Northeastern University
CS6220 – Data Mining Techniques
Fall 2017, Derbinsky

Homework 1

1 Data Wrangling: File Format (10 pts)

A common task in data mining is data *wrangling* (or *munging*): the process of manually converting or mapping data from one "raw" form into another format that allows for more convenient consumption with the help of semi-automated tools.[1]

In this problem you will write Python 3 code to convert data from one format to another. There are many common formats, but we will focus on two: ARFF and CSV. An ARFF (Attribute-Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes[2]. ARFF is the default file format of Weka[3], and so could be quite useful for fast prototyping of data mining models. A comma-separated values (CSV) file stores tabular data (numbers and text) in plain text[4]. CSV is a fairly universal format, and is readable both by Microsoft Excel and Weka.

Your task is to write a Python program, `csv2arff.py`, which takes as its argument[5] the path to a CSV file, and prints to the console an equivalent file in ARFF format[6]. To simplify matters, you can make some assumptions:

- The first row of the file will contain attribute names – the ARFF attribute list should mirror this order
- All data will be either numeric or nominal[7] – assume numeric if possible
- You should **not** use the Sparse ARFF format (we'll get to that later!), but you should be able to handle missing values, which translate to `?`
- The output relation name should be the name of the file, without any path/extension information[8]
- The set of values for nominal attributes should be listed in sorted order, irrespective of the order in which they appear in the CSV
- The data rows in the ARFF should appear in the same order as in the CSV

Since there are subtleties in various CSV dialects, it would behoove you to use Python's `csv` module[9] rather than writing your own parser from scratch. You have been provided input/output samples based upon Weka's example data sets.

[1] `https://en.wikipedia.org/wiki/Data_wrangling`
[2] `http://weka.wikispaces.com/ARFF`
[3] `http://www.cs.waikato.ac.nz/ml/weka/`
[4] `https://en.wikipedia.org/wiki/Comma-separated_values`
[5] `https://docs.python.org/3/library/sys.html#sys.argv`
[6] Weka actually has this functionality, but it's good to know how to code this up yourself.
[7] `https://en.wikipedia.org/wiki/Level_of_measurement#Nominal_scale`
[8] See `basename`: `https://docs.python.org/3/library/os.path.html#os.path.basename`
[9] `https://docs.python.org/3/library/csv.html`

2 Data Wrangling: Feature Scaling (10 pts)

A common preprocessing task is to *rescale* (or *normalize*) a dataset: that is, adjust values in one or more features to be within a common scale[10]. In this problem you will write Python 3 code that inputs data from a file, normalizes within fixed bounds, and then outputs the normalized data to the console with a specified precision.

Your task is to write a Python 3 program, `normalize.py`, which takes as its arguments[5] (i) the path to a text file of data – one instance per line, space separating each element, where you can assume the dimensions will be consistent between lines; (ii) the lower; and (iii) upper bounds of each feature, such as -1 – 1 or 0 – 1; and (iv) the number of digits to provide for each output value, rounding where necessary. For example: `./normalize.py 2_in.txt -1 1 4`

You have been provided input/output samples – the name of each output identifies the parameters supplied to create the file.

3 Dataset Analysis (30 pts)

Your task is to produce an iPython/Jupyter notebook[11] to analyze a public citation dataset from Aminer[12]. You will perform some exploratory analysis and data visualization for this dataset.

Note: this is a large dataset (about 2 million publications – it takes about a minute just to parse!). While your notebook must successfully work on the entire dataset, you may find it useful to work on a subset while getting your code to work.

Where there are explanation questions below, provide your responses as `markdown` cells within the notebook.

3.1 Basic Counts

a. Compute the number of distinct authors, publication venues, publications, and citations/references

b. Are these numbers likely to be accurate? As an example look up all the publications venue names associated with the conference "Principles and Practice of Knowledge Discovery in Databases"[13] – what do you notice?

3.2 Publications, Authors, Venues

a. For each author, construct the list of publications. Plot a histogram of the number of publications per author (use a logarithmic scale on the y axis).

b. Calculate the mean and standard deviation of the number of publications per author. Also calculate the Q1 (1st quartile[14]), Q2 (2nd quartile, or median) and Q3 (3rd quartile) values. Compare the median to the mean and explain the difference between the two values based on the standard deviation and the 1st and 3rd quartiles.

