**Data Mining Portfolio Entry for Association (Frank Liu)**

**Data Acquisition**

A heads up

* There are 6 datasets that I used in this data portfolio, including 5 provided by professor Santago, and 1 is my own dataset

**Data provided by instructor:**

“Assoc\_Analysis\_Vidhya.dat.csv”

* This dataset contains 315 transactions that each transaction listed in a row.

“Table\_5\_1.txt”

* This dataset is a simple transaction dataset that only contains 5 transactions, which means it is easy to start with debugging and then generalize it to other bigger datasets.

“GroceryStoreStacked.dat.csv”

* This dataset is a stacked dataset where each translation has a number associated with it but is not in a row. There are about 7000 items being transacted, but only 1000 recorded transactions.

“GroceryStoreStacked\_sub.dat.csv”

* This dataset is also a stacked dataset, which contains 3483 items being transacted but only 1000 recorded transactions.

“House\_vote\_84.dat”

* This indicates the party of a person (democrat or republican) based on several features. The features are very similar to one-hot encoding. This means I have to process the data differently. It is difficult to deal with since it has a missing value. I will discuss how I handle missing value in the package use section.

**My own data**

“Groceries.csv”

* This dataset is referenced from <https://www.kaggle.com/irfanasrullah/groceries>
* The dataset is also uploaded to /data folder in the association analysis folder.
* This dataset has a similar format to all previous four datasets, each row containing a transaction that includes the item being purchased. However, the dimension of this dataset is significantly larger, containing 9835 transactions. This is **useful to analyze the time complexity** of the apriori algorithm.

**Usage of dataset**

* I use ALL 6 datasets in my code development and package use section of the code in order to do a thorough analysis of ideas around association analysis.

**Code Development / Program Development**

This section of the DMP contains the analysis of the **Apriori algorithm implementation**. I will also be working on extending the apriori algorithm by adding a **rule generation section**. This extra section will be done soon.

I will use Python in Google Colab to develop the code. Especially, I use two very helpful data structures: dictionary and set:

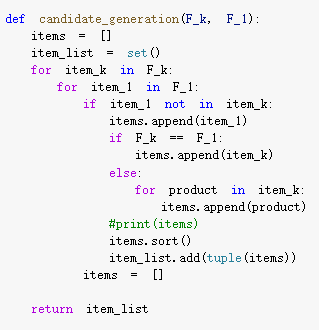
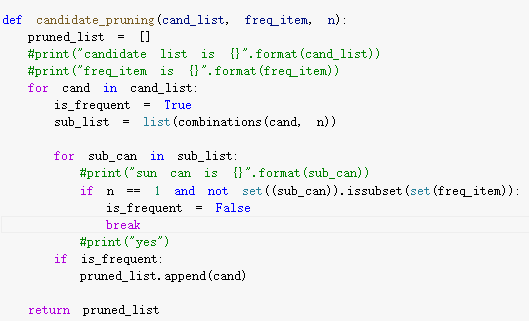
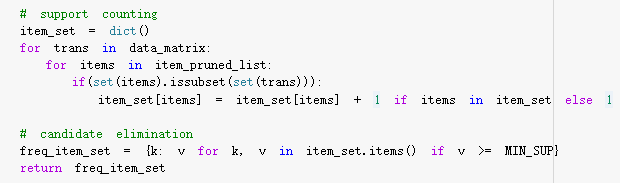
* **Sets** are able to help me to eliminate redundant items, this will be **useful in candidate generation** where I can eliminate candidates that are the same, especially in candidate generation.
* **Dictionary** is essentially the same data structure as HashMap in Java, where I can keep track of a unique set of keys, and use value as the counter (support count).
* More details on how to use them will be explained later.

