ITI 1121. Introduction to Computing II
Winter 2021

Assignment 1
(Last modified on January 13, 2021)

Deadline: February 5, 2021, 11:30 pm

Learning objectives

- Edit, compile and run Java programs
- Utilize arrays to store information
- Apply basic object-oriented programming concepts
- Understand the university policies for academic integrity

Introduction

This year, we are going to implement, through a succession of assignments, a simplified version of a useful machine learning technique, called *decision tree classification*. If you are not familiar with decision trees and are curious to know what they are, you may wish to have a quick look at the following Wikipedia page: https://en.wikipedia.org/wiki/Decision_tree_learning.

For Assignment 1, however, you are *not* going to do anything that is specific to decision trees; you can complete Assignment 1 without any knowledge of decision trees! We will get to decision trees only in Assignments 2 and 3. If you find the above Wikipedia page overwhelming, fear not! As we go along, we will provide you with simple and accessible material to read on decision tree classification. Ultimately, the version of decision tree classification that you implement, despite still being extremely useful, has many of the complexities of the more advanced implementations removed (for example, handling "unknown" values in your training data).

As far as the current assignment – Assignment 1 – is concerned, we have modest goals: we would like to read an input file, which will (in future assignments) constitute the training data for our learning algorithm, and perform some basic tasks that are prerequisites to virtually any type of machine learning.

Specifically, you will be implementing the following tasks in Assignment 1:

- **Task 1.** Parsing comma-separated values (CSV) from a given data file and populating appropriate data structures in memory
- **Task 2.** Extracting certain summary data (*metadata*) about the characteristics of the input data; this metadata will come handy for the construction of decision trees in future assignments.

These two tasks are best illustrated with a simple example. Suppose we have a CSV file named `weather.csv` with the content shown in Figure 1.[1] The data is simply a table. The first (non-empty) row in the file provides the names of the table columns in a comma-separated format. Each column represents an *attribute* (also called a feature). The remaining (non-empty) rows are the *datapoints*.

In our example, each datapoint is a historical observation about weather conditions (in terms of outlook, temperature in fahrenheit, humidity and wind), and whether it has been possible to "play" a certain tournament (for example, cricket) outside. What a machine learning algorithm can do here is to "learn from examples" and help decide / predict whether one can play a tournament on a given day according to the weather conditions on that day.

Now, going backing to Task 1 and Task 2, below is what each of these tasks would do with the data in Figure 1.

[1] This example is borrowed from "Data Mining: Practical Machine Learning Tools and Techniques" 3rd Ed. (2011) by Ian H. Witten, Eibe Frank and Mark A. Hall.

```
outlook,temperature,humidity,windy,play
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 1: Example CSV file (`weather.csv`)

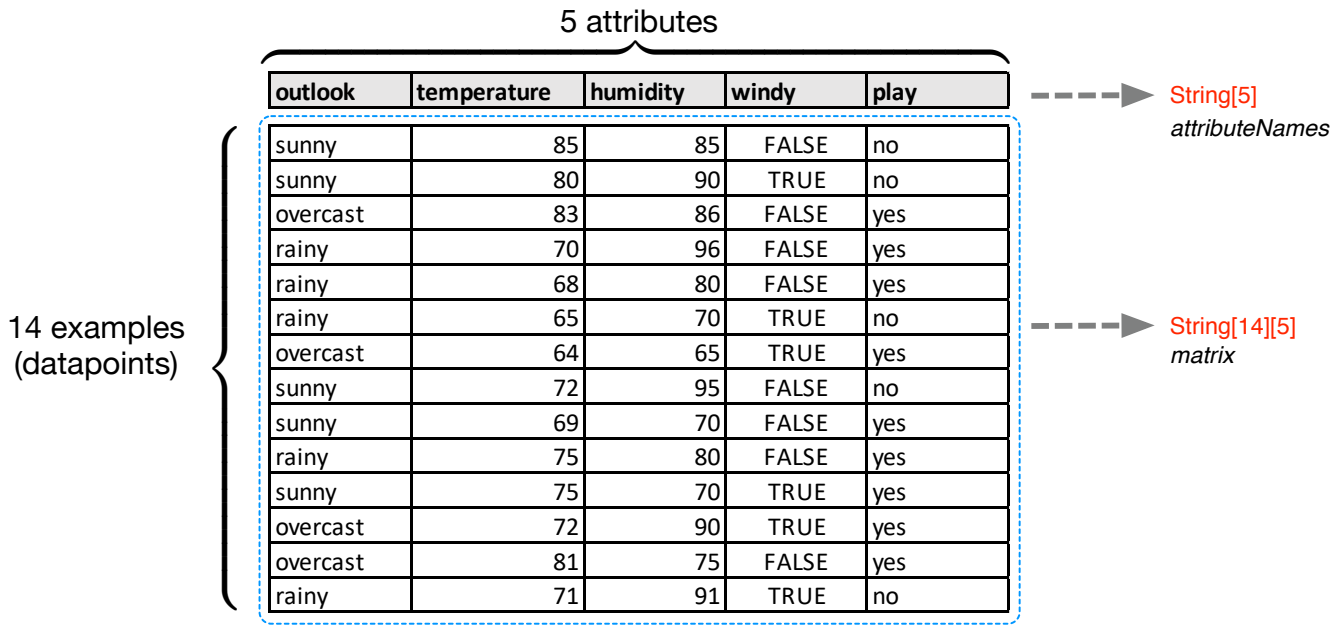

Figure 2: Results of parsing our example input file

- **Task 1** will parse the input data and build the conceptual memory representation shown in Figure 2. More precisely, we get (1) an instance variable, `attributeNames` (discussed later), instantiated with a String array of length 5 and containing the column names, and (2) an instance variable, `matrix` (also discussed later), instantiated with a two-dimensional String array (of size 14×5) and populated with the datapoints in the file.
- **Task 2** will identify the *unique* values observed in each column. If all the values in a column happen to be numeric, then the column is found to be of numeric type. Otherwise, the column will be of nominal type, meaning that the values in the column are to be treated as labels without any quantitative value associated with them. For our example file, the column types and the set of unique values for each column would be as shown in Figure 3. Note that, for this assignment, you do not need to sort the numeric value sets in either ascending or descending order. This becomes necessary only in the future assignments.

