PROJECT 3 README

NAMES AND UNI

Matthew Labasan, mjl2278

Phoebe Tang, ft2619

FILES SUBMITTED

1. proj3.tar.gz
    a. /proj3
        - main.py
        - INTEGRATED-DATASET.csv
2. example-run.txt
3. Project 3 README.pdf

GETTING STARTED

Prerequisites

1. *Python 3*

Installation

Creating VM (credit from class website and project description)

1. Click “CREATE INSTANCE” to create your instance
2. Type any name (for example, “cs6111-instance”)
3. Change “Region” to “us-east1 (South Carolina)”
4. Change “Zone” to any “us-east1-x” (the last letter doesn’t matter) or, if you are far from New York right now, to a zone near you
5. In the "Machine configuration" tile set "Series" to "N1" and "Machine type" to "**n1-standard-4 (4 vCPU, 15 GB memory)**."
6. In the "OS and storage" tile, click on "Change" and then go to "Boot disk"; then, in the new popup menu, select "Ubuntu" as "Operating system" and **Ubuntu 22.04 LTS (x86/64, amd64 jammy image built on 2025-02-28)** as "Version."; finally, change the boot disk size to 25 GB in the "Size (GB)" field and click the “SELECT” button
7. Now, you should be back on the “Create an Instance” page; scroll all the way down to the bottom of the page, and click the "Create" button; wait until the green checkmark shows up.

In the Google VM, after extracting the .tar.gz file, follow these steps:

1. Install Python 3 or newer.

   ```
   sudo apt update
   sudo apt install python3
   ```

Usage

1. Run & replace with your parameters, using a query in quotations:

   ```
   python3 main.py <dataset_path> <support> <confidence>
   ```

   a. **<dataset_path>** is the path to the desired csv file
   b. **<support>** minimum support for association rules - decimal between 0 and 1
   c. **<confidence>** minimum confidence for association rules - decimal between 0 and 1

   Ex : `python3 main.py ./INTEGRATED-DATASET.csv 0.1 0.5`

DESCRIPTION OF PROJECT

Dataset Decision

a. We decided to use the NYPD Shooting Incident Data (Historic) to generate our INTEGRATED-DATASET.csv file.
b. We did not make any modifications or combine any other tables, and the dataset was downloaded directly from the website without any query modifications. It was only renamed to 'INTEGRATED-DATASET.csv'.
c. Using this dataset, we are able to obtain association rules that reveal relationships between perpetrator demographics and victim demographics, location and victim demographics, and more. These more informative association rules tend to appear at lower support thresholds, as not all rows will include relevant information for these rules.

   These association rules can be valuable in identifying which demographics and areas within the city are more affected by shootings. This information could help focus resources on education initiatives about gun violence or guide decisions around enforcing stricter gun control measures in targeted areas. Additionally, it can inform outreach efforts toward communities that may benefit most from preventative resources. However, it is important to approach these findings thoughtfully and avoid using them to further stigmatize already marginalized populations.

Internal Design

We decided to implement the Apriori algorithm described in section 2.1 of the paper by Agrawal and Srikant. We also added the query optimization step as mentioned in lecture 10.

1. `main()`:
    a. We parse the command-line inputs into the correct data types. We open the CSV file and parse it row by row. For each row, we create a transaction by pairing each non-empty field value with its corresponding column name (formatted as `COLUMN=VALUE`) and store the items in a sorted tuple. These transactions are collected into a list for further processing. We check that `min_sup` and `min_conf` values are valid between 0 and 1.
        i. Note: Since we are using the column names included in the NYC Data datasets in the creation of our transactions, the algorithm will not output the result for data.csv correctly. So, if testing against data.csv, turn line 138 ‘`datacsv_check = False`’ to `True` to create transactions without columns.
    b. After preparing the transactions, we call the `apriori()` function to find all frequent itemsets that satisfy the minimum support.
    c. We then call `get_association_rules()` to generate all high-confidence association rules from the frequent itemsets based on the minimum confidence.
    d. Finally, we call `print_results()` to output both the frequent itemsets and the strong association rules in a human-readable format.
2. `apriori()`:
    a. This function is called once from `main()` after transactions are prepared. It counts the occurrence of each individual item across all transactions to find frequent 1-itemsets that meet the minimum support threshold.
    b. It then iteratively builds larger frequent itemsets by combining smaller ones, making sure at each step that all (k-1)-sized subsets of a candidate exist (pruning step). We also added the query optimization step mentioned in lecture 10. We check if the first k-2 items of both queries are equal and if the first k-1th item is less than the second k-1th item to make sure there is no repetition. We chose k-2 as the upper bound so the query optimization step is less strict, and we can extract more rules.