**Apriori Algorithm:**

**Code link:**

<https://colab.research.google.com/drive/1nu6z_nZ8owC12l96oHLoirNx92Ff6RbZ#scrollTo=fnd8wz-LBC8j>

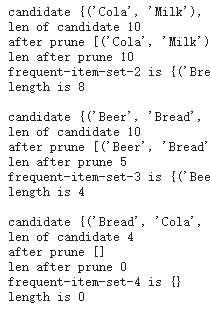
**Important points to note (principle of operation):**

* The first point to note is that the **data should be input and processed differently with regular row-ish data and with stacked data**. What I did is I wrote an if-else statement to identify if the data is row-ish or stacked. I process regular data using the normal method that was used in the previous code assignments, but in stacked data, I create a temp array to store all transactions with the same id number and append to the data matrix until the id number changes. This way, I can keep the input data matrix to the program the same to have the same analysis
* **Candidate generation:**
  + I initially tried to generate candidates by only using a frequent 1-item-set because it is easy to start with. After making the programs run, I realized that the **time complexity skyrocketed** since mathematically, combinations are in factorial level complexity, and it is extremely inefficient to use this method.
  + I later switch to **generating L\_k+1 using Fk and F1**, which is much faster. But this is very hard to implement, where I created several loops and spend a lot of time debugging. I think this is the hardest part of the program.
  + I found an **interesting trick to prevent generating redundant candidates** like {A, B} and {B, A} by first **sorting them**, and then putting them into a set.
  + Here is a snip of my candidate generation:
  + 
* **Candidate pruning:**
  + The way I did candidate pruning is to create an exhaustive sublist of candidates with size k-1 and see if all sublists are part of the frequent itemset k-1. This might not be the optimal one, but I can do it correctly.
  + Here is a snip of my candidate pruning
  + 
* **Support counting and candidate elimination**
  + This part is pretty simple by only having several dictionary function call.
  + The dictionary is structured with a key equal to itemset and the value is the counter.
  + The logic is to append itemset into the dictionary if the itemset is not in the dictionary, otherwise, we increment the counter
  + 

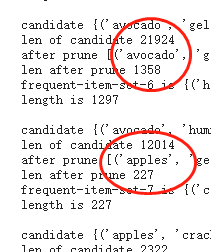
**Analysis of the results:**

I feel like this is very much like a data structure coding program that I have been practicing for interviews. It is important to have flexible use of different functions of sets and dictionaries. I need to familiarize myself with more function calls of sets and dictionaries.

Attached is the console output for the “Table\_5\_1.txt” data, and I omit the full printout of the itemset but only focus on the length and number. This result matches the output from the instructor.



For more complicated datasets like the stacked groceries store, the most frequent itemset has a k of 8. I found **an interesting phenomenon** that pruning at a higher k value (greater than 5) will cut around 90 ~ 95% of the candidate, meaning that candidate generation from F\_k-1 and F\_1 will result in a lot of infrequent candidates. This might be resolved by using F\_k-1 \* F\_k-1, which I will mark as a To-do.



I have also printed the results into an output text file where I uploaded in the drive for the Vidhya dataset. The link is attached here:

<https://drive.google.com/file/d/1ozdCwCDpS2TMyIhMil9jdKfFIT2XxbjT/view?usp=sharing>

Finally, I applied this implementation to my data set which has roughly 10000 transactions. The process is **extremely slow**, and python is not able to output the result but “the IOPub data rate is exceeded”.

This leads to the discussion of the **limitations of the apriori algorithm**:

In my own data, there are 13695 frequent 1-itemset. It needs to generate more than 10^7 candidates 2-item sets to be calculated and tested. Furthermore, the pruning algorithm essentially has to make sure all subsets of k-1 belong to the frequent k-1 item-set, and generating all subsets essentially costs factorial time (use combination concept in statistics). Therefore, apriori will be very inefficient when memory capacity is limited and the number of transactions is huge.

One way to speed up the process is to increase the min\_sup to decrease the size of frequent itemsets so Fk+1 will have a lower n. Also, more optimal coding algorithms can also help.

