> leafspots-marg = w-s-marg
12. /\* 0,685216 0,002928 \*/ stem = norm ==> stem-cankers = absent
13. /\* 0,673900 0,017570 \*/ canker-lesion = dna ==> stem-cankers = absent
14. /\* 0,671521 0,016105 \*/ leafspot-size = gt-1/8 ==> leafspots-halo = no-yellow-halos

Number of hypotheses considered: 9890  
Number of hypotheses explored: 4528  
Time: 00 min 19 s 892 ms



=== Run information ===

Scheme: weka.associations.tertius.Tertius -K 10 -F 0.0 -C 0.0 -N 1.0 -L 3 -G 0 -c 0 -I 0 -A 2 -B 0 -  
V 0 -P 0 -p  
Relation: soybean  
Instances: 683  
Attributes: 36  
date  
plant-stand  
precip  
temp  
hail  
crop-hist  
area-damaged  
severity  
seed-tmt  
germination  
plant-growth  
leaves  
leafspots-halo  
leafspots-marg  
leafspot-size  
leaf-shread  
leaf-malf  
leaf-mild  
stem  
lodging  
stem-cankers  
canker-lesion  
fruiting-bodies  
external-decay  
mycelium  
int-discolor  
sclerotia  
fruit-pods  
fruit-spots  
seed  
mold-growth  
seed-discolor  
seed-size  
shriveling  
roots  
class

=== Associator model (full training set) ===

Tertius  
=====

1. /\* 0,801047 0,043924 \*/ stem-cankers = absent and fruit-spots = absent ==> stem = norm
2. /\* 0,797104 0,051245 \*/ stem-cankers = absent and int-discolor = none ==> stem = norm
3. /\* 0,796114 0,048316 \*/ fruiting-bodies = absent and fruit-spots = absent ==> stem = norm
4. /\* 0,792484 0,035139 \*/ canker-lesion = dna and fruit-spots = absent ==> stem = norm
5. /\* 0,791883 0,048316 \*/ fruit-spots = absent ==> stem = norm or class = brown-spot
6. /\* 0,789718 0,036603 \*/ canker-lesion = dna and fruiting-bodies = absent ==> stem = norm
7. /\* 0,784201 0,039531 \*/ canker-lesion = dna and int-discolor = none ==> stem = norm
8. /\* 0,777930 0,020498 \*/ stem-cankers = absent and fruit-spots = absent ==> canker-lesion = dna
9. /\* 0,775079 0,010249 \*/ leaves = abnorm and stem = norm ==> canker-lesion = dna
10. /\* 0,773579 0,002928 \*/ stem = norm ==> canker-lesion = dna or class = purple-seed-stain

Number of hypotheses considered: 244429  
Number of hypotheses explored: 77571  
Time: 28 min 21 s 069 ms



=== Run information ===

Scheme: weka.associations.tertius.Tertius -K 10 -F 0.0 -C 0.0 -N 1.0 -L 3 -G 0 -c 0 -I 0 -A 2 -B 0 -  
V 0 -P 0 -U -p  
Relation: soybean  
Instances: 683  
Attributes: 36  
date  
plant-stand  
precip  
temp  
hail  
crop-hist  
area-damaged  
severity  
seed-tmt  
germination  
plant-growth  
leaves  
leafspots-halo  
leafspots-marg  
leafspot-size  
leaf-shread  
leaf-malf  
leaf-mild  
stem  
lodging  
stem-cankers  
canker-lesion  
fruiting-bodies  
external-decay  
mycelium  
int-discolor  
sclerotia  
fruit-pods  
fruit-spots  
seed  
mold-growth  
seed-discolor  
seed-size  
shriveling  
roots  
class

=== Associator model (full training set) ===

Tertius  
=====

1. /\* 0,759605 0,052709 \*/ leaves = abnorm and canker-lesion = dna ==> stem = norm
2. /\* 0,755110 0,019034 \*/ stem = norm ==> canker-lesion = dna
3. /\* 0,733632 0,000000 \*/ leafspots-marg = dna ==> leafspots-halo = absent
4. /\* 0,727634 0,001464 \*/ leafspot-size = dna ==> leafspots-halo = absent
5. /\* 0,727634 0,001464 \*/ leafspot-size = dna ==> leafspots-marg = dna
6. /\* 0,719363 0,074671 \*/ canker-lesion = dna ==> stem = norm
7. /\* 0,706734 0,098097 \*/ fruit-spots = absent ==> stem = norm
8. /\* 0,689661 0,096633 \*/ leaves = abnorm and fruit-spots = absent ==> stem = norm
9. /\* 0,685707 0,010249 \*/ leafspots-halo = no-yellow-halos ==> leafspots-marg = w-s-marg
10. /\* 0,685216 0,002928 \*/ stem = norm ==> stem-cankers = absent
11. /\* 0,683255 0,103953 \*/ leaves = abnorm and stem-cankers = absent ==> stem = norm

