Data Warehousing and Data Mining Lab Report

Jeevesh Juneja

November 2021

Contents

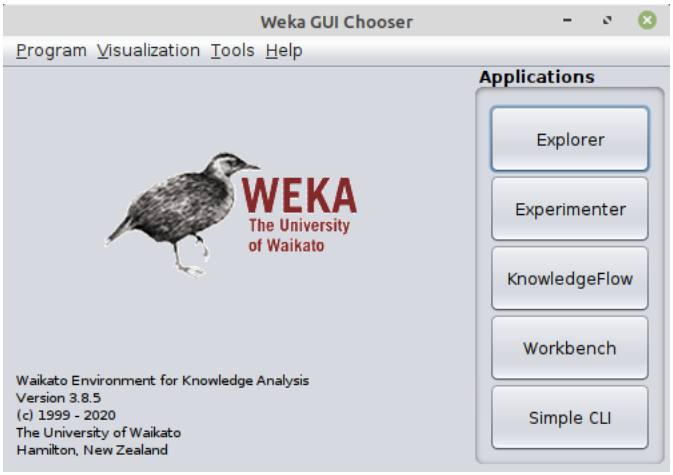


Figure 1: The initially launched window for WEKA listing the various applications available.

1 Experiment-1: Intro to WEKA

1.1 Installation

1. Download the software from here.
2. Unzip the downloaded file.
3. **cd** into the unzipped weka repo, and run:

```
bash weka.sh
```

This will launch the WEKA tool and we can see Explorer, Experimenter, KnowledgeFlow, Workbench and Simple CLI options. Figure 1 shows the window that is launched.

2 Experiment-2: Intro To WEKA tool Features

2.1 Explorer App

First we explore, the **Explorer** application. We see in Figure 2, the interface for generating random data. The random data consists of two types of attributes, numerical, and class-based. The number of classes, number of examples, number of attributes can be specified using the **numClasses**, **numExamples**, **numAttributes** variables, respectively.

Additionally, during generation, a random list of decision rules is also generated. During generation of random data, if any of the random samples fails to be classified based on the random list of decision rules, a new rule is generated

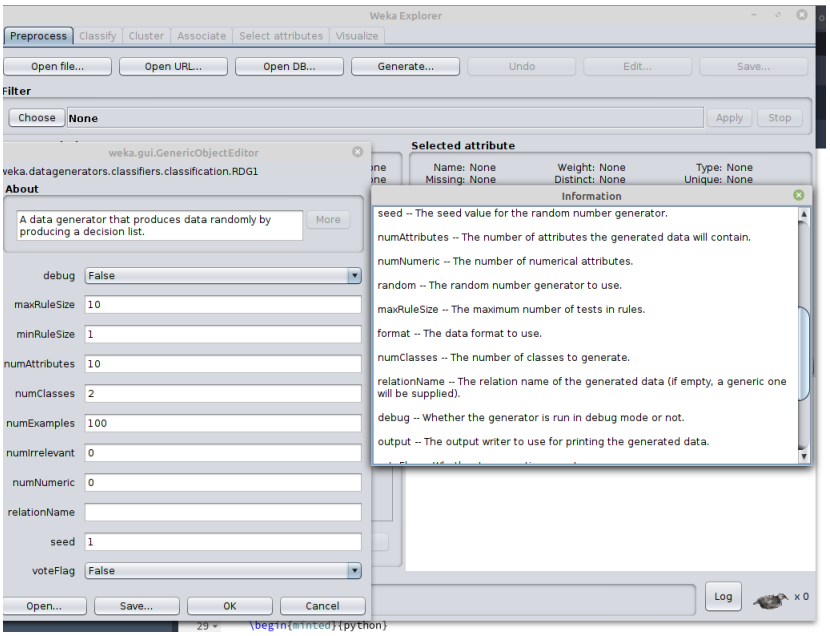


Figure 2: Interface for generating random data in Weka. This is obtained by clicking on the **Generate** button on the **Preprocess** tab

and added to the random list. We can specify the maximum and minimum number of tests in a single rule using the parameters **maxRuleSize** and **minRuleSize**, respectively. These are equivalent to maximum and minimum value for the depth of decision tree. The option **-V** switches on voting, which means that at the end of the generation all instances are reclassified to the class value that is supported by the most rules.

We can also load data from files in various formats like **arff**, **json**, **libsvm**, **csv** etc. using the "Open file" button in the second row from top in Figure 2.

2.2 Experimenter App

The **Experimenter** app allows you to configure and run experiments on data, loaded from some file. We show the interface in Figure 3. The experiment configuration can be either loaded from a new **.exp** file[1] or we can create a new experiment configuration, using the options below. We need to specify the dataset file[2] in the **Datasets** section, the algorithm in the **Algorithms** section.

Other parameters like Experiment type of regression or classification, and the number of times to repeat the experiment can also be specified. The results are stored in a pre-created file, specified by the user in the "Results Destination"

[1] click on open button in top row in Figure 3

[2] must be a file in one of the formats previously specified

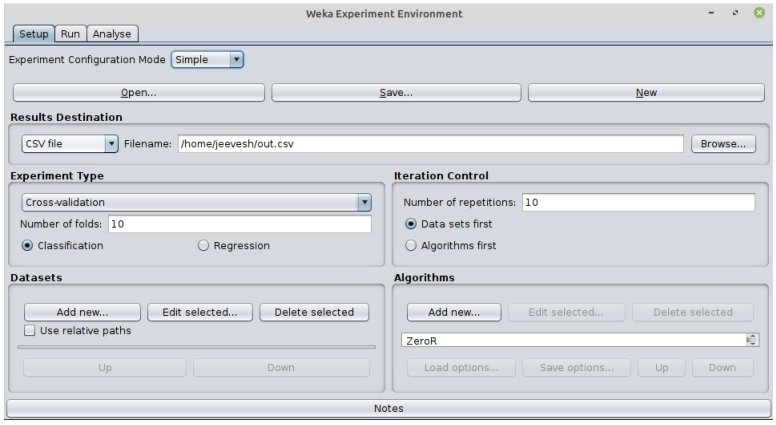


Figure 3: The Weka experimenter interface. The analyse tab can be used to analyse the files in which outputs are stored. It allows us to compare various algorithms.

section. After completing setup, we switch to the Run tab, and click on the "start" button to start the experiment. The results of each repeated experiment will be logged in the output file specified, and can be checked once the run has finished.

