Data Mining Project 2: Frequent Pattern Analysis

Given a dataset D for classification, students will implement a Java program to perform analysis of frequent patterns mined from D.

Before frequent pattern mining, you need to transform D using discretization. You can use any binning method that can give good results. The discretization method can be viewed as a function mapping attribute-interval/value pairs to distinct IDs (natural numbers); store this map in a file called "DiscretizationMap.csv" (with one line for each interval/value and natura-number pair). Store the discretized version of D in a file called "DiscretizedD.csv". You code should be written in a way to make it easy to use other binning methods. To perform pattern mining, you call FP-growth within your program. You can use any FP-growth implementation in java or C or C++ that you can find from the web; I found http://www.borgelt.net/fpgrowth.html and https://github.com/integeruser/FP-growth but I have not tested them. You can use FP-Growth through a program call inside your program.

Your program will perform the following tasks.

1) Implement a function to compute the closed patterns and their minimal generators. The output should be written into a file called "ClosedAndMG.csv", with one line for each closed pattern, its support in D and its minimal generators (where patterns are in the format of {3,26,238} with integers representing the attribute-interval/value pairs as given in the Map discussed above).
2) Implement a function to compute bit-set representations of the matching datasets (mds) of patterns, and implement functions that use word-based bit-set operations to compute the intersection/difference etc of the mds of two patterns. The functions can be used to compute the Jaccard similarity (defined in terms of mds) etc.
3) Implement a function to compute the closed Emerging Patterns (EP) with high growthRate or supportRatio from the mined frequent patterns. The growthRate of an EP P is defined to be max(sup(P,C1)/sup(P,C2), sup(P,C2)/sup(P,C1)). You need to compute those two supports.
4) Implement a function to compute a diversified set PS of K closed EPs that maximize

   $[AVG_{P \text{ in } PS}\ \text{growthRate}(P)] * [1 - AVG_{P,Q \text{ in } PS}\ \text{Jaccard}(P,Q)]$.

   Store info on the computed PS in two files called "PSkEPs.csv" (for the K EPs – one EP per line containing the closed EP, its growth rate, and its supports in the two classes) and "PSkEPJaccard.csv" (each line containing two closed EPs in PS and their Jaccard similarity).

You should provide a make file or a similar file to compile your java program. The executable file should be called p2pa; it takes three commandline arguments: datafilename, minSup, and minGrowthrate.

Extra Credit Tasks

5) Change items 3 and 4 above so that you only work with such closed EPs P where a new local classifier built for mds(P) can significantly reduce the classification error on mds(P) made by a fixed global classifier. Here you use implementations of NBC, Decision Tree, or Logistic Regression (in weka or R) to build the global classifier or local classifiers. The objective function should be

   $[AVG_{P \text{ in } PS}\ ER(P)] * [1 - AVG_{P,Q \text{ in } PS}\ \text{Jaccard}(P,Q)]$

   where ER(P) is the average relative classification error reduction on mds(P).

6) You can also handle another data mining task such as regression, clustering, and outlier detection.
7) Instead of using FP-growth's mining result to produce patterns, push the interestingness analysis into the FP-growth algorithm.

The data for classification should be in csv format, with class as the first column. Your program will count the number of values for non-numerical attributes and will need to test if the values for an attribute are all numerical.

Hints: You may want to speed up the computation of pattern analysis by partitioning the set of frequent patterns based on the support values of the patterns. Hashing of patterns for each hash bucket (having identical supports) may also help.

You should use some heuristic method to search for an optimal PS. The instructor found some related methods for problems such as the Team Formation problem, the independent set problem, and the K-center problem. Some papers may be added to the project folder. Wikipedia pages also exist on solving such problems. Please note that you need to transform our given problems in order to use algorithms for those problems.

Project Submission Guidelines: Turn in all files needed to compile and execute your code (including a make file) via pilot. Put all files into a folder named as YourLastnameP2 and zip it. That folder can have sub folders but not sub-sub folders. We should be able to run your program just after typing "make" inside your directory/folder.

In the submitted folder, include a file called Report.pdf, to summarize your findings and other relevant information.

If there are any special instructions that we need to know about, be sure to include a file named README.TXT detailing them. Limit such instructions to a minimum. In this file you can also discuss limitations of your implementation.

You should not deviate from the guidelines and requirements discussed above. You should follow good programming/documentation practices.

Correctness, efficiency, readability etc will be factors for marking.

In your code, you must include comments that show the major steps and tasks.

A demo may be scheduled for students when their projects cannot be successfully run by the professor/grader.

Marks Available: This project will attract 18% of the total 30% allocated to projects in this course.

Extra credit tasks can attract a total of 10% on top of the 18%. We will determine the amount of extra credit for individual students based on the quality of their solutions, the challengingness of the problems they address, and the efficiency of their programs.

Extra credit (2%) will also be given to submissions that contain a different program that can check the correctness and quality of outputs produced by p2pa (including those by other students).