**Lab 3: Use Apriori algorithm to form association rules.**

|  |  |
| --- | --- |
| ***TID*** | **Items** |
| *1* | A, B, C, D, E, F |
| *2* | B, C, D, E, F, G |
| *3* | A, D, E, H |
| *4* | A, D, F, I, J |
| *5* | B, D, E, K |

**Introduction:**

The Apriori algorithm is a fundamental data mining technique used to find frequent itemsets and generate association rules in large transactional datasets. It is widely applied in market basket analysis to discover patterns such as which items are often bought together.

**How does Apriori works?**

The Apriori algorithm operates on a principle known as the Apriori property, which states that:

*"All non-empty subsets of a frequent itemset must also be frequent."*

The steps of Apriori algorithm can be outlined as below:

1. Initialize k = 1
2. Generate all frequent 1-itemsets by scanning the database and counting item frequencies
3. Repeat:

* Use the frequent itemsets of size k to generate candidate itemsets of size k+1
* Prune candidate itemsets that contain infrequent subsets (Apriori property)
* Scan the database to determine the support count of each candidate
* Retain only those candidates whose support ≥ minimum\_support to form the frequent itemsets Lk+1
* Increment k by 1

1. Continue until no more frequent itemsets are found
2. From the frequent itemsets, generate association rules that satisfy the minimum\_confidence threshold

**Apriori in WEKA:**

WEKA (Waikato Environment for Knowledge Analysis) provides a simple interface to apply the Apriori algorithm on datasets in .arff format. Once the data is loaded:

* 1. The Apriori algorithm can be applied from the "Associate" tab.
  2. The user can set parameters like minimum support, confidence, and number of rules.
  3. The output includes frequent itemsets and association rules in readable form

