

HW 1 – Frequent Pattern Mining

Submission: hw01.ipynb, hw01.pdf, pushed to your Git repo on master.
Points: 50
Pair work: REQUIRED
Due: Friday, February 28, 2pm

Objectives

- Apply frequent pattern techniques and association rule mining on written exercises
- Work with a real-world dataset to explore how frequent pattern methods can be used.

Directions


- Part I – [INDIVIDUAL WORK] Written questions, noted with [PDF] below, will have your answers submitted in a single file called **hw01.pdf**. The file must be submitted in your Git repository. (i.e. Save your work as a PDF in Word or your favorite word processor, or scan your handwritten notes as a PDF. How you do them is up to you, as long as you submit a PDF.)
- Part II – [PAIR PROGRAMMING] - Coding questions, noted with [PYTHON], should be submitted as a single notebook file named **hw01.ipynb**. Both partners should be clearly indicated in the top header cell, just like you do with your labs.

Part I – Frequent Pattern Exercises

This section includes five distinct exercises to help with your understanding of frequent pattern mining and association rule generation. All of these exercises are to be completed WRITTEN (or using a text editor of your choice, and submitted as a single PDF file, as noted above.

Exercise 1 – The apriori algorithm: [PDF]

This first exercise is based off of Exercise 6.6 in the text. The transaction list for the exercise is reproduced for you here, with a minor adjustment to the last transaction:

<i>TID</i>	<i>items_bought</i>
T100	{M, O, N, K, E, Y}
T200	{D, O, N, K, E, Y}
T300	{M, A, K, E}
T400	{M, U, C, K, Y}
T500	{C, O,  , K, I, E}

Since we do not keep track of duplicates at this early stage of frequent pattern mining, I crossed out the extra occurrence of "O" in transaction T500 above. For this exercise, assume $min_sup = 60\%$ and $min_conf = 80\%$

- Find *all* frequent itemsets using the **Apriori algorithm**. Show your work. Recall that the first phase of the algorithm focuses on finding frequent itemsets using candidate generation and elimination. Report the results in the order they would appear in the algorithm. (i.e., first, report the 1-itemsets, then 2-itemsets, etc.). Starting with 2-itemset generation, show the candidates generated, and clearly indicate *relative* support values for frequent itemsets. For candidates eliminated, indicate if they were eliminated because of the *Apriori property*, or because the candidate did not meet the min_sup threshold. For example, your answer should start something like:

1-itemsets:

$\{M\} : 0.6$

$\{O\} : 0.6$

$\{N\} : \text{does not meet min_sup}$

...

- What is a **closed frequent itemset**? List the set of **closed frequent itemsets** from the above list.
- What is a **max frequent itemset**? List the set of **max frequent itemsets** from the list in part (a)
- Generate all strong association rules from the frequent itemset above. Clearly indicate the *confidence* and *lift* measures for each rule generated.
- What is the strongest rule output? Justify your answer.

Exercise 2 – The FP-growth algorithm [PDF]

For this problem, you will work through the *FP-Growth* algorithm, using the same dataset as Exercise 1.

- Perform the first step and create the initial F-list, which creates the *ordered* list of 1-itemsets that are frequent. (You can use your answer above, since you already identified frequent 1-itemsets.)
- Create the initial FP-tree *and* corresponding F-list of items / support count / node links. Label this tree structure as **tree₀**.
- Execute the **FP_growth** algorithm. Start with **FP_growth(tree₀, null)**, and clearly indicate each step of the algorithm starting with the least frequent item. Indicate the recursive call with parameters to show what suffix and tree is currently being applied. Indicate when a frequent pattern is generated, and circle it. As you work through the algorithm, clearly show each conditional pattern base (CPB) and corresponding conditional FP-tree constructed with the CPB. Label each conditional FP-tree as tree_{suffix}, replacing *suffix* with the actual suffix being applied to generate the tree for the recursive call. Your resulting frequent patterns should be identical to the previous exercise.
- Compare and contrast the computational requirements (time and space) for both Apriori and FP-growth working on this exercise.

Exercise 3 – The Eclat algorithm [PDF]

- Convert the dataset in Exercise 1 to a vertical data format
- Find the frequent itemsets using the Eclat algorithm.

