University of Ioannina,
Department of Computer Science & Engineering

Data Mining – CSE 012/CS 059
Fall 2014

Assignment 1

This is the second part of Assignment 1. The deadline for this part is at the beginning of the class on November 25th. For late submissions the late policy on the page of the course will be applied. The details for the turn-in are on the page of the course.

Question 1

Assume that you are given as input a table with *n* rows and *m* columns, with 0/1 values. You want to find all *(r,c)-tiles* of 1's, that is, sets of *r* rows and *c* columns such that the corresponding submatrix has all 1's. Note that tiles may be overlapping. Describe an efficient algorithm for solving this problem that makes use of the APriori idea.

Question 2

The goal of this question is to use frequent itemset mining in practice.

As the input dataset you are given the file "anonymized_grades_data.txt" which has the (anonymized) grades of students for different courses. It is a tab-separated file with three columns: The first column contains a unique id for each student; The second column has the course code; The last column has the grade with which the student has passed this course.

Preprocess the file to produce a "basket" for each student, containing the courses that the student has passed with grade at least 0.7. Then use an implementation from the FIMI repository for mining frequent itemsets to find courses that appear often together. The link to the FIMI repository is on the page of the course. You may need to transform your data to the format required by each package. Use a high enough support threshold so that you do not get a huge amount of frequent itemsets. Use the file "course_names.txt" that maps course codes to their name in order to make the output easier to interpret. Then inspect the results and comment on itemsets that you found.

You should turn in the following:

- All the code that you have written yourselves.
- The output frequent itemsets.
- A short report where you describe how you did the preprocessing, the choice of support threshold, and the commentary on the output.

Bonus: Compute the closed and maximal itemsets. Also report experiments with different support thresholds.

Question 3

The goal of this question is to experiment with recommendation algorithms.

Using the (dataset) from Question 2, remove a randomly selected 10% of the student-course records. The goal is to estimate these grades using the collaborative filtering techniques we saw in class. In order to compute the grade of student S for course C, you will use the following algorithm. Using the remaining 90% of the data, you will find for student S the K most similar users according to the cosine similarity. Then you will estimate the grade of student S for course C, as the average grade of the K most similar users for this course.

You will also implement two simple algorithms for comparison. The first compute the grade of student S for course C, as the average of the all the observed grades of student S (the idea being that the quality of the student determines the grade for the course). The second algorithm estimates the grade of S for C as the average of all observed grades for C (the idea being that the hardness of the course determines the grade).

For the evaluation and comparison of the algorithms you will use the Sum of Square Errors metric. If $g_1, g_2, \dots, g_n$ are the grades that we want to predict, and $p_1, p_2, \dots, p_n$ are the predictions of the algorithm, then the Sum of Square Errors of the algorithm is defined as

$$SSE = \sum_{i=1}^{n} (g_i - p_i)^2$$

You should turn in the following:

- All the code that you have written yourselves.
- The file with the selected 10% that you are trying to predict and the remaining 90%.
- A short report with a description of what you did, and a commentary on the output.

Bonus: Your grades are part of this dataset. If you want I can tell you to which student id they correspond and you can add a comment on the results of the algorithms for your own grades.

Question 3

The goal of this question is to experiment with Min-Hashing and Locality Sensitive Hashing for real vectors.

You will use the same dataset as in Questions 2 and 3. A small modification that you need to do is to normalize the vectors by subtracting the mean value for each row of the data matrix.

First, you will compute the exact cosine similarity between all pairs of students, and you will print the pairs that have similarity at least 0.9. Then you will produce random vectors with coordinates -1 or +1 (randomly chosen) and you will compute the projection of the student vectors on these vectors. You will experiment with 50, 100, and 200 random vectors. The vector of the projection values for each student

defines the “signature” for the student. You will then approximate the similarity between the two students as the cosine similarity of the signatures, and you will output the pairs with approximate similarity at least 0.9. Compute the number of false positives and false negatives. Report the mean value for 5 different runs.

You will then simplify the signature vectors, transforming them into vectors with +1,-1 values, depending on the sign of the projection. Then break up the signature table into b bands (mini-signatures) with r projections for each band, as described in class, and implement Locality Sensitive Hashing. The goal is to find candidate pairs with similarity at least 0.9. Use the table with the 100 projections and experiment with r=5 and b=20, r = 10 and b=10, r = 20 and b = 5. Report again the number of false positives and false negatives, average over 5 runs.

Technical details

1. For pre-processing the data, some unix commands that you may find useful are the following:
    a. cut: allows you to get specific columns from delimited data
    b. sort: sorts the rows of a file in lexicographic order, –n for numeric
    c. uniq: merges consecutive rows of a file that are identical.
2. For the implementation of LSH you do not need to implement a hash table or a list. Use existing implementations that come with the language (e.g., in C++ there is the STL implementation). Also it may be convenient when hashing the mini-signatures to convert them into strings with ‘+’ and ‘-‘ characters.
3. Although it is not efficient, for simplicity you can instantiate the full student-course matrix. You will get bonus points for a more efficient implementation.