Problem Set 4
Due Sunday, December 3, 2017 at 11:55pm

How to Submit

Create one .zip file (**not** .rar or something else) of your code and written answers and submit it via `ilearn.ucr.edu`. Your zip file should contain the following 3 files

- q1.txt or q1.pdf
- findrules.m
- q3.pdf

plus any other Matlab functions your code depends on that you wrote.

Each file should include at the top (in comments if necessary)

- Your name
- Your UCR student ID number
- The date
- The course (CS 171)
- The assignment number (PS 4)

Note: Do not use any Matlab function that is in a toolbox for this problem set.

Problem 1. [5 pts]

If there are m items (or features), there are $3^m - 2^{m+1} + 1$ different association rules possible. Prove this by writing down a formula for the number of different rules and explaining clearly why this formula is correct, and then simplify this formula to obtain the answer. Submit your answer in the file `q1.txt` or `q1.pdf`.

Hint: Think about how items can be divided up to make rules. Also consider what divisions result in invalid rules (rule $X \to Y$ must have non-empty X and Y). It might help to know that $(1+x)^n = \sum_{i=0}^{n} \binom{n}{i} x^i$.

Problem 2. [15 pts]

You are to write code to perform association analysis on the data supplied in `groceries.txt`. To help the following functions have been supplied.

- `D = loaddata(filename)` will return an object representing the dataset (to be used as the `D` argument in the functions below. You should only access `D` through the functions below.
- `num = getcount(set,D)` will return the number of transactions in the dataset `D` for which all element of the set `set` are present. `set` should be a vector of integers, each representing a different item.
- `I = items(D)` will return a vector of all of the items (as integers) in the dataset `D`, in sorted order.
- `m = numexamples(D)` will return the number of transactions (or examples) in the data `D`.
- `str = rule2str(X,Y,D)` will return a string representing the rule encoded by $X \to Y$, where X and Y are represented by vectors of integers.

You are to write a function `findrules(D,smin,amin)` that accepts a dataset (as above), a minimum support, and a minimum confidence and writes (to the console) a list of all rules that meet those restrictions, sorted by confidence. For example:

```
>> D = loaddata('groceries.txt');
```

```
>> findrules(D,0.01,0.5)
0.500000, 0.012913 : {yogurt, root vegetables} => {other vegetables}
0.502092, 0.012201 : {rolls/buns, root vegetables} => {other vegetables}
0.507042, 0.014642 : {other vegetables, whipped/sour cream} => {whole milk}
0.512881, 0.022267 : {yogurt, other vegetables} => {whole milk}
0.517361, 0.015150 : {tropical fruit, yogurt} => {whole milk}
0.517510, 0.013523 : {pip fruit, other vegetables} => {whole milk}
0.523013, 0.012710 : {rolls/buns, root vegetables} => {whole milk}
0.524510, 0.010880 : {yogurt, whipped/sour cream} => {whole milk}
0.552511, 0.012303 : {other vegetables, domestic eggs} => {whole milk}
0.562992, 0.014540 : {yogurt, root vegetables} => {whole milk}
0.570048, 0.011998 : {tropical fruit, root vegetables} => {whole milk}
0.573604, 0.011490 : {other vegetables, butter} => {whole milk}
0.582353, 0.010066 : {yogurt, curd} => {whole milk}
0.584541, 0.012303 : {tropical fruit, root vegetables} => {other vegetables}
0.586207, 0.010371 : {citrus fruit, root vegetables} => {other vegetables}
```

You should use the apriori algorithm shown in class to enumerate all sets of large enough support. There are a few important points to make this more efficient:

- If the itemsets are kept in sorted order, and the sets of itemsets for each level are kept in the order in which they are generated, then the Apriori-Gen algorithm from class can be simplified and made faster:
- When generating the set $\{1, 3, 5, 7\}$ (at level 4), there are a number of pairs of sets from level 3 that could be used: $\{3, 5, 7\}$ and $\{1, 5, 7\}$, or $\{3, 5, 7\}$ and $\{1, 3, 7\}$, or $\{3, 5, 7\}$ and $\{1, 3, 5\}$, or $\{1, 5, 7\}$ and $\{1, 3, 7\}$, or $\{1, 5, 7\}$ and $\{1, 3, 5\}$, or $\{1, 3, 7\}$ and $\{1, 3, 5\}$ We only want to generate $\{1, 3, 5, 7\}$ once, however. Therefore, we will insist that all but the last elements (if taken in sorted order) of each for the two sets to be merged, match. This means that of the 6 pairs above, only the last will be merged (because the first 2 elements are the same).
- To further reduce the processing time, if the sets at the previous lattice level are lexigraphically sorted (and they will be if generated in the way described above), then when finding a match for set index i, we need to only consider set indexes greater than i. Furthermore, the ones that match will be a consecutive block. Therefore, as soon as we find the first non-match to i after i we can stop.
- Taken together, this means that the pseudo-code for the Apriori-Gen function from class can be rewritten as