Number of hypotheses considered: 25572  
Number of hypotheses explored: 12461  
Time: 01 min 24 s 254 ms

Not enough memory to continue the search



**Appendix C**  
**Test results for –A option**



=== Run information ===

Scheme: weka.associations.tertius.Tertius -K 10 -F 0.0 -C 0.0 -N 1.0 -L 4 -G 0 -c 0 -I 0 -A 1 -B 0 -  
V 0 -P 0 -p  
Relation: weather.symbolic  
Instances: 14  
Attributes: 5  
outlook  
temperature  
humidity  
windy  
play  
=== Associator model (full training set) ===

Tertius  
=====

1. /\* 0,633754 0,071429 \*/ play = yes ==> outlook = overcast or humidity = normal
2. /\* 0,607625 0,000000 \*/ humidity = normal ==> temperature = cool or play = yes
3. /\* 0,607625 0,000000 \*/ temperature = cool ==> humidity = normal
4. /\* 0,594071 0,214286 \*/ humidity = normal ==> temperature = cool
5. /\* 0,590214 0,000000 \*/ outlook = sunny and humidity = high ==> play = no
6. /\* 0,555556 0,000000 \*/ play = no ==> outlook = sunny or windy = TRUE
7. /\* 0,486606 0,000000 \*/ humidity = normal ==> outlook = rainy or play = yes
8. /\* 0,486606 0,000000 \*/ outlook = sunny and play = no ==> humidity = high
9. /\* 0,469374 0,000000 \*/ outlook = overcast ==> play = yes
10. /\* 0,469374 0,000000 \*/ temperature = hot ==> outlook = overcast or humidity = high
11. /\* 0,469374 0,000000 \*/ temperature = hot and play = yes ==> outlook = overcast
12. /\* 0,469374 0,000000 \*/ temperature = hot ==> outlook = overcast or play = no
13. /\* 0,469374 0,000000 \*/ outlook = overcast ==> temperature = hot or windy = TRUE
14. /\* 0,469374 0,000000 \*/ outlook = overcast and windy = FALSE ==> temperature = hot
15. /\* 0,469374 0,000000 \*/ play = no ==> humidity = high or windy = TRUE
16. /\* 0,469374 0,000000 \*/ humidity = high and play = no ==> outlook = sunny or temperature = mild
17. /\* 0,469374 0,000000 \*/ temperature = mild and play = yes ==> outlook = rainy or windy = TRUE
18. /\* 0,469374 0,000000 \*/ outlook = sunny ==> temperature = cool or windy = TRUE or play = no
19. /\* 0,467119 0,357143 \*/ play = yes ==> outlook = overcast
20. /\* 0,458333 0,071429 \*/ play = yes ==> outlook = overcast or windy = FALSE
21. /\* 0,458333 0,071429 \*/ humidity = high and play = no ==> outlook = sunny
22. /\* 0,439100 0,071429 \*/ play = no ==> humidity = high
23. /\* 0,439100 0,071429 \*/ humidity = high ==> temperature = mild or play = no
24. /\* 0,439100 0,071429 \*/ humidity = high ==> outlook = sunny or temperature = mild

Number of hypotheses considered: 1730  
Number of hypotheses explored: 921  
Time: 00 min 00 s 487 ms



=== Run information ===

Scheme: weka.associations.tertius.Tertius -K 10 -F 0.0 -C 0.0 -N 1.0 -L 4 -G 0 -c 0 -I 0 -A 2 -B 0 -  
V 0 -P 0 -p  
Relation: weather.symbolic  
Instances: 14  
Attributes: 5  
outlook  
temperature  
humidity  
windy  
play