Exercise 4 – Correlation [PDF]

Consider the following contingency table for the occurrence of arbitrary items A and B in some transaction dataset.

	<u>A</u>	<u>NOT A</u>
<u>B</u>	65	40
<u>NOT B</u>	35	10

- Let $\text{min_sup} = 0.4$, and $\text{min_conf} = 0.6$. Compute support and confidence for the rule **A** \rightarrow **B**. Is this a strong rule?

- b) What does the *lift* measure tell us? Compute **lift(A,B)**. What does this suggest about the occurrence of A and B? What does it suggest about the rule?
- c) Compute the expected values for each observed value above, showing your results in a table.
- d) Compute the **X² correlation coefficient** using the table above and your expected values you computed in the previous question. Does the value imply dependency among A and B?
- e) Consider the rule **A → NOT B**. What is the *support*, *confidence* and *lift* for this rule?
- f) What is the confidence and lift of the rule **NOT B → A**? You should notice there is an imbalance between your answer here and the previous question. Which rule is stronger? Why?
- g) Compute the *Kulczynski measure* for the items **A** and **NOT B**.
- h) Compute the *imbalance ratio* (IR) on **A** and **NOT B**. What do these results say? Does the result confirm your observations on questions e) and f) above?

Exercise 5: Distributed mining [PDF]

Complete exercise 6.9 in the book.

(HINT: Think about our in-class discussion on methods that reduce database scans. One of those methods focused on partitioning a transaction dataset...)

Part 2 – Applying frequent pattern mining: MovieLens [PYTHON]

You've learned a wide range of techniques in Python to preprocess data, perform exploratory data analysis, and most recently, you've learned how to identify frequent patterns and generate "interesting" association rules from these patterns. Though we have extensively covered the application of frequent pattern and association rule mining in the context of market basket transactions, the approach has a wide range of applications. Sometimes it takes some careful thought to determine how to best map your data to the types required for a particular method. For frequent pattern mining, we need *transaction* encoded data. Generally, if you can transform your data to a set of encoded binary observations in a meaningful, consistent way that does not lose any information contained in the original data, then there is a good chance you can apply frequent mining methods to obtain interesting results! One such possible area is with recommendation systems.

Consider the following scenario. You are working for a consulting firm hired by a popular movie streaming company, *Flicks-R-Us*, to improve their movie recommendation system. The company has received many complaints. Particularly disturbing are the complaints from parents wanting recommendations for their kids. For instance, one parent, who had a child who liked "Bob the Builder" wanted movies about tools and construction sites for their child. They were directed to watch "Saw" and "Texas Chainsaw Massacre". (Their child did not sleep for days.) Needless to say, *Flicks-R-Us* is in desperate need of a better recommendation system, and they are counting on you!

Preparation

GroupLens Research (<https://grouplens.org/datasets/>) has collected and made available several datasets for the research community. One of them is a movie ratings dataset collected from the **MovieLens** web site (<http://movielens.org>). Go to the page <https://grouplens.org/datasets/movielens/> and download the file **ml-latest-small.zip**. It represents 100,000 movie ratings applied to 9,000 different movies, collected from a total of 600 users. As of this writing, the file was last updated on 9/2018. For this project, only use the file **ml-latest-small.zip** and not the FULL dataset. Also, NEVER neglect the README file! It contains valuable information (i.e. metadata) about the data!

TODO: Download the zip file, place the file in your `data` folder, and extract it there. Take the time to read through the readme file, and get an understanding of the different files you are working with, and the variables stored in each dataset.

Phase I

The ratings file is a log of movies their customers have watched. For the first phase of the project, you can keep the problem simple. Ignore the actual numeric rating and timestamp variables, and convert the ratings file into a set of transactions, where universe of all possible items are movies. Then, each row is a customer, the items are actual movies they watched. Your objective is to **output a set of the strongest, most interesting association rules you can**. Try to generate at least 10-20 rules. A strong association rule can be interpreted as a potential recommendation. **Your rules must contain actual movie names, and not movie ids!**

Remember, you are generating a report as a Python notebook. Writing Python code with no markdown cells to discuss your process and interpret your findings will receive major points lost!