2.3 KnowledgeFlow App

Next, we see the **KnowledgeFlow** app in Figure 4. The KnowledgeFlow presents an alternate view for processing data. It presents a *data-flow* inspired interface to WEKA. The user can select WEKA steps from the palette(*Design panel*) on the left, place them on a layout canvas and connect them together in order to form a knowledge flow for processing and analyzing data.

Various components can be picked from the design panel. Left clicking anywhere in the layout canvas will place the component there. The inputs to each component can be configured by right-clicking the component and selecting *Configure*. Each component can produce various outputs. Each of these outputs can be connected to other components. To pipe output of component 1 into component 2, right click on component 1 and under the *Connections* section in pop-up menu, select the output you want to pipe into component 2. An elastic line will appear, left clicking on component 2 will pipe the selected output of component 1 into 2. To view the final output, we need to right click on the final *Visualization* element at last, and click on *Show Results*.

2.4 Other Apps

The workbench app is introduced in Weka 3.8 , and provides a combined interface to all the previously specified application and the Simple CLI provides a

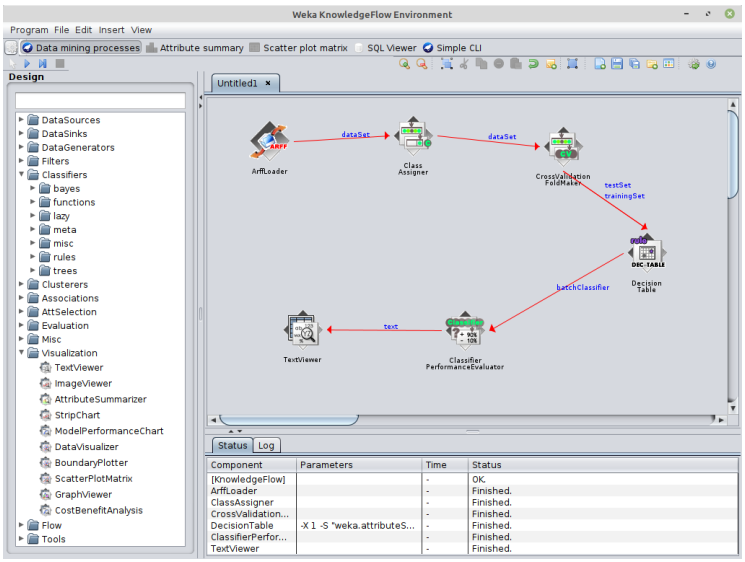


Figure 4: The Weka KnowledgeFlow interface. The Class Assigner is used to choose which class is the label class. To run the flow, we click on the play button right on top of design panel. Status and logs are displayed at bottom.

command-line interface for doing all the previously specified things. In backend, all the GUI above, invoke this CLI.

3 Classification of Data Mining Techniques

Data mining techniques can be broadly divided into the following classes:

- **Classification:** This data mining technique involves grouping data(instances) into different classes. The classes are specified by some "label" attribute given to each instance in the data.
- **Clustering:** This data mining technique again involves grouping data(instance) into different classes. The classes aren't specified by any label though, and are to be inferred using the ambient and internal structure of the data.
- **Regression:** This process involves fitting an analytical model to infer the relation between two or more related numeric attributes.
- **Association Rules:** Association rule are if-then statements inferred from data statistics of the database. It is like co-variance of attributes and encoding them into if-then statements.
- **Outlier Detection:** This technique corresponds to finding elements in the dataset which do not match some expected pattern or behavior of the data. It is also known as Outlier Ananlysis.

```
%
@relation breast-cancer
@attribute age {'10-19','20-29','30-39','40-49','50-59','60-69','70-79','80-89','90-99'}
@attribute menopause {'lt40','ge40','premeno'}
@attribute tumor-size {'0-4','5-9','10-14','15-19','20-24','25-29','30-34','35-39','40-44','45-49','50-54','55-59'}
@attribute inv-nodes {'0-2','3-5','6-8','9-11','12-14','15-17','18-20','21-23','24-26','27-29','30-32','33-35','36-39'}
@attribute node-caps {'yes','no'}
@attribute deg-malig {'1','2','3'}
@attribute breast {'left','right'}
@attribute breast-quad {'left_up','left_low','right_up','right_low','central'}
@attribute 'irradiat' {'yes','no'}
@attribute 'Class' {'no-recurrence-events','recurrence-events'}
@data
'40-49','premeno','15-19','0-2','yes','3','right','left_up','no','recurrence-events'
'50-59','ge40','15-19','0-2','no','1','right','central','no','no-recurrence-events'
'50-59','ge40','35-39','0-2','no','2','left','left_low','no','recurrence-events'
'40-49','premeno','35-39','0-2','yes','3','right','left_low','yes','no-recurrence-events'
'40-49','premeno','30-34','3-5','yes','2','left','right_up','no','recurrence-events'
'50-59','premeno','25-29','3-5','no','2','right','left_up','yes','no-recurrence-events'
'50-59','ge40','40-44','0-2','no','3','left','left_up','no','no-recurrence-events'
'40-49','premeno','10-14','0-2','no','2','left','left_up','no','no-recurrence-events'
'40-49','premeno','0-4','0-2','no','2','right','right_low','no','no-recurrence-events'
'40-49','ge40','40-44','15-17','yes','2','right','left_up','yes','no-recurrence-events'
'50-59','premeno','25-29','0-2','no','2','left','left_low','no','no-recurrence-events'
'60-69','ge40','15-19','0-2','no','2','right','left_up','no','no-recurrence-events'
```

Figure 5: A sample excerpt from an ARFF file.

