1 Startup


2 Walkthrough: zoo data in Weka

1. Start Weka then start the Explorer.
2. Open zoo.arff
3. The Preprocess tab allows you to examine basic properties of the data’s attributes and exclude some attributes from modelling. Click on fins then legs to see the difference.
4. Uncheck the box beside legs and click Apply Filters.
   Take care to always apply filters after you set them.
5. Choose the Associate tab and click Start.
   Weka generates and displays 10 association rules using the Apriori algorithm.
6. You can customise the algorithm’s parameters by clicking on the line beginning Apriori -N 10.

2.1 Questions

1. What do the two integers in each rule represent?
2. How does Weka choose which 10 rules to display?
3. Why did we disable the legs attribute?
4. Describe in your own words what the first rule tells you.

3 Walkthrough: zoo data in WizWhy

1. Start WizWhy
2. In the Basic Data tab (which should be open) choose to open data of type ASCII then click the button.
3. Open zoo.txt from the fileselector.
4. Choose Delimited for Record Type. In Field Delimiters select Comma.
5. Choose First record for fields names.
6. Click Parse.
   WizWhy will guess what type of data is in each column: QLT for qualitative (discrete), QNT for quantitative (continuous) and D-M-Y for dates. For this file all data are qualitative (QLT).
7. For the **animal** column, select **Do Not Import Column**.
   This column is not useful for prediction.
   It turns out that leaving it in does not affect results; WizWhy simply does not use it. We are only excluding it for tutorial purposes.

8. Click **OK**.
   The field grid will be populated according to your selections.

9. Scroll to the bottom of the field grid and select **Type** as the **Dependent Variable**.

10. Deselect **Analyze the Dependent Variable as Boolean**.
    Our dependent variable is categorical, not boolean. WizWhy gives you the option of treating it as boolean on a specific class — say, we could have it generate rules on ‘Yes, type is mammal’ or ‘No, type is not mammal’. In this case we don’t want to do that.

11. Change to the **Rule Parameters** tab.
    Here you could configure rule selection if desired. If the dependent variable was boolean, you could also configure the confidence (which WizWhy calls Probability).

12. Click **Issue Rules** on the toolbar.

13. Expand the tree down the left hand side of the window if it is collapsed.

14. View the **If-Then Rules**.
    These are like the apriori association rules from Weka. For each rule we have the conditions and the implication, its probability (confidence), how many records it occurs in (support), WizWhy’s summary of its significance and some positive and negative examples from the data.

15. View the **If-and-only-if Rules**.
    An If-then rule states that if some conditions are met then some implication is true, with a given probability. An If-and-only-if rule states that too, but also states that if those conditions are not met, then the implication is false. In other words, If-then rules establish sufficient conditions for an implication; If-and-only-if rules establish necessary and sufficient conditions.
    The example from the **zoo** data tells us that if an animal produces milk then it’s a mammal; we already knew this from the equivalent If-then rule. It also tells us that the only way to be a classified as a mammal is to produce milk.
4 Walkthrough: iris data

1. Load the data from iris.txt and configure. The four measurements are quantitative data.

2. Click Issue rules and view the Unexpected cases report.
   This shows cases where WizWhy would have made an incorrect prediction based on the rules it
   derived. The right-hand pane gives the (incorrect) prediction, level of confidence in it, and a list
   of the rules used to generate that prediction.

4.1 Discussion questions

1. Describe the If-and-only-if rule that was derived.

2. How many records were incorrectly classified (use the Predicted Value selector in Unexpected
cases)? What proportion of that number were virginicas? What does this say about the nature
of the rules that WizWhy has extracted?

5 Walkthrough: census data

1. Load the data from census.txt.

2. Select Analyze the Dependent Variable as Boolean if it is unselected.

3. In Rule Parameters, set the Predicted Value to >50k.

4. Click Issue rules and view the Unexpected Rules report.
   An Unexpected rule is a rule with two or more conditions, each of which is a basic rule. What
   makes it unexpected is that the unexpected rule’s support or confidence is markedly different from
   the supports or confidences of the basic rules. WizWhy finds unexpected rules by calculating
   the expected support and confidence of the aggregate rule based on those of the basic rules, and
   highlights rules that are not consistent with this.

5. View the Summary Report.
   Examine the number of misses and false alarms, and success rates.

6. Return to the Main Window. Open Error Costs.
   Here we can set the relative costs of errors.
   For binary attribute selection: stipulate whether it’s more ‘expensive’ to have a ‘miss’ than a
   ‘false alarm’.
   For multivalue attribute selection: stipulate a matrix of costs for all the possible errors. This
   looks just like a confusion matrix, except that the major diagonal is disabled because there is no
   cost for a correct answer!