[10] `https://en.wikipedia.org/wiki/Feature_scaling`
[11] `https://en.wikipedia.org/wiki/IPython#Notebook`
[12] `https://aminer.org`
[13] `https://en.wikipedia.org/wiki/ECML_PKDD`
[14] `https://en.wikipedia.org/wiki/Quartile`

c. Now plot a histogram of the number of publications per venue, as well as calculate the mean, standard deviation, median, Q1, and Q3 values. What is the venue with the largest number of publications in the dataset?

3.3 References, Citations, Impact

a. Plot a histogram of the number of references (number of publications a publication refers *to*) and citations (number of publications referring *to* a publication) per publication. What is the publication with the largest number of references? What is the publication with the largest number of citations? Do these make sense?

b. Calculate the so called "impact" factor for each venue. To do so, calculate the total number of citations for the publications in the venue, and then divide this number by the number of publications for the venue. Plot a histogram of the results.

c. What is the venue with the highest apparent impact factor? Do you believe this number?[15]

d. Now repeat the calculation from item b., but restrict the calculation to venues with at least 10 publications. How does your histogram change? List the citation counts for all publications from the venue with the highest impact factor. How does the impact factor (mean number of citations) compare to the median number of citations?

e. Finally, construct a list of publications for each publication year. Use this list to plot the average number of references and average number of citations per publication as a function of time. Explain the differences you see in the trends.

4 The Lift Measure (10 pts)

In lecture we learned about the `lift` measure: $lift(X \Rightarrow Y) = \frac{c(X \Rightarrow Y)}{s(Y)}$

Provide a mathematical argument to justify the following statement: if a rule has a lift of 1, it would imply that the probability of occurrence of the antecedent and that of the consequent are independent of each other.

[15] `http://mdanderson.libanswers.com/faq/26159`

5 Apriori (10 pts)

Consider the following dataset of transactions:

TID	Items
1	{bananas, carrots, figs}
2	{apples, donuts}
3	{banana, carrots, donuts, figs}
4	{carrots, eggs}
5	{apples, eggs}
6	{donuts}
7	{apples, bananas, carrots}
8	{apples, carrots, eggs}
9	{carrots, donuts, eggs, figs}
10	{apples, donuts, figs}

a. Show all work for the first three passes of the Apriori algorithm (set sizes 1, 2, and 3) for a support threshold of 0.3. For each pass, list the candidate sets and then the frequent sets.

b. List all maximal frequent sets discovered in the first three passes.

c. Using these maximal sets, identify one association rule that has support of at least 0.3 and confidence at least 0.7. Show your work.

6 FP-Growth (10 pts)

Consider the following dataset of transactions:

TID	Items
1	{apples}
2	{apples, carrots}
3	{apples, carrots, donuts}
4	{apples, bananas, eggs}
5	{apples, bananas, carrots, donuts}
6	{bananas, donuts, eggs}
7	{bananas, carrots, figs}
8	{apples, bananas, carrots}
9	{apples, bananas, donuts, eggs}
10	{apples, carrots, eggs}

a. Construct an FP-tree (minimum support of 0.2) from the transaction database, showing **each step of the process**. Follow TID order and where there are ties in item counts, sort alphabetically (so if both X and Y have count 3, then X should precede Y in an itemset).

b. Illustrate **each step of the process** for finding frequent itemsets containing donuts. At each phase, show the FP-tree representation of the prefix paths, the conditional tree, and resulting frequent itemsets.

7 A Scaled FP-Growth Pipeline (20 pts)

Your task is to take a dataset of nearly one million clicks on a news site[16] and use the Weka Explorer to identify interesting association rules. Ordinarily this would be a point-and-click task; however, the input data format is a list of transactions (each line in the file includes a list of anonymized news item id's), whereas Weka requires a tabular format. Specifically, each distinct news item id should be represented via a column/attribute, and each row/instance should be a sequence of binary values, indicating whether or not the user visited the corresponding news item.

