Problem Set 4
Due Monday, May 28, 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

- runq1.m
- q2.txt or q2.pdf
- findrules.m

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]

In this problem, you are to use a decision tree to classify points. The training data are in the file bank-train.data. The first 19 columns are the features. The last column is the binary label: whether or not a customer responded positively to a phone offer to buy a certificate of deposit from a bank. The full details of the features can be found at `http://archive.ics.uci.edu/ml/datasets/Bank+Marketing`. Split the training data, using the last 35% as the pruning set (and the other as the set for growing the tree).

Some of the features are categorical, and some are numeric. The supplied code can handle either type of feature. Here are the supplied functions (all given as `.p` files, but they work just like `.m` files).

- `dt = learndt(X,Y,ftypes,scorefn)`. `X` and `Y` are the training set as you are familiar with from other assignments. `ftypes` is a vector reporting whether each feature is numeric or categorical. In particular, the corresponding value should be `0` if the feature is numeric and equal to the number of categories if the feature is categorical. For the bank dataset the vector should be (you may hard-code this into your function)

  $$\texttt{ftype} = \begin{bmatrix}0 & 12 & 4 & 8 & 3 & 3 & 3 & 2 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0\end{bmatrix}$$

  The `scorefn` should be the function that will score a leaf. This function(s) you will write. The function should take in the fraction of examples of label 1 and return the score. To pass a function into another function, add `@` before the function's name. For instance, if you named your scoring function `giniscore`, then you would call `learndt` as `learndt(X,Y,ftypes,@giniscore)`.

- `drawdt(dt)`: `dt` is a decision tree (as learned from above). This function will draw it on the console.

- `predictdt(dt,X)` will return a vector of the categories for each row in `X`, classified according to the decision tree given by `dt`.

- `pdt = prunedt(dt,X,Y)` will returned the pruned verson of the decision tree `dt`, pruned according to the data given by `X` and `Y`.

You are to write a function called `runq1` that takes no parameters and returns a vector of the predicted values on the testing data given in the file `banktestX.data` (which is missing the last column). `runq1` should also return the learned decision tree: `[Y,dt] = runq1()`.

Problem 2. [5 pts]

If there are m items (or features), there are $3^m - 2^{m+1} + 1$ different association rules possible. Prove this. Submit your answer in the file `q2.txt` or `q2.pdf`

Hint: Think about how items can be divided up to make rules. Also consider what divisions result in invalid rules (rule $X \rightarrow Y$ must have non-empty X and Y).

Problem 3. [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 $X \rightarrow Y$, 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:

  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 $\{\text{citrus fruit}, \text{tropical fruit}, \text{yogurt}\}$ 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).