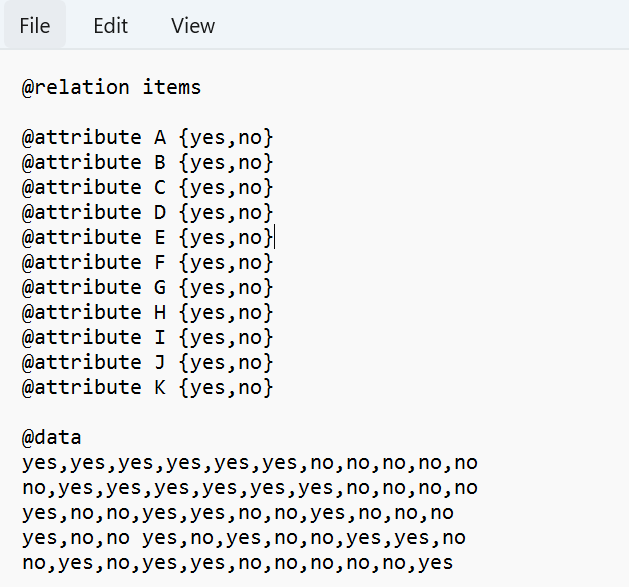


Figure 3.1: .arff file

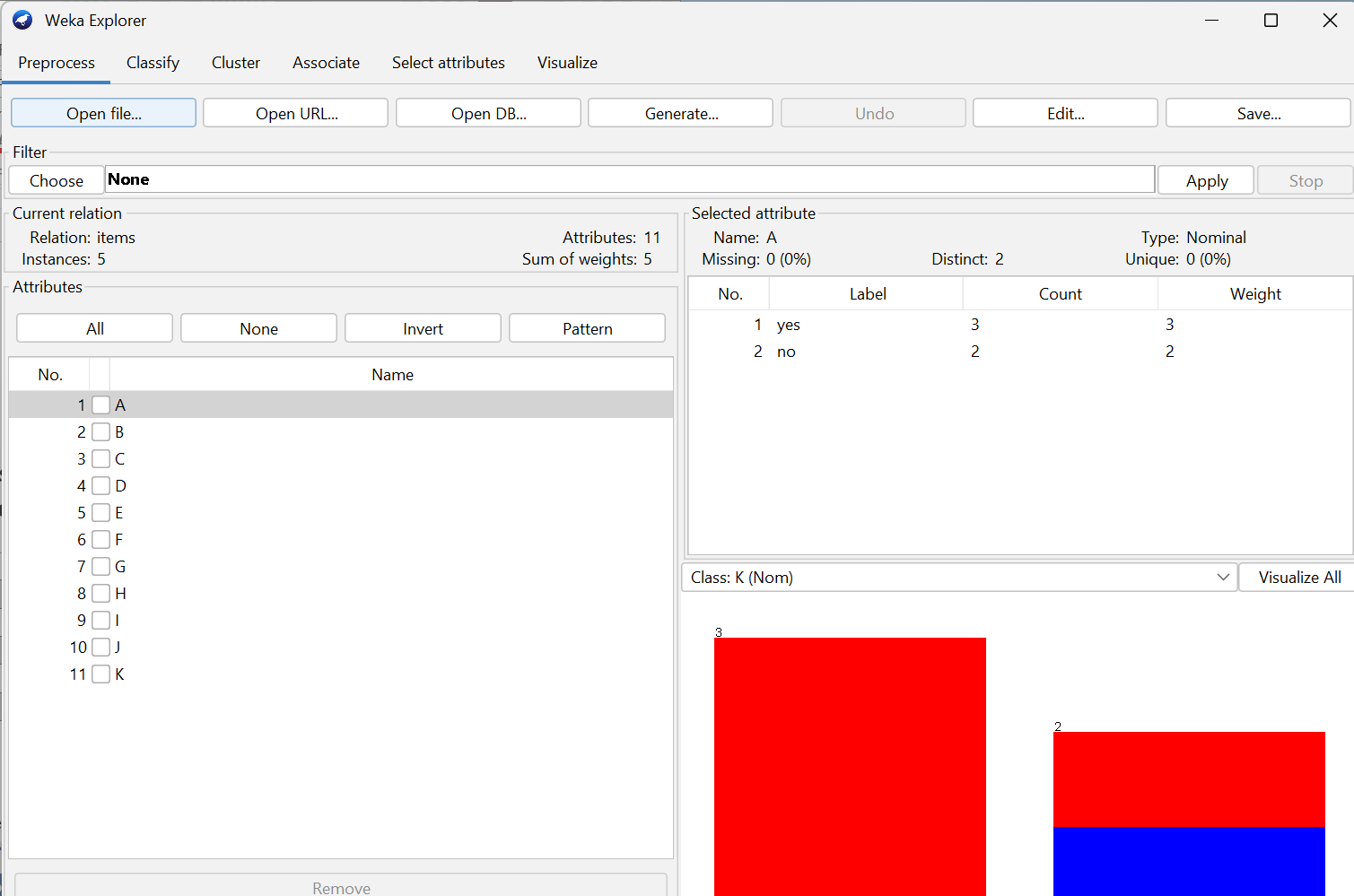


Figure 3.2: Pre-process data

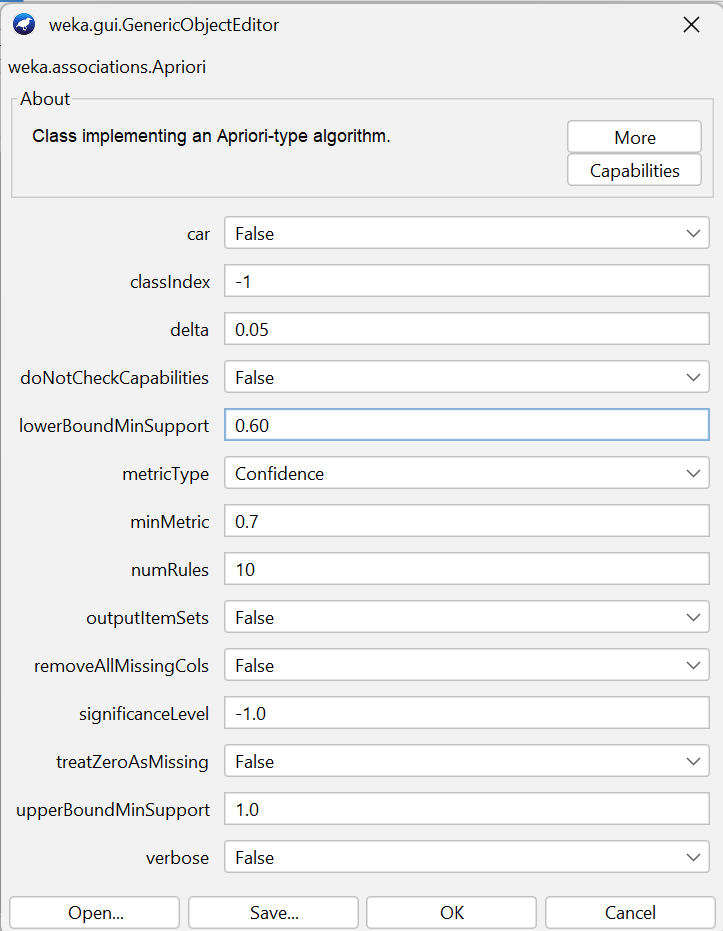


Figure 3.3: Configuration for association

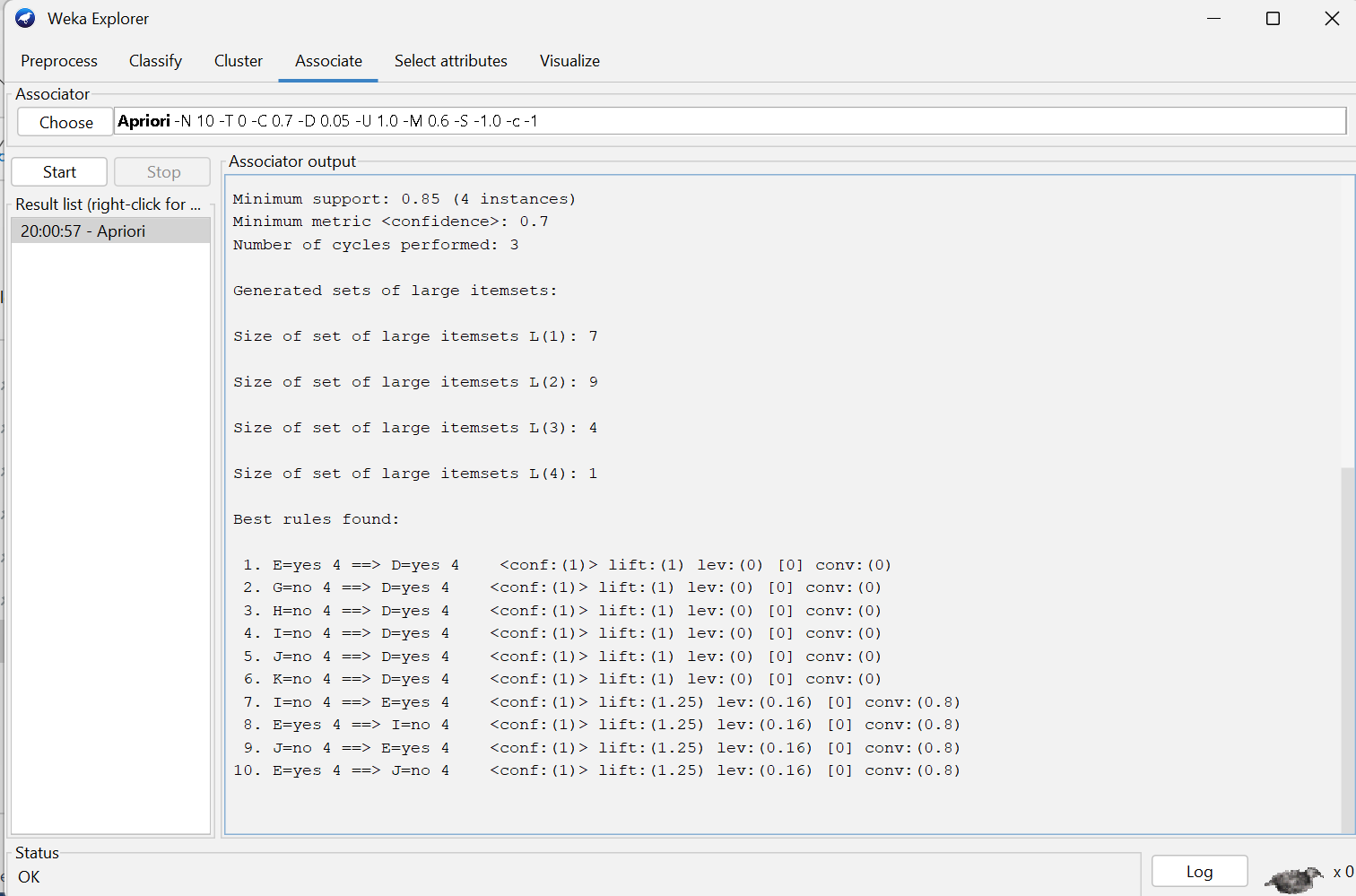


Figure 3.4: Output Result

**Conclusion:**

The Apriori algorithm efficiently discovers frequent itemsets and association rules by leveraging the property that all subsets of a frequent itemset must also be frequent