For example, consider the following example input data file `test.dat` ...

```
1 2 3
1
5
```

The corresponding ARFF representation would be ...

```
@RELATION test
@ATTRIBUTE i1 {0, 1}
@ATTRIBUTE i2 {0, 1}
@ATTRIBUTE i3 {0, 1}
@ATTRIBUTE i4 {0, 1}
@ATTRIBUTE i5 {0, 1}
@DATA
1,1,1,0,0
1,0,0,0,0
0,0,0,0,1
```

Notice how wasteful this representation is in the data section: for datasets where the number of items per row is relatively small relative to the number of columns (i.e. the data is *sparse*), you will find many 0's. For this simple example, there isn't much harm; but for larger datasets, the wasted space can be prohibitive (both for computation time and space/memory; for the dataset in this assignment, this *dense* file would take many minutes to produce and require many gigabytes of storage). To ameliorate this situation, there is a syntax[17] to make a sparse ARFF file ...

```
@RELATION test
@ATTRIBUTE i1 {0, 1}
@ATTRIBUTE i2 {0, 1}
@ATTRIBUTE i3 {0, 1}
@ATTRIBUTE i4 {0, 1}
@ATTRIBUTE i5 {0, 1}
@DATA
{0 1, 1 1, 2 1}
{0 1}
{4 1}
```

Notice now that each value of 1 is prefixed by an index of an attribute (starting with 0), and thus 0's need not be included. Two additional points: this syntax requires the indices be in increasing order, and

[16] Originally: `http://fimi.ua.ac.be/data/`

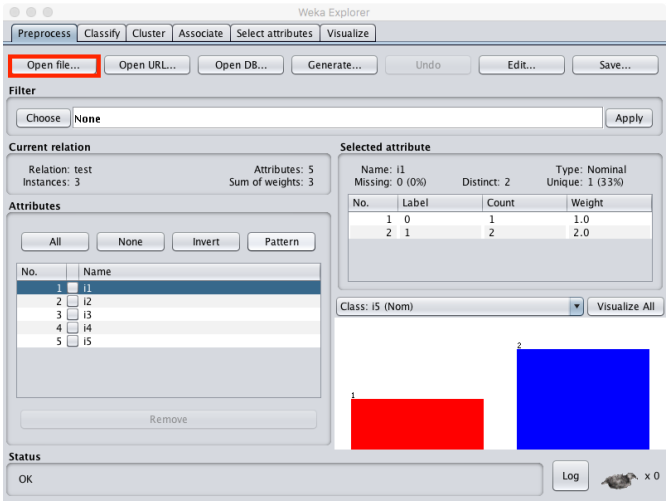
[17] `http://weka.wikispaces.com/ARFF+%28stable+version%29#Sparse%20ARFF%20files`

that there are no duplicates.

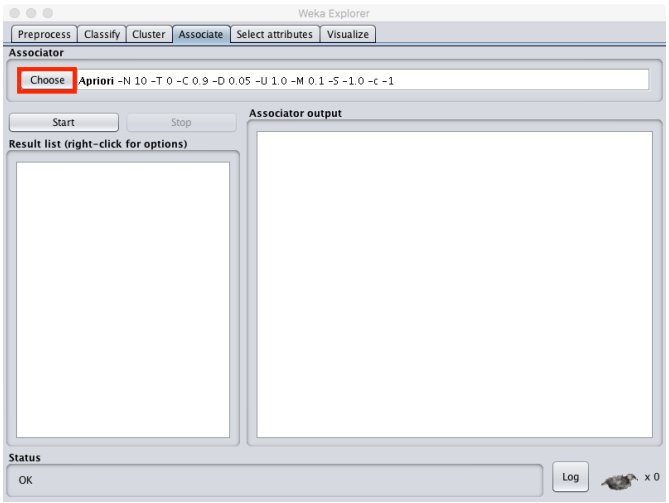
Given this file, you can now load the data into the Weka Explorer ...

1. Run Weka, and click the "Explorer" button in the Weka GUI Chooser.

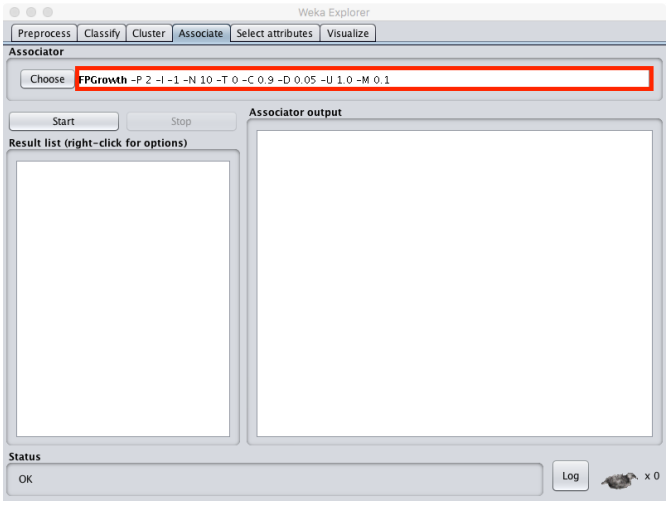
2. Click the "Open file" button, and then choose the ARFF file. Once it loads (which can take a while for larger files), you'll see some basic statistics about rows and columns.