- **Sequential Patterns:** It is the process of inferring interesting patterns, subsequences etc. from sequential data, for e.g., auto-regressive data.
- **Prediction:** It is the final step that involves predicting labels for new data instances, or future expected instances in data, or the cluster to which future data is expected to belong to/come from etc.

4 Experiment 4: ARFF

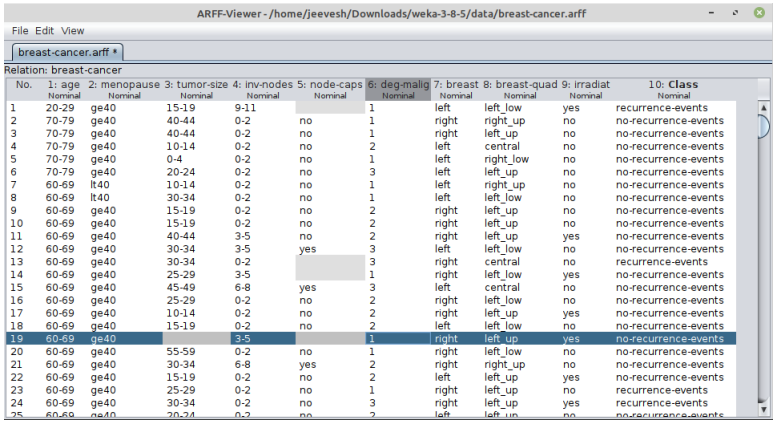
ARFF standing for Attribute Relation File Format, is an ASCII text file desciribing a list of instances sharing a set of attributes. Sample `.arff` files are included in the `data` directory in the unzipped `weka-x-x-x/` directory. An excerpt from `breast-cancer.arff` is provided in Figure 5.

4.1 File Structure and Examples

The entire data itself constitutes a relation on the Cartesian product of the values of each of the attributes. The name of the relation is specified at the top of file using the `@relation` declaration. This is followed by the declarations of attributes. These two together consist the *Header Section*, which is followed by `@data`, marking the beginning of data of the file.

The list of values for categorical attributes can be specified in the declaration as is done in Figure 5. The name of the attribute can be with or without quotes. Weka supports attributes of numeric, string, date, in addition to the categorical type. Here are some examples:

```
@attribute passenger_numbers numeric
@attribute Date date 'yyyy-MM-dd'
@data
112,1949-01-01
118,1949-02-01
```



No.	1: age	2: menopause	3: tumor-size	4: inv-nodes	5: node-caps	6: deg-malig	7: breast	8: breast-quad	9: irradiat	10: Class
	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal
1	20-29	ge40	15-19	9-11		1	left	left_low	yes	recurrence-events
2	70-79	ge40	40-44	0-2	no	1	right	right_up	no	no-recurrence-events
3	70-79	ge40	40-44	0-2	no	1	right	left_up	no	no-recurrence-events
4	70-79	ge40	10-14	0-2	no	2	left	central	no	no-recurrence-events
5	70-79	ge40	0-4	0-2	no	1	left	right_low	no	no-recurrence-events
6	70-79	ge40	20-24	0-2	no	3	left	left_up	no	no-recurrence-events
7	60-69	lt40	10-14	0-2	no	1	left	right_up	no	no-recurrence-events
8	60-69	lt40	30-34	0-2	no	1	left	left_low	no	no-recurrence-events
9	60-69	ge40	15-19	0-2	no	2	right	left_up	no	no-recurrence-events
10	60-69	ge40	15-19	0-2	no	2	right	left_up	no	no-recurrence-events
11	60-69	ge40	40-44	3-5	no	2	right	left_up	yes	no-recurrence-events
12	60-69	ge40	30-34	3-5	yes	3	left	left_low	no	no-recurrence-events
13	60-69	ge40	30-34	0-2		3	right	central	no	recurrence-events
14	60-69	ge40	25-29	3-5		1	right	left_low	yes	no-recurrence-events
15	60-69	ge40	45-49	6-8	yes	3	left	central	no	no-recurrence-events
16	60-69	ge40	25-29	0-2	no	2	right	left_low	no	no-recurrence-events
17	60-69	ge40	10-14	0-2	no	2	right	left_up	yes	no-recurrence-events
18	60-69	ge40	15-19	0-2	no	2	left	left_low	no	no-recurrence-events
19	60-69	ge40		3-5		1	right	left_up	yes	no-recurrence-events
20	60-69	ge40	55-59	0-2	no	1	right	left_low	no	no-recurrence-events
21	60-69	ge40	30-34	6-8	yes	2	right	right_up	no	no-recurrence-events
22	60-69	ge40	15-19	0-2	no	2	left	left_up	yes	no-recurrence-events
23	60-69	ge40	25-29	0-2	no	1	right	left_up	no	recurrence-events
24	60-69	ge40	30-34	0-2	no	3	right	left_up	yes	recurrence-events

Figure 6: The breast-cancer data from before, visualised and edited in the ArffViewer. The blue line is the currently selected one. Multiple lines can be selected and modified at once.

4.2 ArffViewer

The Weka app also provides an **ArffViewer** for viewing arff files in an excel-like tabular format. The ArffViewer can be accessed via `Tools` → `ArffViewer`, in the initial Weka window showing the 5 different apps(Figure 1). Figure 6 shows how the viewer presents the file. Missing values are shown by grayed out boxes. Clicking on any of the boxes will open a drop down menu for categorical data type attributes and allows us to edit them.

We can also limit the attributes that are visible by going to *View* → *Attributes*, or filter the rows by constraints on values of some attributes by going to *View* → *Values*. The *Edit* option in the top bar can be used to delete multiple rows at once(first select them using `shift`+arrow keys), or to rename attributes, or for things like sorting data by some attribute's values.

5 Experiment 5: Supermarket Dataset

This dataset is available in `weka-x-x-x/data/supermarket.arff`. Each column corresponds to a department, and each row to a supermarket. If a supermarket has a department there is a "t"(for true) in that column and otherwise the column is empty. If there are ≥ 100 departments in a supermarket, it is classified as a "high"-end supermarket, otherwise "low".