**Lab 4: Use FP growth algorithm to form association rules.**

**Introduction:**

The FP-Growth (Frequent Pattern Growth) algorithm is an efficient and scalable method for mining frequent itemsets without candidate generation. Unlike the Apriori algorithm, which generates and tests candidate sets repeatedly, FP-Growth uses a compressed representation of the database called the FP-tree (Frequent Pattern tree). This tree structure captures the frequency of itemsets while preserving the association among items.

**How FP-Growth Works?**

The steps of FP Growth algorithm can be outlined as below:

1. Scan the database to determine the frequency (support count) of each item.
2. Discard infrequent items (those with support < min\_sup) and sort the frequent items in descending order of frequency.
3. Construct the FP-tree:

* Initialize the tree with a null root.
* For each transaction:
  + Filter and sort items based on frequency order.
  + Insert the transaction into the FP-tree, updating counts and links as necessary.

1. Mine the FP-tree recursively:

* For each item in the tree (starting from the least frequent):
  + Construct the item’s conditional pattern base (subsets that lead to the item).
  + Build the conditional FP-tree from the pattern base.
  + Recursively mine the conditional tree to extract frequent patterns.

1. Aggregate all patterns discovered through recursion to obtain the final set of frequent itemsets.

**FP-Growth in WEKA:**

WEKA provides a built-in implementation of FP-Growth under the "Associate" tab. After loading the dataset (in .arff format), users can select FP-Growth and configure parameters such as:

* Minimum support threshold and Number of rules to generate

WEKA then generates the frequent itemsets and displays the corresponding association rules based on the mined data.

