1:00 PM, Mon. 4/27

Artificial Intelligence	PROGRAM #5	due: [redacted]

This assignment is to be done individually by each student. You may not discuss any aspect of the analysis, design, coding, or testing with anyone but the instructor.

Problem: Machine Learning / Data Mining

Description: You will prepare data for use with Weka, experiment with several machine learning algorithms, and write a program that implements the classification rules obtained from machine learning.

Specific Requirements

1. Open the movies.csv file in a text editor (e.g., notepad) and note the format of this comma-separated-values (csv) file (attribute names separated by commas on the first line, attribute values separated by commas on subsequent lines). This is a common format that most spreadsheet and database programs can export. Use notepad and/or wordpad to manually create a .csv file from the agaricus-lepiota.data file. Use the corresponding .names file to create the attribute names on the first line of the .csv file (see items 7. and 9.). Note that the class of mushroom (poisonous or edible) is the first attribute.
2. Open your .csv file with Weka.
3. On the Classify tab, select the class (attribute that specifies the classification) using the drop-down list above the start button. Then use the J48 decision tree learning algorithm and copy and paste the test results into a separate report file created with Wordpad or Word. Then view the tree diagram by right-clicking the result name in the Result list. Resize the window and fit the tree to the screen until you get a good view and copy (use Alt-PrtScr) and paste it into your report.
4. Use the JRip rule learning algorithm and copy and paste the test results into your report.
5. Write a program that implements the classification rules that were learned by the JRip algorithm (they should be listed in your report after completing step 4.). You can use LISP, F# Java, C/C++, C#, VB, or CLIPS (but remember they should be checked in order and only the first applicable case should execute, so CLIPS might not be the best fit). The program should classify an example using these rules. Prompt the user for and read only the values of the attributes needed by the rules and then output the class the example belongs to. If you want to loop to try this more than once, allow the user to quit at any time.
6. Create a .arff input file from the car.data and car.names files. ARFF is Weka's preferred input format and is described in the attached pages. Note there's an inconsistency .. the dashes used in the description of several of the legal values for the attributes in the .names file are not used in the .data file, so use the values that appear in the .data file.
7. Open your .arff file.
8. Select the class, and run the same two learning algorithms used in steps 3 & 4 and add the test results to your report (don't include the tree diagram this time – it has too many branches).

ARFF format

We now look at a standard way of representing datasets that consist of independent, unordered instances and do not involve relationships among instances, called an *ARFF file.*

Figure 2.2 shows an ARFF file for the weather data in Table 1.3, the version with some numeric features. Lines beginning with a `%` sign are comments. Following the comments at the beginning of the file are the name of the relation (`weather`) and a block defining the attributes (`outlook`, `temperature`, `humidity`, `windy`, `play?`). Nominal attributes are followed by the set of values they can take on, enclosed in curly braces. Values can include spaces; if so, they must be placed within quotation marks. Numeric values are followed by the keyword `numeric`.

```
% ARFF file for the weather data with some numeric features
%
@relation weather

@attribute outlook { sunny, overcast, rainy }
@attribute temperature numeric
@attribute humidity numeric
@attribute windy { true, false }
@attribute play? { yes, no }

@data
%
% 14 instances
%
sunny, 85, 85, false, no
sunny, 80, 90, true, no
overcast, 83, 86, false, yes
rainy, 70, 96, false, yes
rainy, 68, 80, false, yes
rainy, 65, 70, true, no
overcast, 64, 65, true, yes
sunny, 72, 95, false, no
sunny, 69, 70, false, yes
rainy, 75, 80, false, yes
sunny, 75, 70, true, yes
overcast, 72, 90, true, yes
overcast, 81, 75, false, yes
rainy, 71, 91, true, no
```

Figure 2.2 ARFF file for the weather data.

Although the weather problem is to predict the class value `play?` from the values of the other attributes, the class attribute is not distinguished in any way in the data file. The ARFF format merely gives a dataset; it does not specify which of the attributes is the one that is supposed to be predicted. This means that the same file can be used for investigating how well each attribute can be predicted from the others, or to find association rules, or for clustering.

Following the attribute definitions is an `@data` line that signals the start of the instances in the dataset. Instances are written one per line, with values for each attribute in turn, separated by commas. If a value is missing it is represented by a single question mark (there are no missing values in this dataset). The attribute specifications in ARFF files allow the dataset to be checked to ensure that it contains legal values for all attributes, and programs that read ARFF files do this checking automatically.

In addition to nominal and numeric attributes, exemplified by the weather data, the ARFF format has two further attribute types: string attributes and date attributes. String attributes have values that are textual. Suppose you have a string attribute that you want to call *description*. In the block defining the attributes, it is specified as follows:

```
@attribute description string
```

Then, in the instance data, include any character string in quotation marks (to include quotation marks in your string, use the standard convention of preceding each one by a backslash, \). Strings are stored internally in a string table and represented by their address in that table. Thus two strings that contain the same characters will have the same value.

String attributes can have values that are very long—even a whole document. To be able to use string attributes for text mining, it is necessary to be able to manipulate them. For example, a string attribute might be converted into many numeric attributes, one for each word in the string, whose value is the number of times that word appears. These transformations are described in Section 7.3.

Date attributes are strings with a special format and are introduced like this:

```
@attribute today date
```

(for an attribute called *today*). Weka, the machine learning software discussed in Part II of this book, uses the ISO-8601 combined date and time format *yyyy-MM-dd-THH:mm:ss* with four digits for the year, two each for the month and day, then the letter *T* followed by the time with two digits for each of hours, minutes, and seconds.[1] In the data section of the file, dates are specified as the corresponding string representation of the date and time, for example, `2004-04-03T12:00:00`. Although they are specified as strings, dates are converted to numeric form when the input file is read. Dates can also be converted internally to different formats, so you can have absolute timestamps in the data file and use transformations to forms such as time of day or day of the week to detect periodic behavior.