5.1 Preprocessing & Visualisation

Figure 7 shows some sample rows and columns from the data, where the greyed out boxes have missing value, which means that the department is missing from the supermarket. Figure 8 shows the distribution of low and high-end

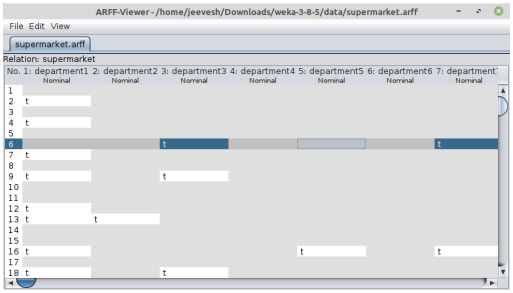


Figure 7: Few rows and columns of supermarket dataset as visualised in **ArffViewer**.

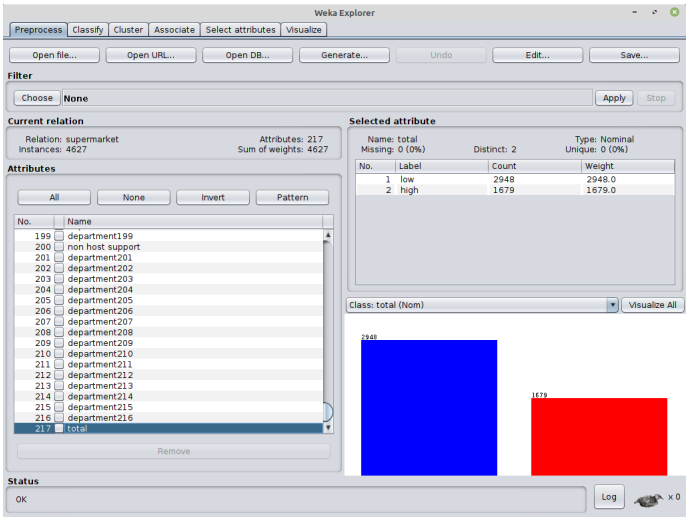


Figure 8: Supermarket Dataset statistics visualized in the *Explorer* app.

supermarkets. We see a class imbalance, but note that each supermarket is either low-end or high-end.

5.2 Classification & Visualisation

We use Decision Table classifier for classifying the instances. We can select it from `classifiers/rules/` after clicking on the **Choose** button. The results for cross-validation are presented in Figure 9. The ROC characteristic curve for the **low** class is shown in Figure 10.

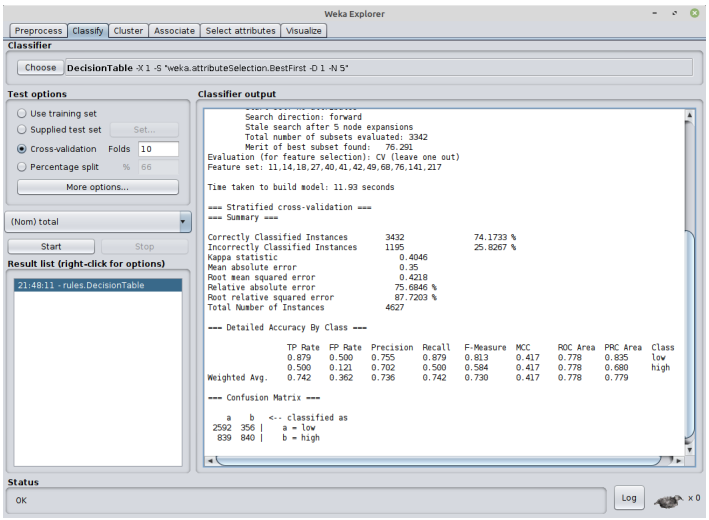


Figure 9: Results on supermarket dataset. The decision table classifier is able to classify $\approx 75\%$ of the supermarkets correctly.

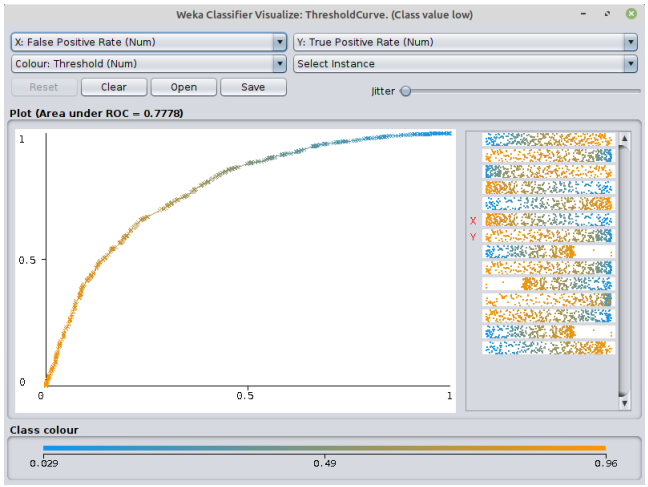


Figure 10: ROC curve the **low** class of supermarket dataset. On X-axis, the threshold varies and on Y-axis the true positive rate for **low** instances of data.

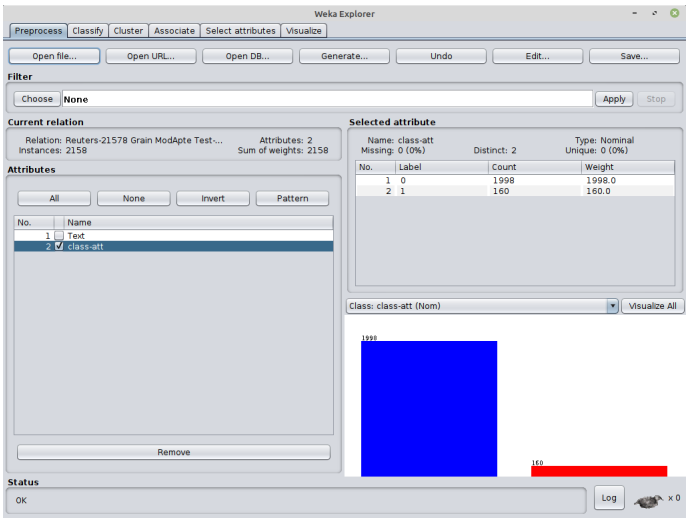


Figure 11: Label distribution for the whole ReutersGrain dataset, made during the first step.