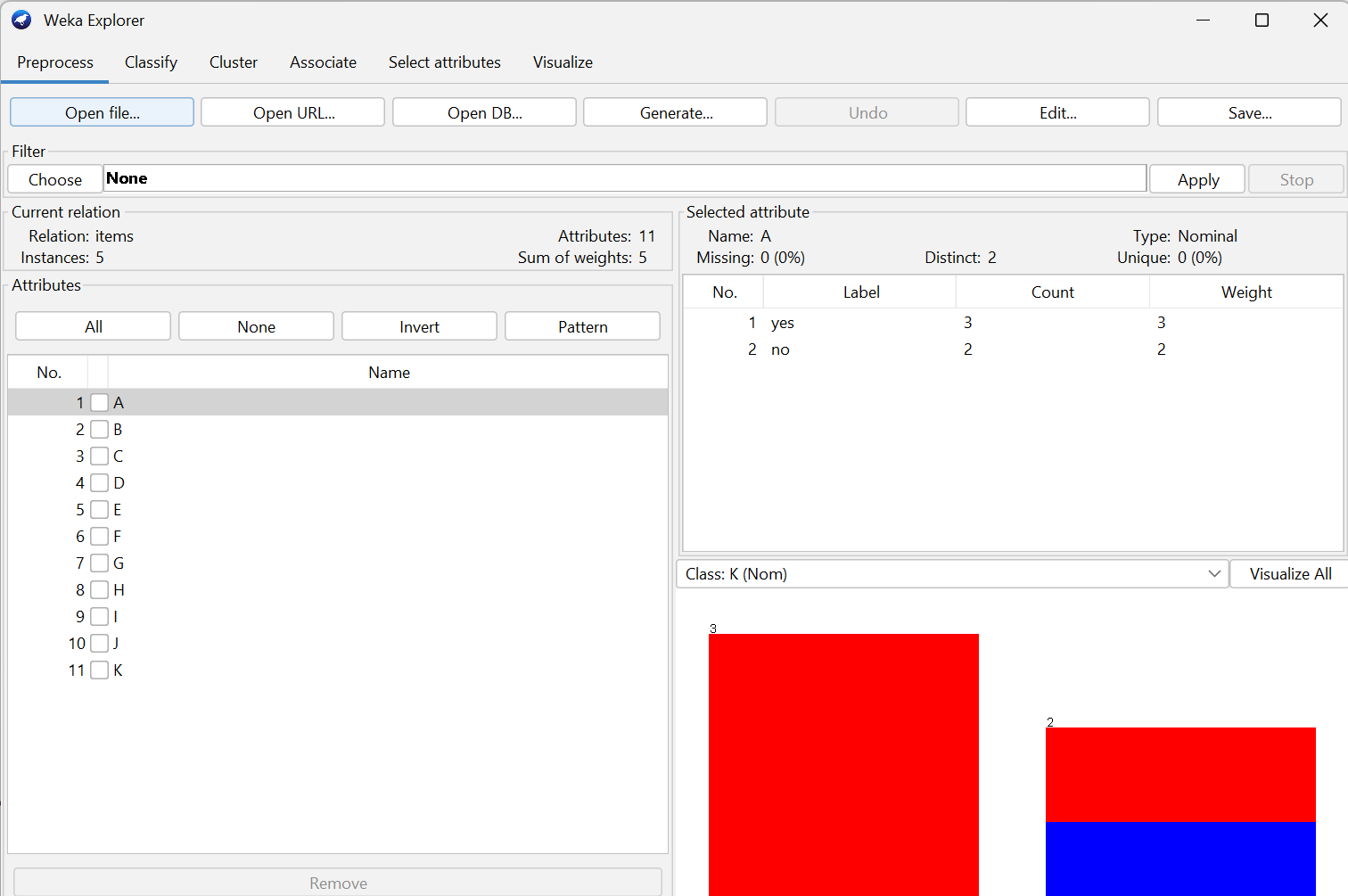


Figure 4.1: Pre-Process data

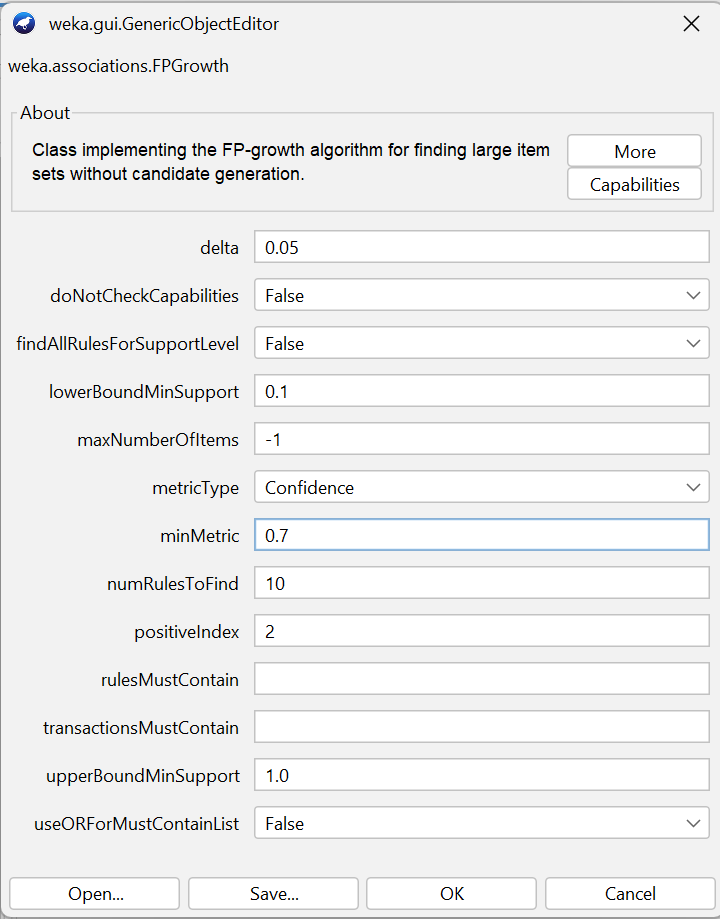


Figure 4.2: Configuration for association

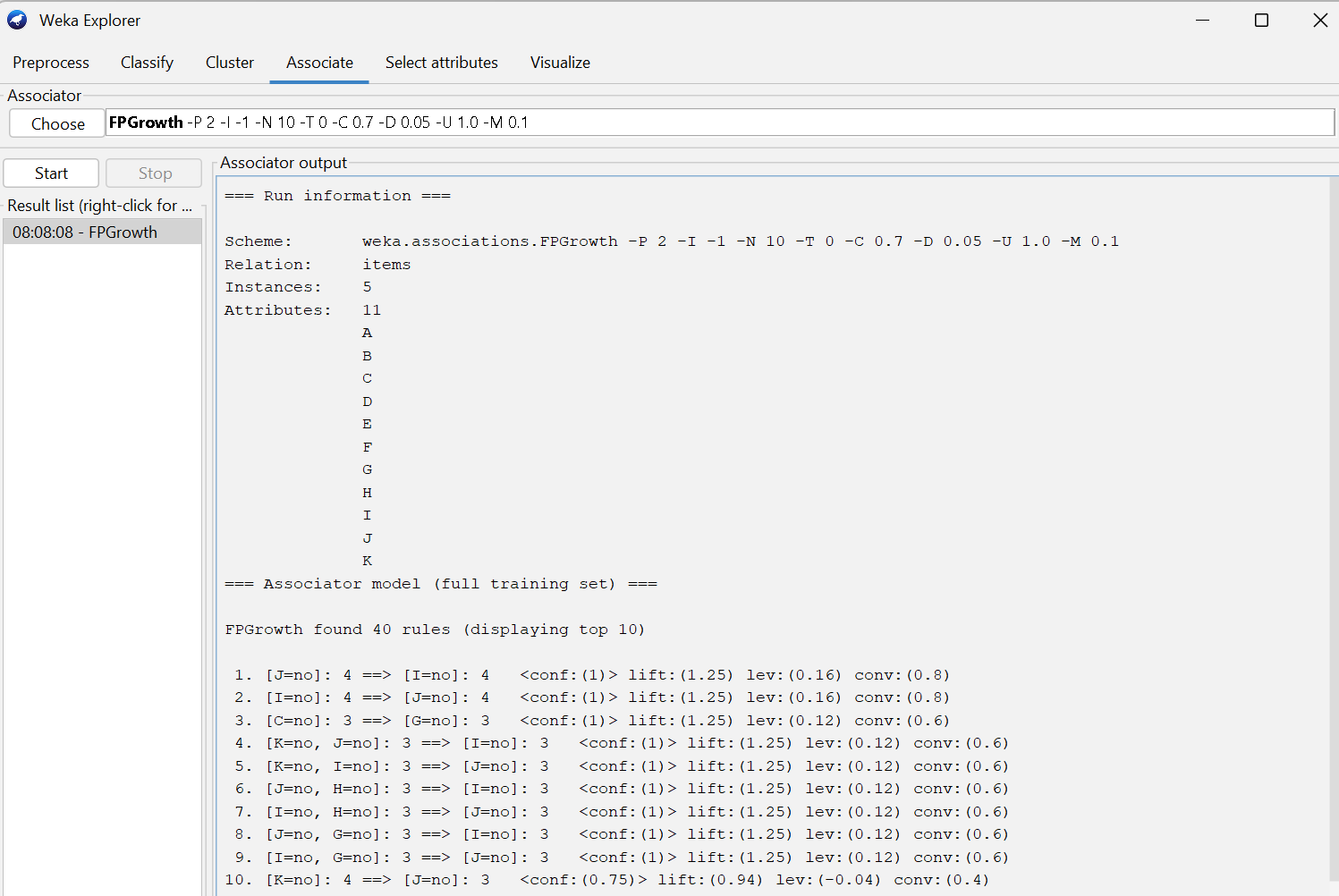
****

Figure 4.3: Output Result

**Conclusion:**

The FP-Growth algorithm mines frequent itemsets without candidate generation by using a compact FP-tree structure, making it faster and more scalable than Apriori.

**Lab 5: Use the J48 algorithm on the data and generate a decision tree.**

**Introduction**

J48 is an open-source implementation of the C4.5 decision tree algorithm used for classification tasks in machine learning. It is commonly available in the Weka data mining tool. J48 constructs a decision tree by recursively splitting the dataset based on the attribute that provides the highest information gain or gain ratio.