=== Associator model (full training set) ===

Tertius

=====

1. /\* 0,633754 0,071429 \*/ play = yes ==> outlook = overcast or humidity = normal
2. /\* 0,607625 0,000000 \*/ humidity = normal ==> temperature = cool or play = yes
3. /\* 0,607625 0,000000 \*/ temperature = cool ==> humidity = normal
4. /\* 0,594071 0,214286 \*/ humidity = normal ==> temperature = cool
5. /\* 0,590214 0,000000 \*/ outlook = sunny and humidity = high ==> play = no
6. /\* 0,555556 0,000000 \*/ play = no ==> outlook = sunny or windy = TRUE
7. /\* 0,486606 0,000000 \*/ humidity = normal ==> outlook = rainy or play = yes
8. /\* 0,486606 0,000000 \*/ outlook = sunny and play = no ==> humidity = high
9. /\* 0,469374 0,000000 \*/ outlook = overcast ==> play = yes
10. /\* 0,469374 0,000000 \*/ temperature = hot ==> outlook = overcast or humidity = high
11. /\* 0,469374 0,000000 \*/ temperature = hot and play = yes ==> outlook = overcast
12. /\* 0,469374 0,000000 \*/ temperature = hot ==> outlook = overcast or play = no
13. /\* 0,469374 0,000000 \*/ outlook = overcast ==> temperature = hot or windy = TRUE
14. /\* 0,469374 0,000000 \*/ outlook = overcast and windy = FALSE ==> temperature = hot
15. /\* 0,469374 0,000000 \*/ play = no ==> humidity = high or windy = TRUE
16. /\* 0,469374 0,000000 \*/ humidity = high and play = no ==> outlook = sunny or temperature = mild
17. /\* 0,469374 0,000000 \*/ temperature = mild and play = yes ==> outlook = rainy or windy = TRUE
18. /\* 0,469374 0,000000 \*/ outlook = sunny ==> temperature = cool or windy = TRUE or play = no
19. /\* 0,467119 0,357143 \*/ play = yes ==> outlook = overcast
20. /\* 0,458333 0,071429 \*/ play = yes ==> outlook = overcast or windy = FALSE
21. /\* 0,458333 0,071429 \*/ humidity = high and play = no ==> outlook = sunny
22. /\* 0,439100 0,071429 \*/ play = no ==> humidity = high
23. /\* 0,439100 0,071429 \*/ humidity = high ==> temperature = mild or play = no
24. /\* 0,439100 0,071429 \*/ humidity = high ==> outlook = sunny or temperature = mild

Number of hypotheses considered: 1730

Number of hypotheses explored: 642

Time: 00 min 00 s 465 ms



=== Run information ===

Scheme: weka.associations.tertius.Tertius -K 10 -F 0.0 -C 0.0 -N 1.0 -L 4 -G 0 -c 0 -I 0 -A 3 -B 0 -  
V 0 -P 0 -p  
Relation: weather.symbolic  
Instances: 14  
Attributes: 5  
outlook  
temperature  
humidity  
windy  
play

=== Associator model (full training set) ===

Tertius  
=====

1. /\* 0,633754 0,071429 \*/ play = yes ==> outlook = overcast or humidity = normal
2. /\* 0,607625 0,000000 \*/ humidity = normal ==> temperature = cool or play = yes
3. /\* 0,607625 0,000000 \*/ temperature = cool ==> humidity = normal
4. /\* 0,594071 0,214286 \*/ humidity = normal ==> temperature = cool
5. /\* 0,590214 0,000000 \*/ outlook = sunny and humidity = high ==> play = no
6. /\* 0,555556 0,000000 \*/ play = no ==> outlook = sunny or windy = TRUE
7. /\* 0,486606 0,000000 \*/ humidity = normal ==> outlook = rainy or play = yes
8. /\* 0,486606 0,000000 \*/ outlook = sunny and play = no ==> humidity = high
9. /\* 0,469374 0,000000 \*/ outlook = overcast ==> play = yes
10. /\* 0,469374 0,000000 \*/ temperature = hot ==> outlook = overcast or humidity = high
11. /\* 0,469374 0,000000 \*/ temperature = hot and play = yes ==> outlook = overcast
12. /\* 0,469374 0,000000 \*/ temperature = hot ==> outlook = overcast or play = no
13. /\* 0,469374 0,000000 \*/ outlook = overcast ==> temperature = hot or windy = TRUE
14. /\* 0,469374 0,000000 \*/ outlook = overcast and windy = FALSE ==> temperature = hot
15. /\* 0,469374 0,000000 \*/ play = no ==> humidity = high or windy = TRUE
16. /\* 0,469374 0,000000 \*/ humidity = high and play = no ==> outlook = sunny or temperature = mild
17. /\* 0,469374 0,000000 \*/ temperature = mild and play = yes ==> outlook = rainy or windy = TRUE
18. /\* 0,469374 0,000000 \*/ outlook = sunny ==> temperature = cool or windy = TRUE or play = no
19. /\* 0,467119 0,357143 \*/ play = yes ==> outlook = overcast
20. /\* 0,458333 0,071429 \*/ play = yes ==> outlook = overcast or windy = FALSE
21. /\* 0,458333 0,071429 \*/ humidity = high and play = no ==> outlook = sunny
22. /\* 0,439100 0,071429 \*/ play = no ==> humidity = high
23. /\* 0,439100 0,071429 \*/ humidity = high ==> temperature = mild or play = no
24. /\* 0,439100 0,071429 \*/ humidity = high ==> outlook = sunny or temperature = mild

Number of hypotheses considered: 1730  
Number of hypotheses explored: 643  
Time: 00 min 00 s 460 ms