6 Experiment 6: Agriculture Dataset

We use the Reuters-21578 dataset. It is a text classification dataset where the texts correspond to news articles and they are binary classified based on whether they talk about grain or not. This is available in the `ReutersGrain-train.arff` file in the `weka-x-x-x/data/` folder.

6.1 Preprocessing & Visualisation

Firstly we merge together the train and test files together using the command :

```
java weka.core.Instances append data/ReutersGrain-test.arff \\
data/ReutersGrain-train.arff > data/ReutersGrain.arff
```

in the simple CLI application.

As expected, we observe a class imbalance between the two kinds of articles in Figure 11. Surely the number of articles not talking about grains, should be larger than those talking about it. Moreover as $1998 + 168 = 2158$, which is the total number of instances, this means we don't have any misssing data.

Next, to basically apply any classification algorithm, we will need to convert out texts to word vectors. Hence we will choose the `StringToWordVector` filter from `weka/filters/unsupervised/attribute` using the choose button in filter tab of Figure 11. And then we click `Apply`. As a result we now have binary attributes corresponding to many frequently occurring words. These attributes take value 1 if the word occurs in a particular text and 0 otherwise. Some example words are shown in Figure 12.

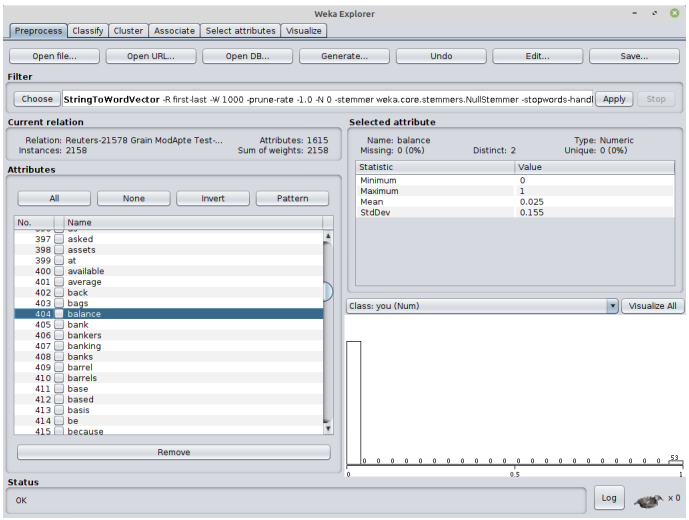


Figure 12: Some sample words chosen for representing text as bag of word vectors. The mean in the selected attribute section is indicative of what percentage of articles contain that word.

6.2 Classification & Visualisation

Now we are in a position to run the classifier. We move to the *Classify* tab and change the attribute to be predicted from the last one to the nominal(i.e. categorical) **class-att** which specifies the label of each article. This can be done by opening the drop down menu between the **Test Options** and **Results Lists** panels. We choose the **J48** classifier, which is like Weka's implementation of ID3, and run the classifier with 10 fold cross-calidation. Figure 13 shows one of the sample decision tree produced during a cross validation run. Figure 14 provides the metrics over 10-fold cross validation.

To better visualise the tree, we can right click on the `trees.j48` option in the results list and choose *Visualise Tree*. The result is presented in Figure 15.

7 Experiment 7: Weather Dataset

The weather dataset for classification is a small sample toy dataset consisting of four attributes, viz., **outlook**, **temperature**, **humidity**, **windy**, used to predict a label **play** which corresponds to whether we should go to play or not.

7.1 Preprocessing & Visualisation

The dataset is shown in Figure 16. As we can see, there are no missing values. All attributes are nominal. We visualise the distribution of values of various attributes in Figure 17. This can be generated by first selecting all attributes in

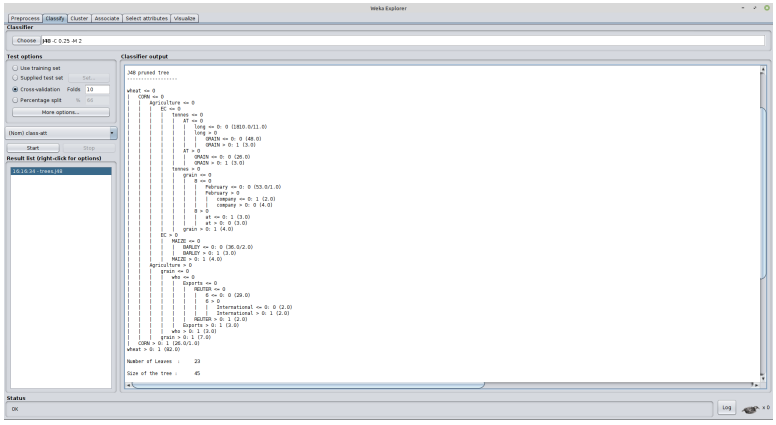


Figure 13: The final pruned tree produced during one of the cross-validation runs by the J48 algorithm.

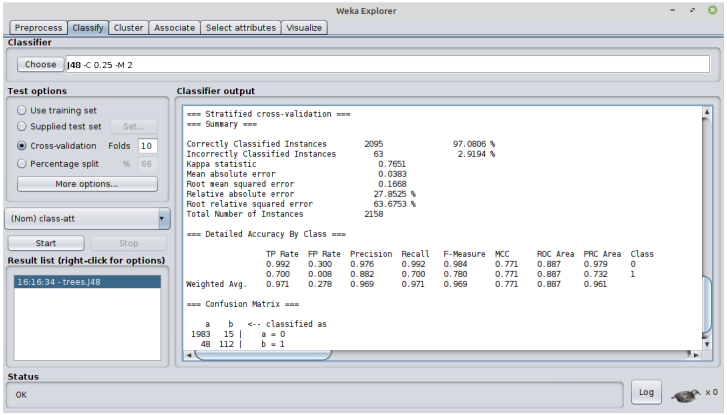


Figure 14: Final metrics for 10-fold Cross Validation on the ReutersGrain dataset using the J48 classifier.