**How does J48 Algorithm works?**

The steps of J48 algorithm can be outlined as below:

1. **If all instances** belong to the same class, create a **leaf node** with that class.
2. **If attributes are empty**, create a leaf with the **majority class.**
3. For each attribute, calculate **information gain** (or gain ratio) using the formula:

Gain(S,A)=Entropy(S)−∑(∣Sv∣/∣S∣⋅Entropy(Sv))

1. **Select the attribute** with the highest gain as the **decision node.**
2. **Split the dataset** into subsets based on the selected attribute’s values.
3. **Recursively apply** the above steps to each subset to build subtrees.
4. **Prune the tree** after creation to remove branches that do not improve accuracy (optional but often done in J48).

# C4.5 (J48) in WEKA:

To implement C4.5 in WEKA:

1. Open WEKA and load a dataset (in .arff format).
2. Go to the "Classify" tab.
3. Choose trees > J48 as the classifier.
4. (Optional) Set parameters like:
   * Confidence factor for pruning (default is 0.25)
   * Minimum number of instances per leaf
   * Enable or disable unpruned tree
5. Click Start to train the model.
6. WEKA displays the decision tree, accuracy, and confusion matrix.

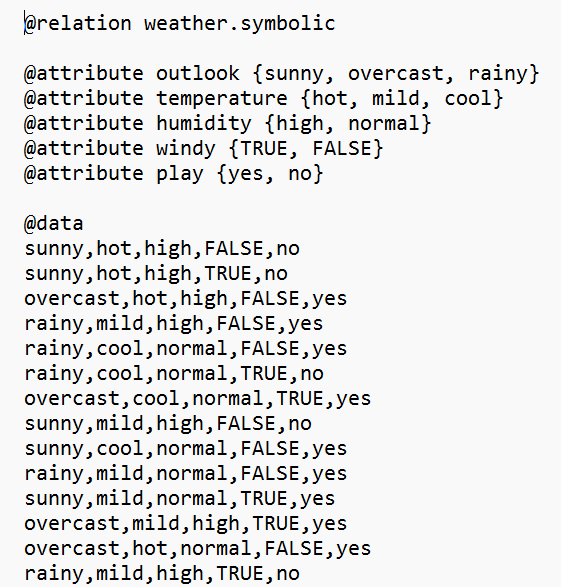


Figure 5.1: .arff file

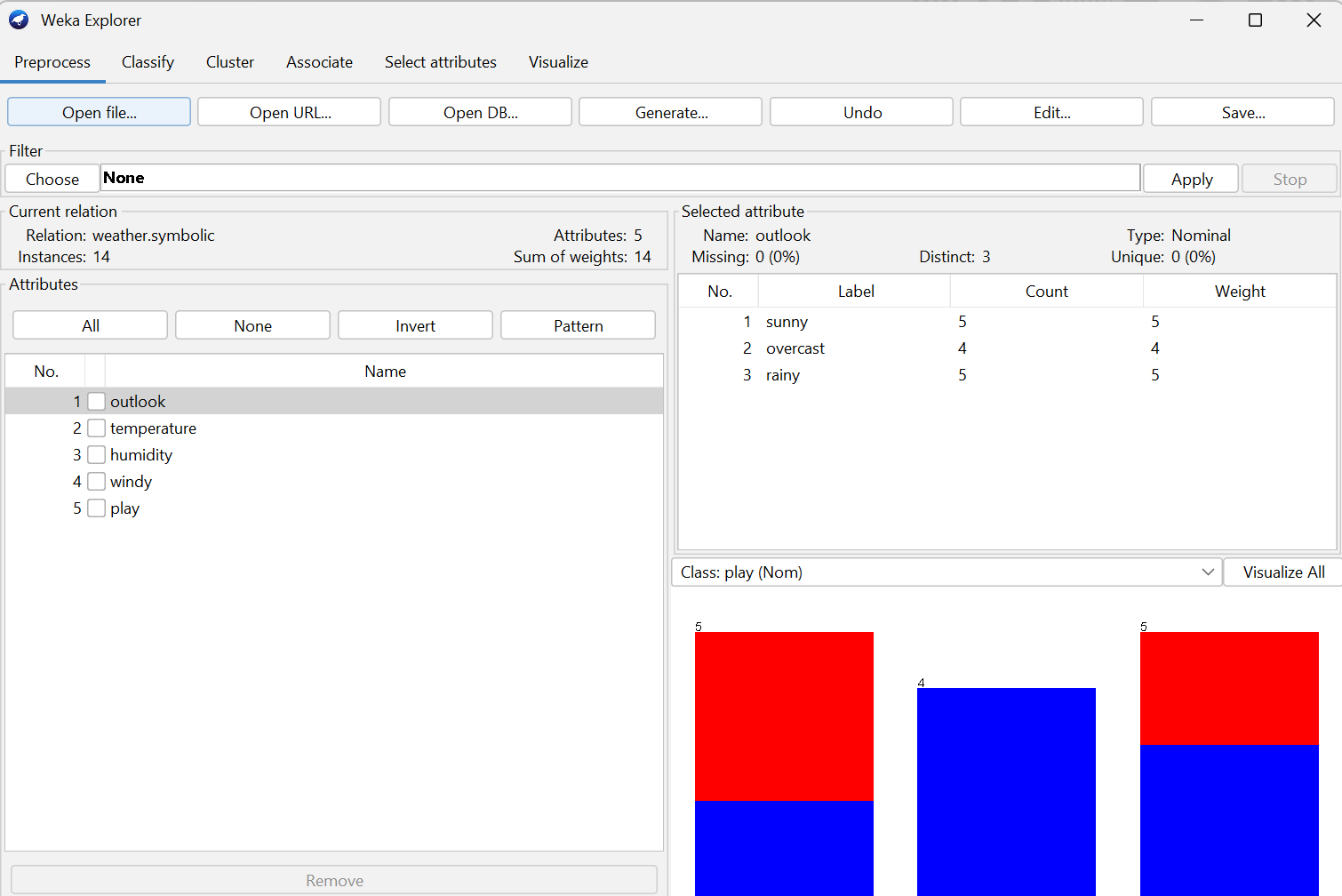


Figure 5.2: Pre-Process data

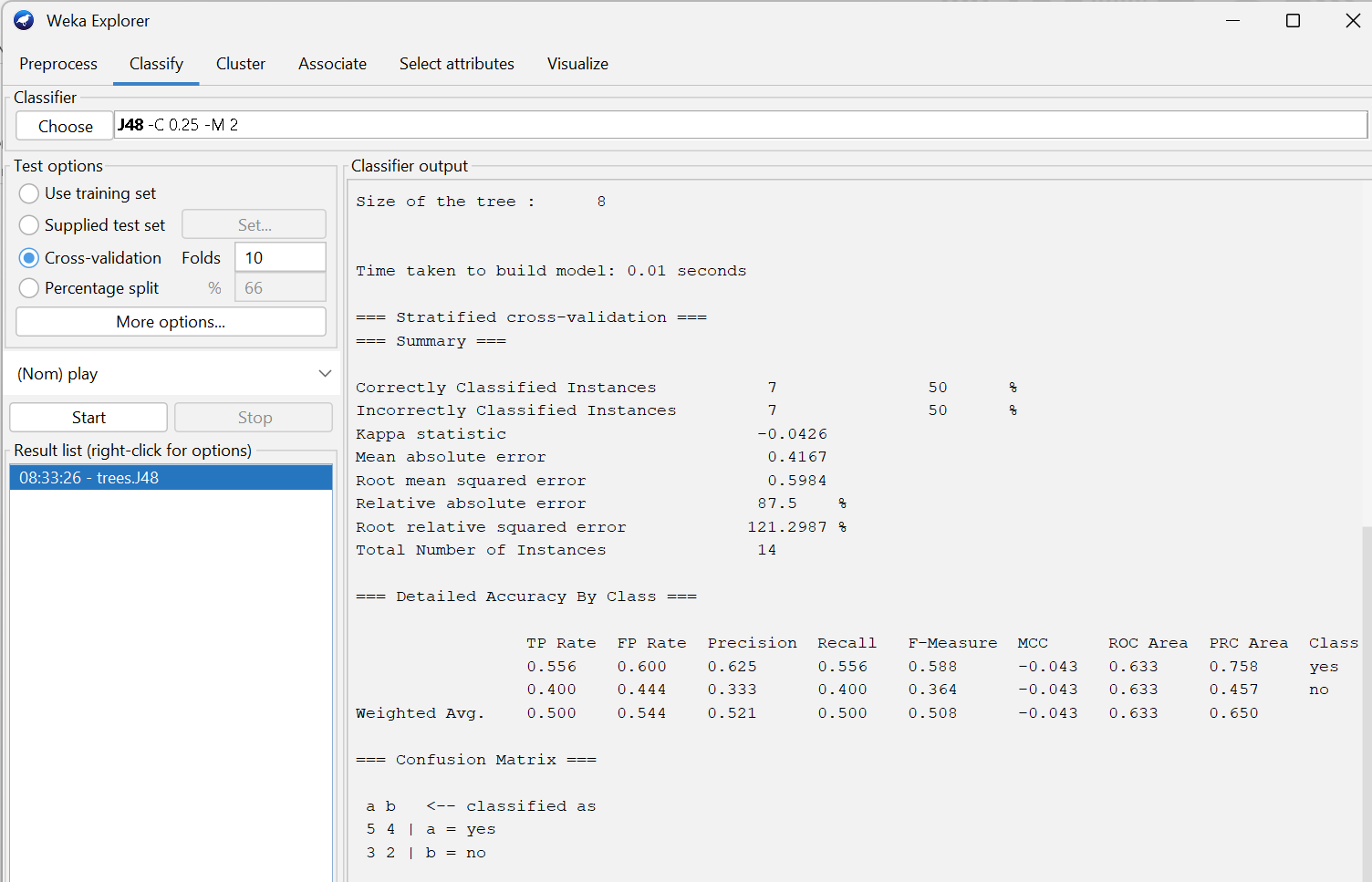
****

Figure 5.3: Classifier output

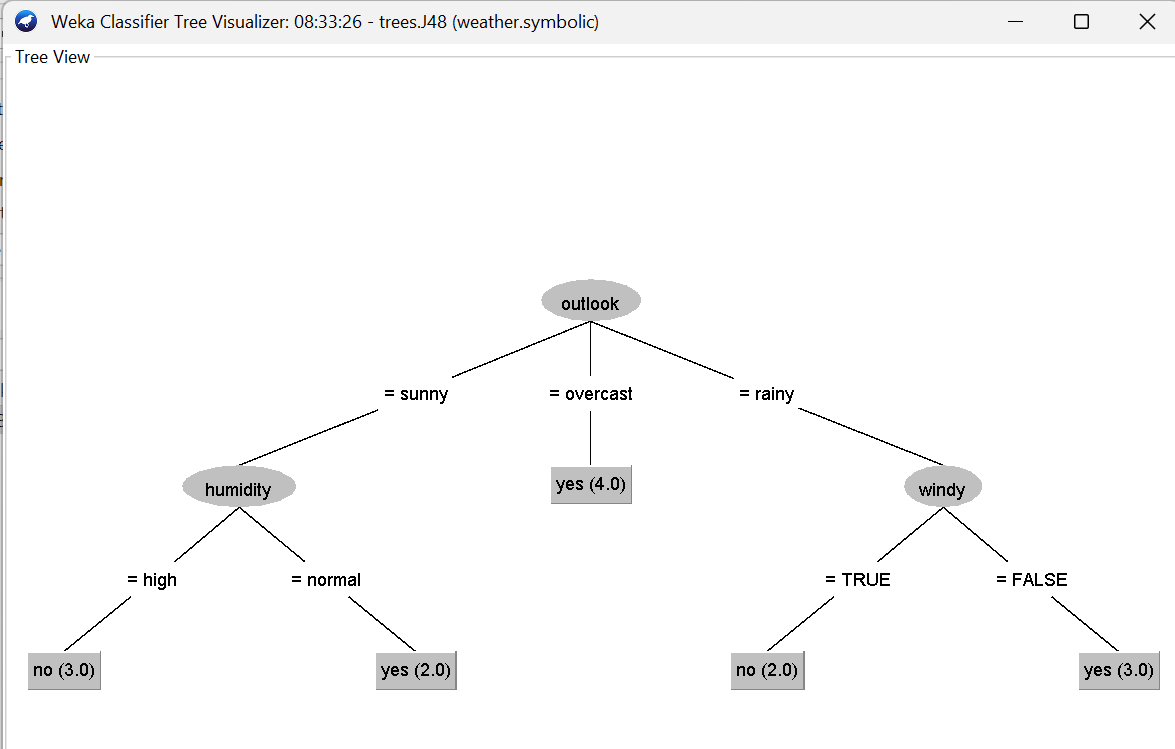
****

Figure 5.4: Decision tree

**Conclusion:**

J48 builds an efficient and interpretable decision tree by selecting attributes based on information gain, making it a reliable algorithm for classification tasks.