function Apriori-Gen(L_{i-1})
▷ From large items sets of size $i-1$, generate candidate large items sets of size i
▷ assume L_{i-1} is an ordered list of sets
$C_i \leftarrow \{\}$
for all $j \in \{1, \ldots, |L_{i-1}|\}$
for all $k \in \{j+1, \ldots, |L_{i-1}|\}$
if $L_{i-1}[j]$ and $L_{i-1}[k]$ agree on all but their last elements
$C_i \leftarrow C_i \bigcup \{L_{i-1}[j] \bigcup L_{i-1}[k]\}$
else
break
return C_i

- Note that taking the union of $L_{i-1}[j]$ and $L_{i-1}[k]$ is particularly simple, as only the last element is different. However, to make this whole process work, the elements must remain (in the union set) in sorted order.

A few notes about implementation in Matlab:

- You should keep itemsets as vectors. For instance the set {citrus fruit, tropical fruit, yogurt} should be kept as $[0, 4, 5]$. Keep them in sorted order (or the above Apriori-Gen algorithm will not work).

  Matlab has operations for sets of this type: `union`, `intersect`, `setdiff`. You should only need the last of these, but may use any of them.

- You will need to keep sets of itemsets. These (because the itemsets will have different lengths) cannot be kept as vectors or matrices. They will need to be kept as cell arrays.

  You can create an empty cell array as `C = {}`.

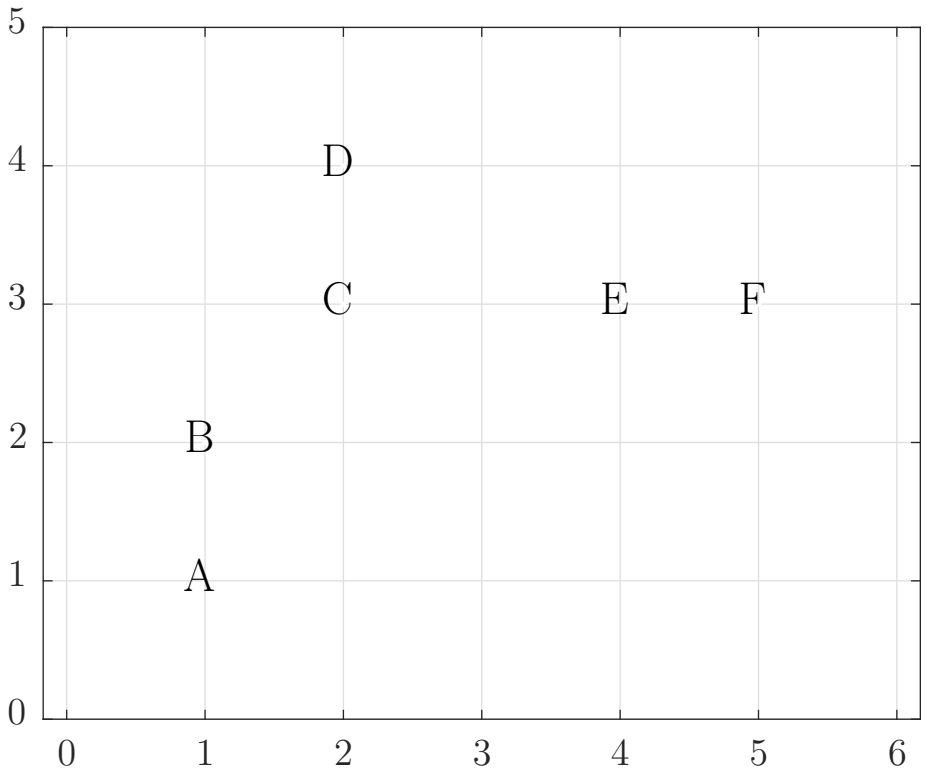
  You can add an element on to the end of a cell array as `C = [C e]` (to add the element `e` to the end of `C`).

  You can find the number of elements in a cell array as `l = length(C)`.

  You can access the `i`th element of a cell array as `C{i}` (1-based indexing).

Problem 3. [5 pts]

Draw three dendrograms (one for single-linkage, one for average-linkage, and one for complete-linkage) for the data below. Clearly label your vertical axis to show the level at which the merges happen. Each point has integer coordinates.



Place your answer in the file q3.pdf.