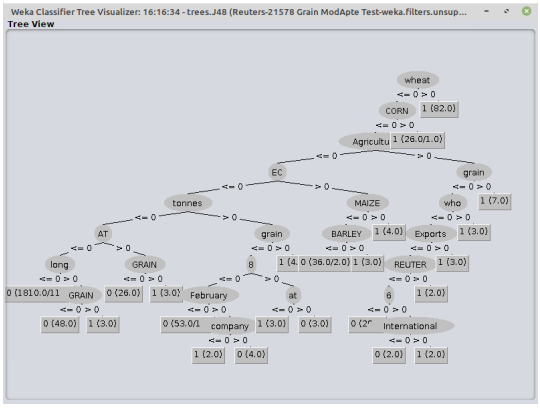
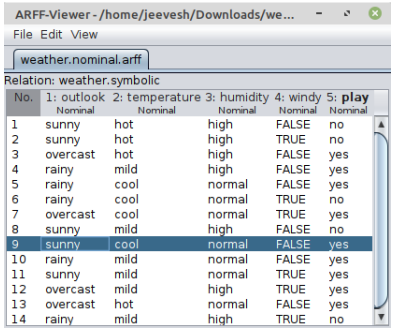


Figure 15: A proper visualisation of the output tree produced by J48 on the ReutersGrain dataset.



No.	1: outlook Nominal	2: temperature Nominal	3: humidity Nominal	4: windy Nominal	5: play Nominal
1	sunny	hot	high	FALSE	no
2	sunny	hot	high	TRUE	no
3	overcast	hot	high	FALSE	yes
4	rainy	mild	high	FALSE	yes
5	rainy	cool	normal	FALSE	yes
6	rainy	cool	normal	TRUE	no
7	overcast	cool	normal	TRUE	yes
8	sunny	mild	high	FALSE	no
9	sunny	cool	normal	FALSE	yes
10	rainy	mild	normal	FALSE	yes
11	sunny	mild	normal	TRUE	yes
12	overcast	mild	high	TRUE	yes
13	overcast	hot	normal	FALSE	yes
14	rainy	mild	high	TRUE	no

Figure 16: The weather dataset for classification of whether we should go to play based on weather conditions.

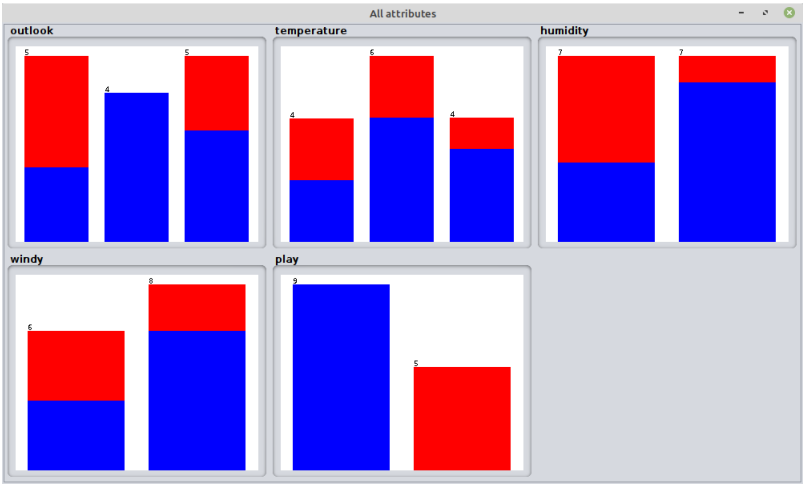


Figure 17: Distribution of values of various attributes. On X-axis are various nominal values of the attributes. The height of blue color is for number of samples with their labels as yes or "do play" out of the total number of samples for which the attribute takes a particular value. Outlook can be sunny, overcast or rainy. Temperature is hot, mild or cold. Rest of the attributes are binary.

the *Attributes panel* of the Explorer App, using the **All** button and then clicking on **Visualize All** on top of the graph at bottom right.

7.2 Classification & Visualisation

We again select the J48 classifier from `weka/classifiers/trees/` by clicking the **choose** button in the *Classify* tab. The results for 10 fold Cross-Validation are presented in Figure 18.

We observe that only 50% of samples are correctly classified. One can think of this as a bad classifier, but it is actually good as we can see that the classifier doesn't try to infer too much(i.e., make too many nodes in its tree) given the limited amount of samples it has. This ensures generalisation. We try other classifiers and find that this performs worse than ZeroR(majority class classifier), but better than Decision Table.

The visualized decision tree produced is presented in Figure 19.

8 Experiment 8: Clustering Weather Data

8.1 K-Means

We cluster the Weather Data presented in previous section by going to the *Cluster tab* and selecting the K-Means Clusterer. By default, it uses Euclidean distance as the distance measure. We get two clusters shown in Figure 20. We form clusters using the 4 predictive attributes and evaluate on the basis of the

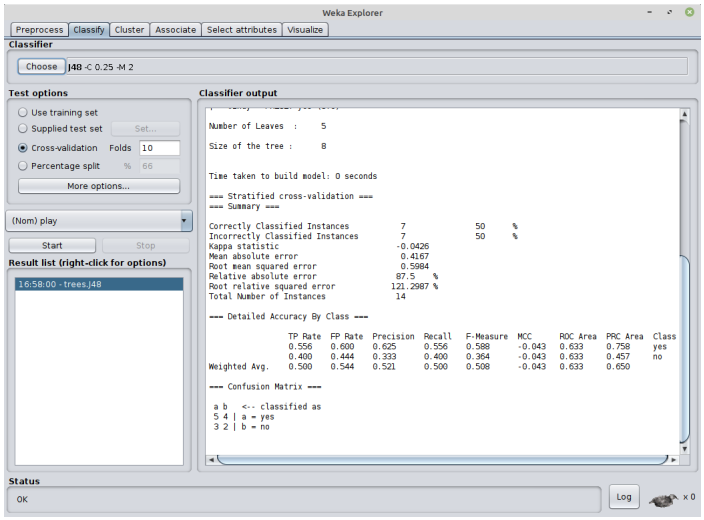


Figure 18: Metrics for 10 fold Cross-Validation using J48 classifier on the Toy Weather dataset.