**Lab 6: Use the Naïve-Bayes algorithm on the data to classify them.**

**Introduction**

Naive Bayes is a probabilistic classification algorithm based on **Bayes' Theorem**, with the key assumption that features are conditionally independent given the class. Despite this "naive" assumption, the algorithm performs well in many real-world scenarios, especially in text classification, spam filtering, and sentiment analysis.

**How does Naïve-Bayes Algorithm works?**

The steps of Naïve-Bayes algorithm can be outlined as below:

1. For each class Ck, compute the prior probability:

P(Ck​)=Total number of instances/Number of instances in Ck​​

1. For each feature value xi in the input, compute the conditional probability given the class:

P(xi​∣Ck​)

1. Apply Bayes' Theorem:

* For each class Ck, compute the posterior probability:

P(Ck​∣X′)∝P(Ck​)×

1. Predict the class with the highest posterior probability:

=arg ​max​P(Ck​∣X′)

**Naïve Bayes in Weka:**

Weka provides a direct implementation of Naïve Bayes under the “Classify” tab.

Steps to implement:

1. Load a dataset (.arff format).
2. Go to the “Classify” tab.
3. Select bayes> NaiveBayes.
4. Click Start to train the model.

WEKA shows the confusion matrix, accuracy, and detailed class performance metrics like precision, recall, and F-measure.