```
1) outlook (nominal): ['sunny', 'overcast', 'rainy']
2) temperature (numeric): [85, 80, 83, 70, 68, 65, 64, 72, 69, 75, 81, 71]
3) humidity (numeric): [85, 90, 86, 96, 80, 70, 65, 95, 75, 91]
4) windy (nominal): ['FALSE', 'TRUE']
5) play (nominal): ['no', 'yes']
```

Figure 3: Derived column types and unique values sets for our example input file

```
   outlook,temperature,humidity,windy,play

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 4: Example with surrounding spaces and empty lines that need to be ignored

Important Considerations (Read Carefully!)

While the assignment is conceptually simple, there are some important consideration that you need to carefully pay attention to in your implementation.

Determining the size of the arrays to instantiate: You will be storing the attribute names and datapoints using two instance variables that are respectively declared as follows:

```
private String[] attributeNames;
private String[][] matrix;
```

One problem that you have to deal with is how to instantiate these variables. To do so, you need to know the number of attributes (columns) and the number of datapoints. You can know the former number only after counting the attributes names on the first (non-empty) line of the file. As for the latter (number of datapoints), you can only know this once you have traversed the entire file. Later on in the course, we will see "expandible" data structure like linked lists, which do not have a fixed size, allowing elements to be added to them as you go along. For this assignment, you are **expressly forbidden** from using lists or similar data structures with non-fixed sizes. Instead, you are expected to work with fixed-size arrays. For this assignment, the easiest way to instantiate the arrays is through a *two-pass* strategy. This means that you will go over the input file twice. In the first pass, you merely count the number of columns and datapoints. With these numbers known, you can instantiate `attributeNames` and `matrix`. Then, in a second pass, you can populate (the now-instantiated) `attributeNames` and `matrix`. Note that, as illustrated in Figure 2, **you are expected to instantiate `matrix` as a *row* × *column* array**, as oppposed to a *column* × *row* array. While this latter strategy is correct too, you are asked to use the former (that is, *row* × *column*) as a convention throughout this assignment.

Removing blank spaces and empty lines: The blank spaces surrounding attribute names and values should be discarded. For example, consider the input file in Figure 4. This file is the same as the one in Figure 1, only

```
outlook,'temperature, in fahrenheit',humidity,windy,play

sunny,85,85,FALSE,no
sunny,80,90,TRUE,no
overcast,83,86,FALSE,yes
'rainy, mild',70,96,FALSE,yes
'rainy, mild',68,80,FALSE,yes
'rainy, heavy',65,70,TRUE,no
overcast,64,65,TRUE,yes
sunny,72,95,FALSE,no
sunny,69,70,FALSE,yes
'rainy, mild',75,80,FALSE,yes
sunny,75,70,TRUE,yes
overcast,72,90,TRUE,yes
overcast,81,75,FALSE,yes
'rainy, heavy',71,91,TRUE,no
```

Figure 5: Example usage of commas in attribute names and values

```
1) outlook (nominal): ['sunny', 'overcast', 'rainy, mild', 'rainy, heavy']
2) temperature, in fahrenheit (numeric): [85, 80, 83, 70, 68, 65, 64, 72, 69, 75, 81, 71]
3) humidity (numeric): [85, 90, 86, 96, 80, 70, 65, 95, 75, 91]
4) windy (nominal): ['FALSE', 'TRUE']
5) play (nominal): ['no', 'yes']
```

Figure 6: Metadata derived from the input shown in Figure 5

with some surrounding spaces added. These surrounding spaces need to be *trimmed* and ignored. The same goes with empty lines. Empty lines can be treated as non-existent.