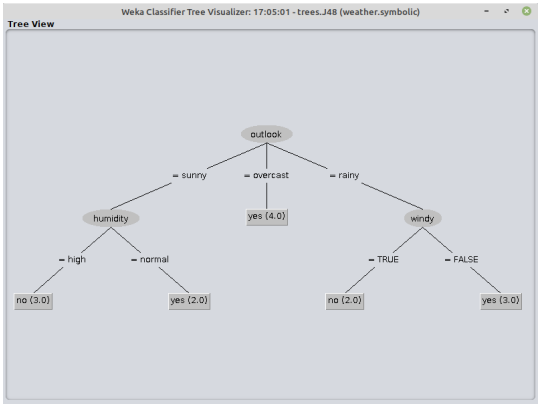


Figure 19: The final decision tree using the small number of examples in the Weather dataset. The decisions seem more or less in line with real life choices, hence correct.

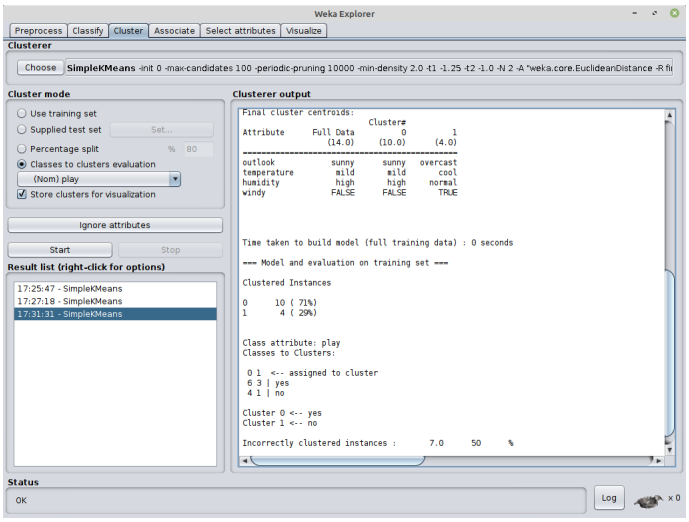


Figure 20: We obtain two clusters: one with centroid (sunny, mild temp, high humidity, not windy) and other with centroid: (overcast, cool temp, normal humidity, windy). Most attributes are opposite in the two clusters.

play labels. Again, we observe 50% of instances are labelled correctly. Though the clusters seem logical and in line with out intuitions.

8.2 Cobweb

We show the results for clustering using the cobweb algorithm in Figure 21. Firstly, we trained using the default cutoff value, and found that each element is considered a different cluster. We slowly increase this cut-off from its default value of 0.0028 to 0.28 and find that the number of clusters stays as the number of samples, but once we reach 0.29, all clusters collapse to a single cluster. This provides a higher accuracy for training set, namely, same as majority class classifier, ZeroR, but fails to gather usual information from the limited data.

9 Experiment 9: Association Technique

We use the Contact Lenses data, which is another toy dataset provided by Weka consisting of 24 instances, with 5 attributes each. It is available in `contact-lenses.arff` and is shown in Figure 22. There are four nominal predictors again, viz., **age**, **spectacle prescription**, **astigmatism**, **tear production rate** used to predict whether someone needs contact lenses and which kind of contact lenses, if needed.

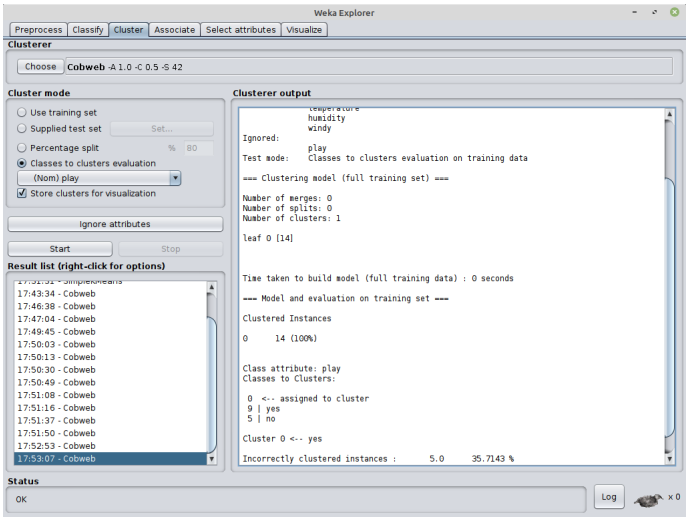


Figure 21: Final evaluation metrics for clustering using cobweb on the Toy Weather Dataset provided in Weka.

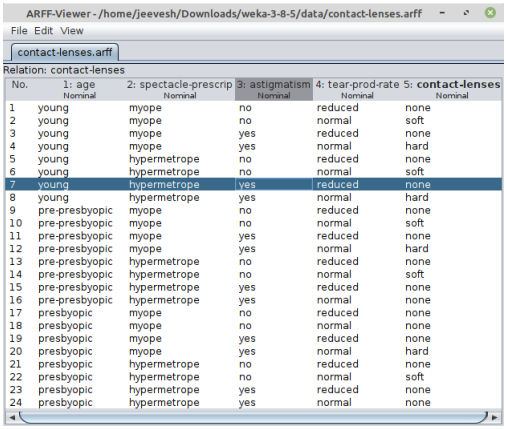


Figure 22: Contact lenses dataset provided by Weka visualised in **ArffViewer**.