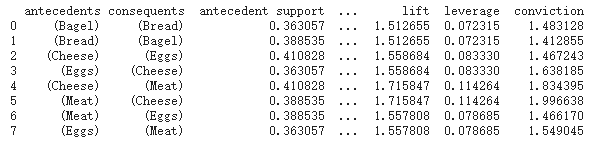
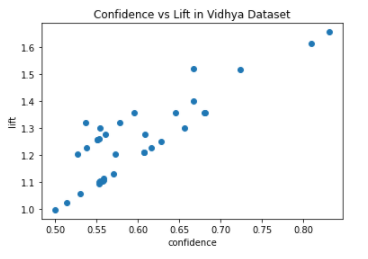
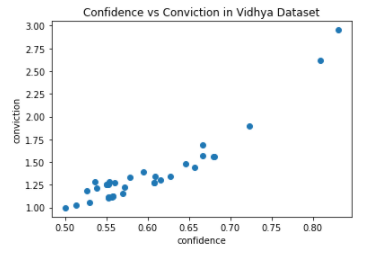
**Package Usage (mlxtend)**

Code link: <https://colab.research.google.com/drive/1Dh8I7jaes5Cb3o--HnctFfjeGGk4hsEO>

In the package use section, I use the mlxtend package in python to explore different functionalities to four different dataset

* Vidhya dataset
* 2 grocery store dataset
* House vote dataset
* My own data

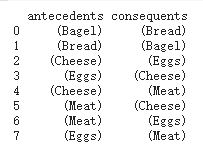
**Procedures**

* The very first step is to **input the data** in, but this step should be done **differently** with a different dataset
  + For Vidhya and my own dataset, I follow the normal way of importing them as data\_matrix
  + For the grocery store dataset, I will need to make sure items with the same id number appear in the same row of the data\_matrix
  + For housing vote data, I will need to encode the one-hot code (y,n) so every row only includes the feature that has y in it. I will also need to handle missing value
    - **Missing value handle will be explained in the result section**
* I **generate frequent itemse**t using either apriori or fpgrwoth method. I did **hyperparameters tuning** to min\_sup to make sure it shows at least 10 frequent itemsets.
  + Code I use is
    - 
* Then I **generate the association rule** using the association rule function
  + The hyperparameter that I used is confidence, I also tune the min\_threshold.
  + 
  + 
  + When looking at the table (example in Vidhya dataset, we can see all possible other criteria other than confidence follow a **similar pattern** to confidence, so I hypothesize using different criteria won’t matter in the smaller datasets. I will graph the **confidence vs lift scatter plot** and **confidence vs conviction plot** to see if they follow the same trend.
    - I decrease the min\_conf to show more data and plot a graph to show the trend.
    - 
    - Correlation coefficient = 0.86087091
    - 
    - correlation coefficient = 0.93233611
    - This confirms my hypothesis that in the smaller dataset, confidence works the same with other criteria (positively correlated). So I will stick with confidence as a criterion to generate association rules.
* We can have other operations such as **pruning the association rules or using selection criteria** to show rules only have certain antecedents or only have certain consequents or to eliminate some rules we do not want to see
* I will use this to **explore the house voting data** to see what features will associate with only have “republican” or “democrat” as consequents since this is an outstanding feature comparing to others.

**Results for different datasets**

* For all the results that I present, I tune the min\_sup to have it show up at least 10 frequent itemsets, and I tune min\_conf to show at least 5 rules. But in some datasets, there are only 2 rules (my own data) no matter how low the min\_conf is.

**Vadhya: min\_sup = 0,2. min\_conf = 0.2**



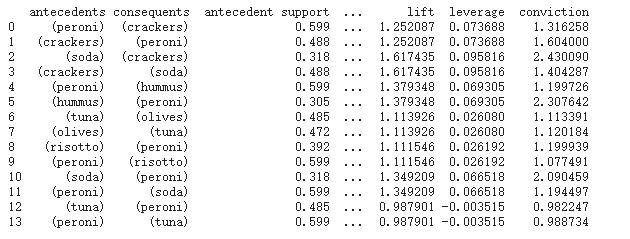
We can see that meat is often bought with eggs and cheese, and bread and bagel are often bought together.