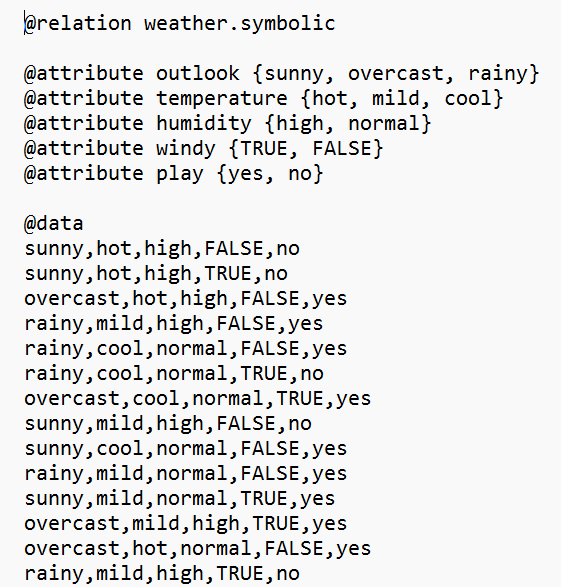


Figure 6.1: .arff file

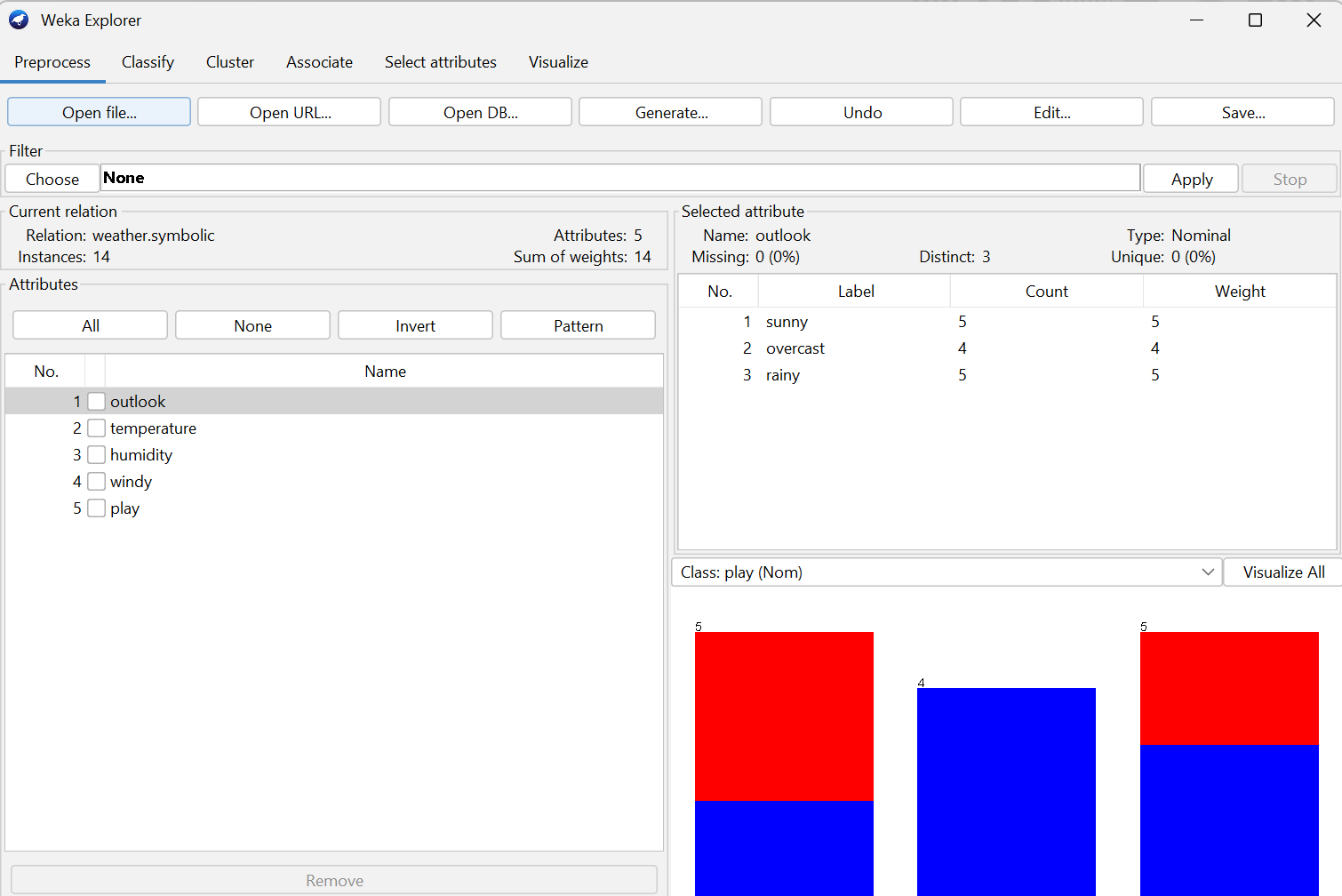


Figure 6.2: Pre-process data

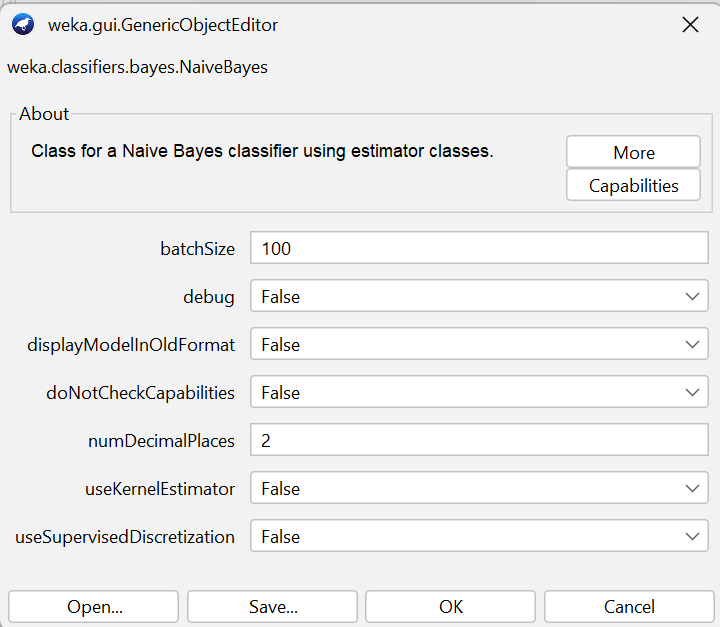
****

Figure 6.3: Configuration

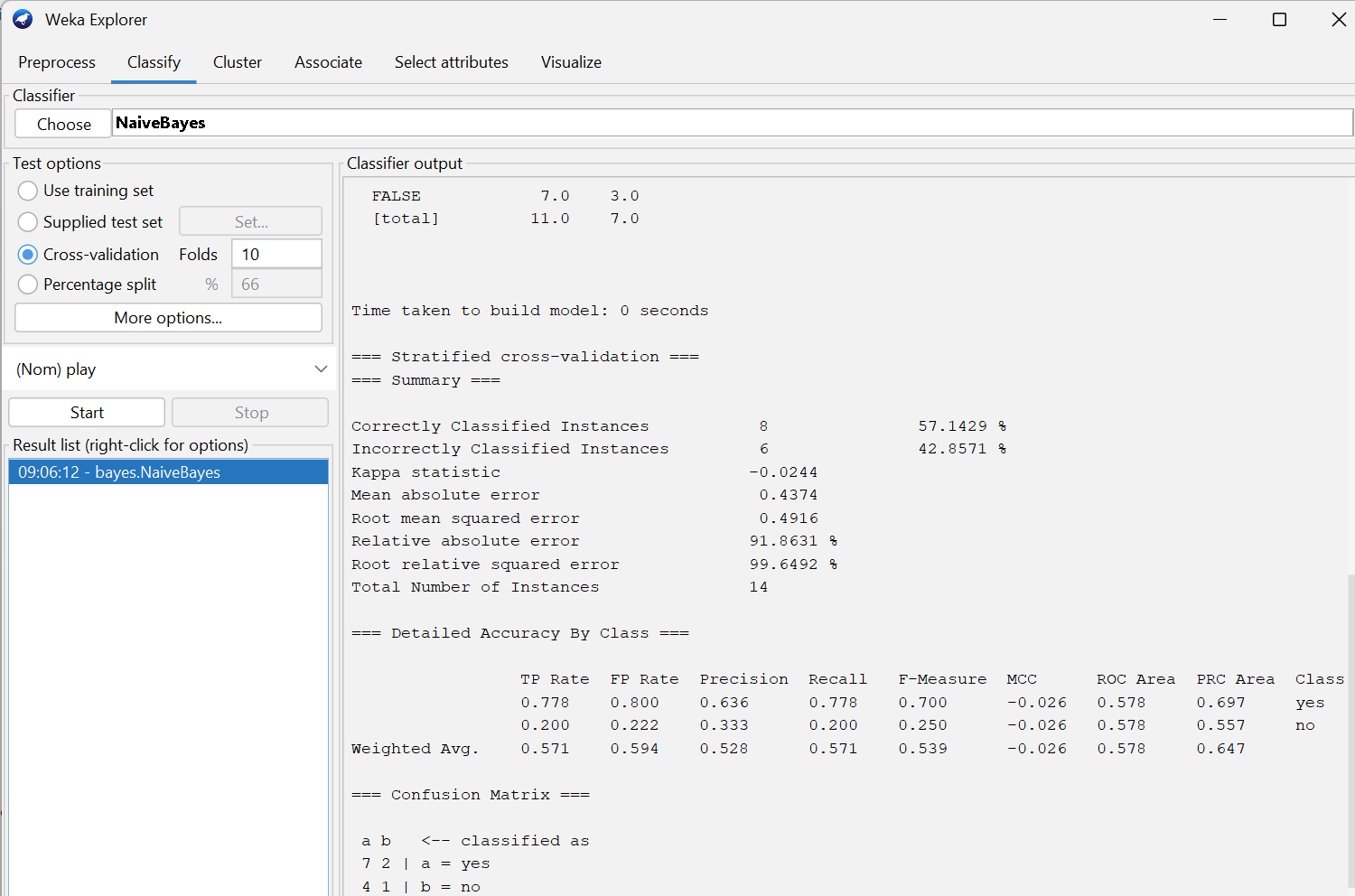
****

Figure 6.4: Output result

**Conclusion:**

Naive Bayes is a simple yet powerful probabilistic classifier that performs well in many domains by assuming conditional independence between features.