Supporting commas in attribute names and values: Since commas are used as separators (delimiters), it is important to provide a way to support commas *within* attribute names and attribute values. To do so, you will need to implement an escape sequence mechanism. You will do so using single quotes ('). More precisely, commas are to be treated as regular characters if a text segment is embraced with single quotes. To illustrate, consider the example input in Figure 5. The metadata information derived from this input is shown in Figure 6. While not explicitly shown by Figure 6, the values to store in `attributeNames` and `matrix` are obviously affected when escape sequences with single quotes are present in the input file.

Missing attribute values: There may be situations where not all attribute values are known (for example, due to incomplete data collection). In such cases, the attribute values in question may be left empty. Your implementation needs to be able empty (missing) attribute values. You can choose to represent missing values with a special value, for example `` `MISSING' ``. Alternatively, you can choose to represent missing values with an empty string (``''``). To illustrate, consider the input file in Figure 7, where some values are missing. The metadata derived from this input file is shown in Figure 8. Here, we have chosen to represent missing values with the empty string. For this assignment, you should designate any column that has missing values as *nominal*. For example, in Figure 7, some of the values for the `humidity` attribute are missing. This has resulted in `humidity` to no longer be a numeric attribute but rather a nominal one, as shown in Figure 8.

Efficiency in identifying unique attribute values: You need to be prepared for the possibility that your input file would be large. One particular place you need to be careful with is when you are determining the unique set of values that a given attribute can assume. Your implementation should be efficient (hint: should not do futile search) in order to avoid a quadratic runtime.

```
outlook,temperature,humidity,windy,play
sunny,85,85,FALSE,
sunny,80,90,TRUE,no
overcast,83,,FALSE,
rainy,70,96,FALSE,yes
rainy,68,80,FALSE,yes
rainy,65,70,TRUE,no
overcast,64,65,TRUE,yes
sunny,72,,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,  ,TRUE,
```

Figure 7: Example input file with missing values

```
1) outlook (nominal): ['sunny', 'overcast', 'rainy']
2) temperature (numeric): [85, 80, 83, 70, 68, 65, 64, 72, 69, 75, 81, 71]
3) humidity (nominal): ['85', '90', '', '96', '80', '70', '65', '75']
4) windy (nominal): ['FALSE', 'TRUE']
5) play (nominal): ['', 'no', 'yes']
```

Figure 8: Metadata derived from the input file shown in Figure 7

Implementation

We are now ready to program our solution. We will only need two classes for this. For the assignment, you need to follow the patterns that we provide. You **cannot change any of the signatures of the methods (that is you cannot modify the methods at all). You cannot add new *public* methods or variables.** You can, however, add new *private* methods to improve the readability or the organization of your code.

DataSet

There are a number of methods in `DataSet` that you need to either complete or implement from scratch. Guidance is provided in the template code in the form of comments. The locations where you need to write code have been clearly indicated with an inline comment that reads as follows:

```
// WRITE YOUR CODE HERE!
```

The easiest way to navigate the template code is to start from the `main` method of the `DataSet` class, shown in Figure 9. Intuitively, this method works as follows: First, it reads from the standard input the name of the CSV file to process. Next, it creates an instance of `DataSet`; the constructor of `DataSet` will process the input file and populate `attributeNames` and `matrix` (explained earlier). Finally, the `main` method prints to the standard output the metadata of the instance of the `DataSet` that was just created. Doing so requires calling the `metadataToString()` instance method. To better illustrate `metadataToString()`, Figure 10 shows the return value of `metadataToString()` by our reference implementation for the input file of Figure 1 (the instructors' reference implementation will be released to you after the assignment due date).