5.1 Questions

1. What type of data do the columns Capital gain and Work hours per week contain? Can Apriori
   handle this kind of data?

2. Explain how the unexpected rule is actually unexpected. What would you expect the support and
   confidence to be (roughly)?

3. With regard to the Summary Report, explain what a ‘miss’ is.
5.2 Discussion questions

1. Assume that you want to market $2000 investment packages to high income earners, and that your marketing effort costs $450 per prospective customer for time and materials. In terms of this business what’s more expensive, a miss or a false alarm (give assumptions)? Set an error cost matrix to reflect this. Give an explanation of what you have done and why, at a level that a businessperson could understand. Justify your error cost matrix in terms of how it improved your performance.

2. Look at the unexpected rules and if-and-only-if rules. Which of these are insights?

3. Discuss the quality of the generated if-then rules.

4. Choose an unexpected case and describe why it was incorrectly classified.

5.3 Homework

What settings would you use to have WizWhy examine the characteristics of divorced people? From the data, write down the typical characteristics of a divorced person.

Submit to mbeaureg@it.uts.edu.au by Monday. No attachments, please.
6 Additional material: algorithm walkthroughs

6.1 apriorigen

Listing 1: aprioriGen

```python
def aprioriGen(size, source_itemsets):
    # Return all the candidate frequent itemsets of a certain size
    # we will store our candidates in this variable
    candidates = Set()

    for set1 in source_itemsets:
        for set2 in source_itemsets:
            # compare all source itemsets against all other itemsets
            if set1 == set2:
                # don't compare a source itemset with itself!
                continue

            if numSameElements(set1, set2) == size - 2:
                # these two sets have (size - 2) elements in common
                # and, therefore, 2 elements are different
                # so the union of the sets will be (size - 2 + 2) in length
                # making this a candidate frequent itemset of size (size).
                candidate = Set(set1 | set2)
                candidates.add(candidate)

    return candidates
```

What would this function return for a

1. source of \{\{C\}, \{C++\}, \{Perl\}, \{Python\}\} and size 2;
2. source of \{\{C++, Perl\}, \{C, C++\}, \{C, Perl\}\} and size 3;
3. source of \{\{C, C++, Perl\}\} and size 4?
6.2 apriori

Listing 2: apriori

```python
def apriori(initial_candidates, records, min_support):
    # Find frequent itemsets
    # storage for our frequent itemsets
    largeItemsets = Set()

    consider_size = 1
    candidates = initial_candidates

    while len(candidates):
        # loop until there are no more itemsets to consider
        # we start by considering sets of size 1

        for itemset in candidates:
            if support(itemset, records) >= min_support:
                largeItemsets.add(itemset)

        # make a set of large itemsets of the current size
        large_itemsets_this_size = Set([itemset for itemset in largeItemsets
                                         if len(itemset) == consider_size])

        # get a set of candidate itemsets that are size + 1
        consider_size += 1
        candidates = aprioriGen(consider_size, large_itemsets_this_size)

    return largeItemsets
```

For an input of \{\{C\}, \{C++\}, \{Perl\}, \{Python\}\} and \(s = 0.5\), what frequent itemsets are generated?
6.3 rulegen

Listing 3: rulegen

```python
def rulegen(records, items, frequent_itemsets, min_support, min_confidence):
    # Generate association rules from frequent itemsets

    # storage for generated rules
    rules = []

    for itemset in frequent_itemsets:
        for potential_LHS in frequent_itemsets:
            # we want to use all the subsets of our frequent itemset as potential
            # left-hand sides for the rule.
            # by definition, any subset of a frequent itemset is also frequent
            # so all the subsets will be in our list of frequent itemsets;
            # this saves us having to recalculate the subsets, which is bothersome

            if itemset != potential_LHS and potential_LHS.issubset(itemset):
                rule_support = support(itemset, records)
                confidence = rule_support / support(potential_LHS, records)

                if confidence >= min_confidence:
                    # we will accept this rule
                    # obviously, the RHS is everything in the itemset
                    # that's not on the LHS
                    RHS = itemset - potential_LHS

                    rules.append((potential_LHS, RHS, rule_support, confidence))

    return rules
```

For $s = 0.5$ and $\alpha = 0.5$ what rules are generated? What is their support and confidence?