For example, one possible rule might be:

`["Stripes (1981)", "Big Lebowski, The (1998)"] → ["Happy Gilmore (1996)"]`

You are encouraged to format your rules in a readable way, and output the measures of interestingness. Do not just merely output the data frame of rules. The above is just an idea. Come up with a good, readable way to show your rules. And, output them in some order of interestingness. Discuss your findings.

Phase II - Genre

The client is interested in a restricted set of rules for specific genres. For this task, demonstrate your skill by selecting a genre of your own choosing. Select the subset of movies that match that genre, and rerun your rule generation algorithm. For example, if the genre is "Comedy", then all ratings of movies that have `Comedy` in the genre list should be selected. Run your algorithm on that subset, and generate a small set of strong rules. REPEAT THIS FOR THREE DIFFERENT GENRES OF YOUR OWN CHOOSING.

Discuss – is this a better method than considering all movies? Or worse?

Phase III – Genre Rules

The client has a bright idea. (Being a good agile developer, you eagerly respond positively, to ensure the client knows they are valued and part of your team. ☺) The client wants to take a more general view of genre. How? Create a new transaction dataset, where the item universe is now all possible genres, not movies. A transaction for each customer is then a list of genres collected over all movies they watched. The customer wants to understand both the general frequent patterns among these data and their support levels. Again, be sure to output a good set of strong rules. This can help the customer determine what types of movies they should invest in the most based on current genres most watched. (NOTE: This is going to amount to a very dense dataset, compared to the movies, and thus will require very different hyperparameters.)

Phase IV – Incorporating Additional Variables

Consider how you can use other variables? You have access to the numeric ratings, a unique timestamp for the rating, the year of the movie, and user-defined tags. Or, consider that, for the previous exercise, you ignored multiplicity of genres. What else can you do with all of these data? For instance, are there patterns with movie years? Could you create new items such as "70s", "80s", and so on for the decade of the movie and re-run your frequent pattern search? Could you combine the decade and the genre? Imagine if you could figure out how to generate rules that tell the client that people who like 80s movies are likely to watch "Comedy" or "Romance" with a given confidence level. And of course, what about the ratings!!! Why would you output a rule that contains a movie only given a rating of a 1 or a 2? You might be able filter these patterns and rules more intelligently!

For this last phase, come up with three different ideas that involve including additional variables in some way, and implement it. In all three cases, generate a new set of association rules. Depending on what you choose to do here, it will likely require that you filter rules out that do not meet certain criteria? Or, perhaps you could modify or rewrite your own variant of the apriori algorithm. You could rewrite apriori just to generate relevant frequent patterns, and still use mlxtend's association rules package, as long as the format of the data frame that is used as input into the association rule generation are consistent.

I have no specific requirements here. I want you and your partner to think. Be creative. Put yourself in the client's shoes. You have a lot of data. How can you leverage it to provide the best possible recommendations for their customers?

Requirements

For Part II of this homework assignment, I will be very "hands-off" and answer very few questions other than to act like a client. You must do the work, including the appropriate data integration and cleaning to get the results you want. And do not ignore basic, fundamental EDA and visualizations. What might be interesting to the client to help you and your client understand the data, general distributions of genres, ratings, etc. I strongly encourage you to use Google to your advantage. Can you identify any interesting ways to visualize these types of data? Frequent patterns? What about visualizing association rules?

IMPORTANT - Your final notebook should convey a report to the client, but graded by your professor! Thus, you need a solution with CLEAN code and with good markdown cells throughout explaining what you are doing, and then interpreting your results throughout the process. All cells that process your data files from start to finish should be included. And, comment your code if it's extensive. Functions should be properly documented with docstrings! PLEASE INCLUDE ALL CODE YOU USE TO DERIVE YOUR ANSWERS. NOTE: If I have to search around for your answers, and you are not clearly outputting your results, plots, etc. you will lose credit!

I do not have any hard requirements. As I said, this is "hands-off". I encourage you to have some fun with this data, and see what strong patterns you can identify.

Deliverables

Commit both your **hw01.ipynb** and **hw01.pdf** files and push them to the remote Git repository. Be sure you have every cell run, and output generated before you commit. All plots should have an appropriate command to show the plot (unless you are using Plotly and the plot is too large to push to Gitlab. DOUBLE CHECK GITLAB AFTER YOU PUSH! I only grade what I can view!