Please note that there are a couple of technicalities which you will learn about only later in the course. One is how to process files. The other (and much more important technicality, for that matter) is the notion of exceptions in Java. For this assignment, the template code for file processing is provided wherever it is needed. As for exceptions, you do not have to deal with them in this assignment, but you will see that some methods in the code are declared as throwing exceptions. You can ignore these exception declarations for now.

```
	public static void main(String[] args) throws Exception {

		System.out.print("Please enter the name of the CSV file to read: ");

		Scanner scanner = new Scanner(System.in);

		String strFilename = scanner.nextLine();

		DataSet dataset = new DataSet(strFilename);

		System.out.print(dataset.metadataToString());
	}
```

Figure 9: The `main` method in `DataSet`

```
Number of attributes: 5
Number of datapoints: 14

* * * Attribute value sets * * *
1) outlook (nominal): ['sunny', 'overcast', 'rainy']
2) temperature (numeric): [85, 80, 83, 70, 68, 65, 64, 72, 69, 75, 81, 71]
3) humidity (numeric): [85, 90, 86, 96, 80, 70, 65, 95, 75, 91]
4) windy (nominal): ['FALSE', 'TRUE']
5) play (nominal): ['no', 'yes']
```

Figure 10: String returned by `metadataToString()` for the input file of Figure 1

Util

The `Util` class is provided in order to faciliate the implementation of the `metadataToString()` method, which you will be implementing in the `DataSet` class (see the template code). The `Util` class provides four static methods:

- `public static boolean isNumeric(String str)`: Checks if `str` represents a numeric value
- `isArrayNumeric(String[] array)`: Checks an array of strings, `array`, and returns true if and only if `array` is non-empty and all its elements represent numeric values
- `public static String nominalArrayToString(String[] array)`: Produces a string representation of an array of nominals. Note that all nominal labels are embraced with single quotes.
- `public static String numericArrayToString(String[] array)`: Produces a string representation of an array of numerics. Unlike nominals, numeric values are not embraced with single quotes in the representation.

JUnit Tests

We are providing a set of JUnit tests for the class *DataSet*. These tests should help make sure that your implementation is correct. They can further help clarify the expected behaviour of this class, if need be. Please note that the `DataSetTest` class assumes that the following CSV files are located in the current directly: `credit-info-with-commas.csv`, `weather-nominal.csv`, `credit-info.csv`, `weather-numeric.csv`, `large.csv`, `weather-with-spaces.csv`, and `missing-value`
You can find all these CSV files in the `datasets` directory of the template zip file you have been provided with.

Academic Integrity

This part of the assignment is meant to raise awareness concerning plagiarism and academic integrity. Please read the following documents.

- https://www.uottawa.ca/administration-and-governance/academic-regulation-14-other-important-informat
- https://www.uottawa.ca/vice-president-academic/academic-integrity

Cases of plagiarism will be dealt with according to the university regulations. By submitting this assignment, you acknowledge:

1. I have read the academic regulations regarding academic fraud.
2. I understand the consequences of plagiarism.
3. With the exception of the source code provided by the instructors for this course, all the source code is mine.
4. I did not collaborate with any other person, with the exception of my partner in the case of team work.
    - If you did collaborate with others or obtained source code from the Web, then please list the names of your collaborators or the source of the information, as well as the nature of the collaboration. Put this information in the submitted README.txt file. Marks will be deducted proportional to the level of help provided (from 0 to 100%).

Rules and regulation

- Follow all the directives available on the assignment directives web page.
- Submit your assignment through the on-line submission system virtual campus.
- You must preferably do the assignment in teams of two, but you can also do the assignment individually.
- You must use the provided template classes below.
- If you do not follow the instructions, your program will make the automated tests fail and consequently your assignment will not be graded.
- We will be using an automated tool to compare all the assignments against each other (this includes both, the French and English sections). Submissions that are flagged by this tool will receive the grade of 0.
- It is your responsibility to make sure that BrightSpace has received your assignment. Late submissions will not be graded.

Files

You must hand in a **zip** file (no other file format will be accepted). The name of the top directory has to have the following form: **a1_3000000_3000001**, where 3000000 and 3000001 are the student numbers of the team members submitting the assignment (simply repeat the same number if your team has one member). The name of the folder starts with the letter "a" (lowercase), followed by the number of the assignment, here 1. The parts are separated by the underscore (not the hyphen). There are no spaces in the name of the directory. The archive a1_3000000_3000001.zip contains the files that you can use as a starting point. Your submission must contain the following files.

- README.txt
    - A text file that contains the names of the two partners for the assignments, their student ids, section, and a short description of the assignment (one or two lines).
- DataSet.java
- Util.java[2]
- StudentInfo.java (Make sure to update the file, so that the `display()` method shows your personal information).

[2] You are **not** supposed to change Util.java; you simply resubmit the file given to you.

Questions

For all your questions, please visit the Piazza Web site for this course:

- https://piazza.com/uottawa.ca/winter2021/iti1121/home

Last modified: January 13, 2021