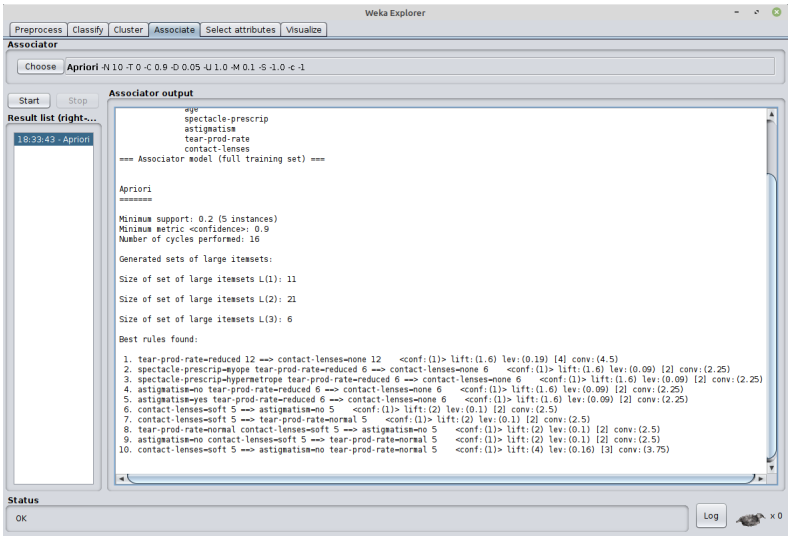


Figure 23: Association Rules inferred using the Apriori rule algorithm on the contact lenses dataset.

9.1 Association Rules

Figure 23 shows the association rules inferred from the contact lenses dataset using the Apriori algorithm. We find that the strongest rules inferred are indeed correct. For example, we shouldn't prescribe contact lenses to people with reduced tear production rate. We also observe a relation between soft contact lenses and normal tear production rate in rule 7. Additionally, there are lots of spurious association rules predicted too. This is mainly due to lack of data.

10 Experiment 10: Nearest Neighbor

We use the Iris dataset, available in `weka-3-8-5/data/iris.arff`(an excerpt is shown in Figure 24). This dataset consists of 4 predictors(sepal length and width, petal length and width) which are used to predict the type of the iris flower from the set {setosa, virginica, versicolor}. As the length and width are numeric attributes, we hope the nearest neighbor search with Euclidean distances to produce good results.

10.1 Running Nearest Neighbor

To run nearest neighbor in the *Classify* tab in the *Explorer* App, click the **choose** button and choose the classifier `lazy/IBk`. Next click on **start** button to start the nearest neighbor classification for each sample. We observe the results for this in Figure **??**.

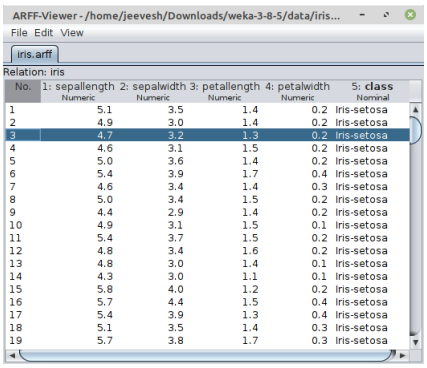


Figure 24: Some instances from Iris Dataset as visualised in **ArffViewer**

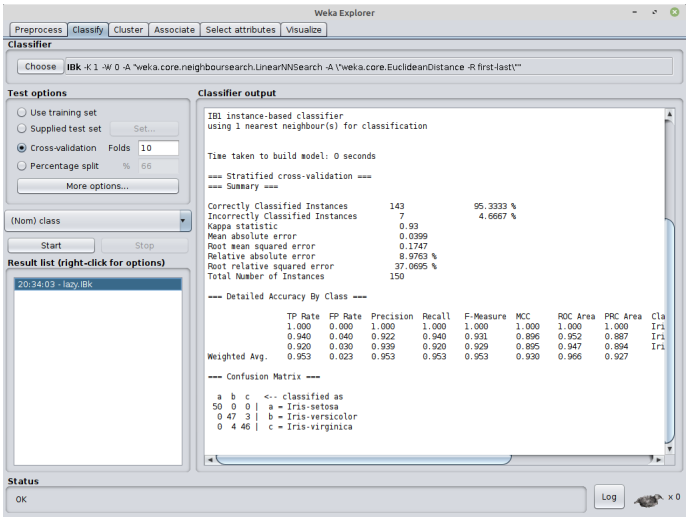


Figure 25: A whooping 95.33% of instances are classified correctly using nearest neighbor classification for the Iris dataset.

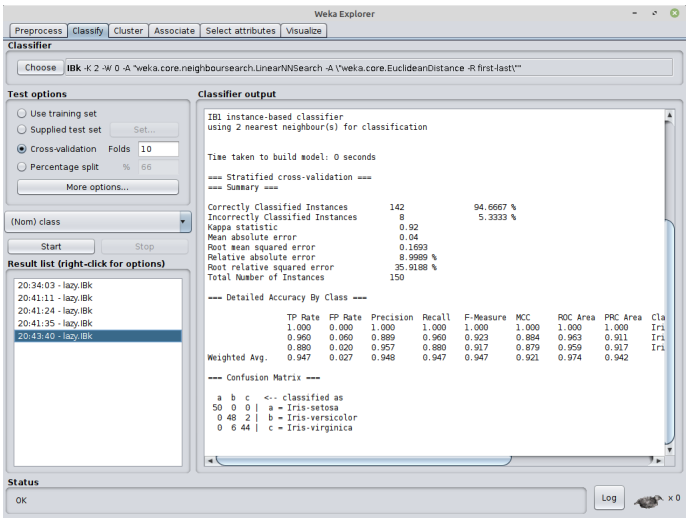


Figure 26: The output for $k = 2$ nearest neighbor search on the Iris dataset.

The results in Figure 25 are for $k = 1$. To change the value of k(the number of nearest neighbors considered), click on the bar containing the **IBk** next to the **Choose** button and modify k. We observe results for higher values of k too. The results for $k = 2$ are observed in Figure 26. We observe a slight drop in performance to 94.66%. But, for higher values of k, viz., $k \geq 3$, we observe that the performance rises back to 95.33%.