   c. At the end, our algorithm normalizes the support values by dividing each count by the total number of transactions and returns a dictionary of frequent itemsets with their normalized support. The output is passed to `get_association_rules()` and `print_results()`.
3. `get_association_rules()`:
   a. This function is called from `main()` after the frequent itemsets are generated by `apriori()`. It takes the frequent itemsets and attempts every possible non-empty split into left-hand side (antecedent) and right-hand side (consequent).
   b. For each potential rule, it calculates the confidence as (support of full itemset) / (support of left side).
   c. If the confidence exceeds the minimum threshold, the rule is stored. This function returns a list of valid rules, which are then printed by `print_results()`.
4. **Print_results**:
   a. This function is called at the end of `main()`, after both frequent itemsets and association rules have been computed.
   b. It prints the frequent itemsets, sorted by descending support, and the high-confidence association rules, sorted by descending confidence.

Command Line Specification

1. `python3 main.py ./INTEGRATED-DATASET.csv 0.1 0.5`

   As mentioned in our *Dataset Decision* section, we chose this dataset to identify which demographics and areas within the city are more affected by shootings, which could help focus resources on education initiatives about gun violence or guide decisions around enforcing stricter gun control measures in targeted areas.

   In this sample run, we set the minimum support to 10% and the minimum confidence to 50%. The resulting frequent itemsets and rules revealed some strong non-trivial patterns about NYC crime incidents.

   - There was a high prevalence of male victims: over 90% of the transactions involved a male victim, and the rules strongly predicted the victim is male. This indicates that gender plays a very significant role in the nature of these cases.

- We also saw that `VIC_SEX=M` and `STATISTICAL_MURDER_FLAG=false` often lead to `JURISDICTION_CODE=0` with 84% and 83% confidence, respectively. This reveals that both gender and nature of crime are strong indicators of the administrative handling of cases (in this case handled by NYPD).
- We were also able to extract victim information in specific NYC Boroughs. We saw that in Queens, Brooklyn, and Manhattan, the `VIC_SEX=M`, with confidence around 90% for each. This reveals the gender distribution across the city is skewed towards men, making them the primary target of shootings.
- Along the same lines, we found that for various perpetrator demographics, like `PERP_RACE=BLACK,` and `PERP_AGE_GROUP=25-44` also resulted in `VIC_SEX=M` with confidences at 89-91%.
- Below are some sample results:
  - [('PERP_AGE_GROUP=18-24', 'PERP_RACE=BLACK')]=>[('VIC_SEX=M',)] (Conf: 89.11376224968045%, Supp: 14.063340505648197%)
  - [('PERP_AGE_GROUP=25-44',)]=>[('VIC_SEX=M',)] (Conf: 88.45789971617786%, Supp: 18.860946745562128%)
  - [('BORO=BROOKLYN',)]=>[('VIC_SEX=M',)] (Conf: 90.17543859649125%, Supp: 35.425632060247445%)
  - [('BORO=MANHATTAN',)]=>[('VIC_SEX=M',)] (Conf: 89.99245662559719%, Supp: 12.032678859601937%)
  - [('BORO=QUEENS',)]=>[('VIC_SEX=M',)] (Conf: 89.60686850429282%, Supp: 13.333781603012373%)

Additional Information

We decided to include the column name when saving the results in the dictionary. This decision was to help us with interpreting the results since some of the data values are hard to interpret - ex: some values are just `False`, or `0`. So it's extremely helpful to understand what column the value is referring to: `'STATISTICAL_MURDER_FLAG=false','JURISDICTION_CODE=0'`. So, we are assuming that the first row of the data will be the column names. We also assume that the same value texts from different columns will be treated differently. For example, for the following data:

- pen,ink,diary,soap
- pen,ink,diary
- pen,diary
- pen,ink,soap

Even though the diary is in 3 transactions, they are in different columns (columns 2 and 3). Our code will process them as different values. Our justification for this is that in our data set, the same values in different columns have very different meanings. For example: `age=24 vs num_victims=24`. In this case, even though the value is the same across different columns, their meanings are very different.

Another important note is that with our low values for confidence and support, we will ultimately output many association rules that do not have much meaning–even rules with higher confidences may not be as meaningful. So, we sorted through each of the rules and chose the ones that gave us new insights into the data that were not previously trivial. Thanks to the Apriori algorithm, we can find these insights and come to meaningful conclusions that we can use to inform our solutions to various problems.