3. Click the "Associate" tab and click the "Choose" button on the resulting pane.



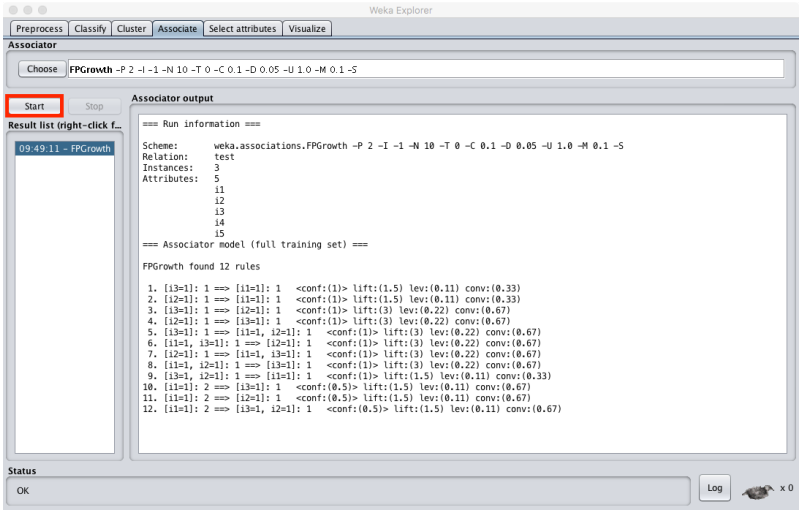
4. The resulting list has a few algorithms – click FPGrowth. In the resulting screen, click the textbox that has the name of the algorithm to configure it's properties.



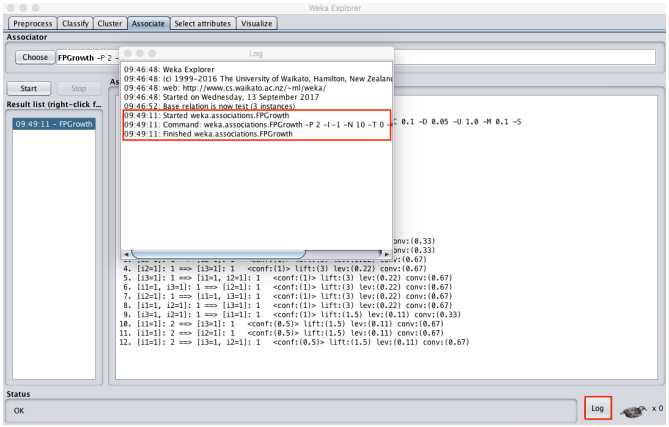
5. In the resulting screen, be sure to change `findAllRulesForSupportLevel` to `True`. Then, choose your minimum support (`lowerBoundMinSupport`) and confidence (`minMetric`). Click "Ok".

6. Now click "Start" – the result will show in the output pane to the right. The output includes rules,

confidence, and other meta data.



7. Finally, if you click the "Log" button, you can see approximately how long it took to execute the algorithm on this dataset (under a second, in this case).



Your Task

a. Write a Python program, `itemsets2sparsearff.py`, which takes as its argument[5] the path to a text file of data (assumed to be in the itemset format above) and produces as output to the console a sparse ARFF file.

b. Use your program to convert the `kosarak.dat` file to a sparse `kosarak.arff`. About how long did it take to run?

c. Load the resulting file into Weka (as described above; you should have $41,270$ attributes and $990,002$ instances). About how long did it take to load this file?

d. Use Weka's FP-Growth implementation to find rules that have support count of at least $49,500$ and confidence of at least 99% – record your rules (there should be 2).

e. Run the algorithm at least 5 times. Then look to the log and record how much time each took. How does the average time compare to the time necessary to convert the dataset and then load into Weka?