Description

For this assignment, you will look for frequent itemsets and association rules in a movie ratings dataset. See if you can learn about what movies tend to be watched by the same people.

The data

We'll be using MovieLens data for this assignment. The MovieLens project is a movie recommendation project based at the University of Minnesota. Using movie recommendation data isn't precisely the right sort of data one uses for association rules, but if you throw away the rating information and think of each person as a "basket," it works perfectly and is fun.

Here's a caveat. There are better algorithms for making movie recommendations based on ratings that we'll look at later in the term. I'm merely using the MovieLens data here because it's an easily obtainable example that works. It is actually extremely difficult to find datasets that are better fits for association rules data that are freely available, easy to interpret, and produce results that are at least somewhat interesting. I've tried multiple times and failed. If you can find an interesting free dataset that is well-suited for association rules problems, please send me a link to it. Many of them are designed for showing how fast your algorithm is, and so they merely have item ids without description; so you can't interpret the result. I was excited about this bakery dataset for a while, until I realized that the products were nearly entirely limited to different types of cakes and tarts. Learning that people who buy lemon cookies also buy apple danishes seemed underwhelming and only made me hungry.

The MovieLens datasets come in three sizes. We'll be using the medium sized dataset, which contains 1 million ratings over 6040 users. The larger dataset is actually more interesting, but starts to run considerably slower unless you think really carefully about how to optimize everything you're doing. I'm declaring that we'll use the medium dataset as the basis for this assignment, but after you have everything working for the medium-sized dataset, feel free to experiment with an additional version of your project that works on the large dataset if you like. The large dataset is formatted a little differently, however, so make sure the version that you actually submit works with the medium one.

When you download the zip file, *make sure that you do not put it into your shared network directory.* If you are using a lab machine, put it in `/tmp`. You'll find within it a "ratings" file in each where each line contains a user id, a movie id, and some other information. It's the user id and the movie id that you want. When you read this data, you should transform each user into a "basket" so that you end up with a set of movies watched for each user. Throw away the rating info. Do this all in memory; there's no need to write out another file containing the rearranged data, since this will all fit in memory.

Frequent itemsets and Association Rules

Implement the Apriori algorithm to find frequent itemsets on this dataset. Your program should ask the user to input a support threshold and a confidence threshold. First, your program should print out all frequent itemsets that have this level of support or higher. Though you should use movie ids only throughout your entire program, at this point you should connect those movie ids back to their names so that the output is more interesting to look at. Second, you should print out all association rules that you can generate from those frequent itemsets that meet the confidence threshold.

Efficiency and Grading

Grading this assignment is partially easier than some other recent ones in that there is a clear right answer for a given support and confidence threshold. The trickier challenge is to see if you are writing your code efficiently. There are many, many things that you can do to make this fast, some of which we've discussed in class. I'd encourage you to play with your code and see what you can do. There are two main techniques, however, that I want to make sure that you use:

1. You should particularly implement the approach we discussed where you generate candidates of size $n + 1$ by combining frequent itemsets of size n that match on the first $n - 1$ terms, and then eliminating those with other subsets that are not frequent.
2. You should count candidate itemsets efficiently by storing their counts in a hash tree.

Both of these approaches, which we're doing in class, are described well in another textbook that we're not using. That textbook is not available in its entirety online, but the portion that describes the above content is. You can read about these approaches in the Tan, et al. data mining book; the content is in Chapter 6. The first approach above is described on page 341 as the $F_{k-1} \times F_{k-1}$ method; the hash tree is explained on page 344.

You should print out the number of candidate itemsets you have of each size right before you go back to the data to actually count. This will also help the graders determine if you are implementing as specified.

I'm going to be instructing the graders to look at your code in order to verify that it appears you're using the above approaches correctly. You must make it as clear in your code as you possibly can how you are doing it. Write your code in a way so as to be as clear as possible, and supplement with comments to explain what you are doing. Your job is to make a convincing argument with your code that you are doing this correctly. I'll be instructing the graders that they can take points off if they cannot determine to their satisfaction that these algorithms are not correctly implemented;; this could be because they are not done right, or because the code is hard to understand (or both!).

Breaking it Down

Part 1: Turn in a program that takes as input a support threshold, and prints out all itemsets of size 1 that have that level of support. You don't need to do this using any of the fancier approaches described above, though you can. The goal of part 1 is to get you started.

Part 2: Turn in the entire assignment as described above.

Optional extension for fun: Many of the rules that you get will be true, but obviously so. That's a great way to confirm that your code is working, but also lacks some zing. Experiment with approaches to get *interesting* rules. If you do this, make sure to distinguish it as separate from the standard assignent as specified.