**My\_own\_data: min\_sup = 0.05, min\_conf = 0.01**



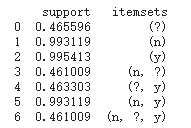
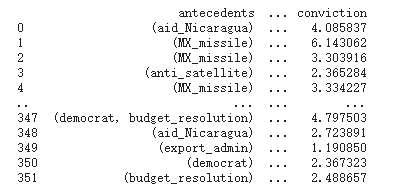
Since the dimensions of this data is so large, and set comparatively small min\_sup and min\_conf, but only get two association rules. We can see milk is often bought with vegetables. The **reason why there are only two of such rules** may be due to the very detailed classification of this dataset. In general, you have a soft drink as a category, but this one further separates into soda, coke, etc. The same rules happen to other categories. So this makes the frequent itemset smaller, which further limits the possible association rules.

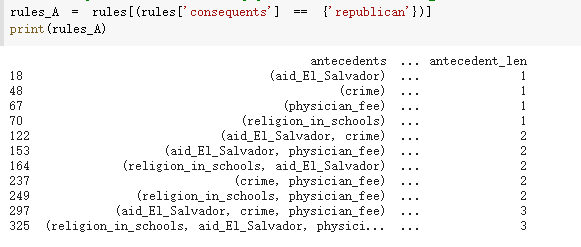
**Grocery stacked: min\_sup = 0.25, min\_conf = 0.4**



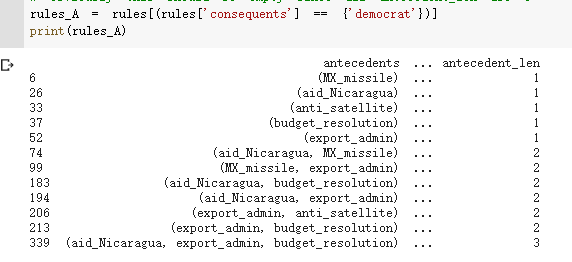
We can see tuna, Peroni is often associated with each other. So do hummus and Peroni, olives and tuna, etc. Thus this is helpful for businesses to do marketing strategy design so associated products can sell together or have discounts.

**Housing data:**

* If we do not handle missing data and just do not encode it to make it concise, we can see “?” will become a frequent itemset.
* 
* There are **lots of ways to handle missing values**, one is to delete the data that contains a missing value, or replace the missing value with the majority, or predict the missing value.
* Trying to predict the missing value needs to find similar data points. After manually looking at the dataset, I found that the missing data is created using previous data points by replacing some value with “?”, but in reality, the dataset might not be so clean. Finding similar points requires looping through the entire dataset and comparing each, and then matching individual elements inside the data point, which has a complexity of O(N^2). Given the high complexity of the missing value handle, I will just go ahead and delete the data with missing value this time.
* After handling missing value and correctly encoding, using **min\_sup = 0.5, min\_conf = 0.5**, we have:
  + 
* To only include democrat or republican in consequence



**We can see aid\_EI\_Salvador and physician\_fee are strong indicators of “republican”**



**We can see aid\_Nicaragua, budget\_resolution, export\_admin are strong indicators of “democrat”.**

The result is really interesting to know and use for a political campaign if the team recruits data-mining students :)

Looking away from rule generation, frequent itemset itself contains useful information that is some of the most popular products. I am a double major in CS and Business, so I am more sensitive to the application of CS into the business world. The association analysis of transactions to discover consumer behavior patterns can especially be analyzed with different seasons. Consumer preferences are influenced by seasonal change, data of the time, weather, location (learned from BEM221, Principle of Marketing). This is especially important in grocery stores. There are research shows that people tend to buy food at night rather than early morning. Weather conditions may influence mood, which further facilitates buying unhealthy food (which is delicious). Thus, we can also take a look into this issue from the frequent itemset perspective if we have data associated with time or season.

**Chapter 5 Exercise Theory**

[Chapter 5 Exercise Question](https://docs.google.com/document/d/16RMwE8wwfquj0mIUu17NejcmdKPdnRXo1v3PDv_FfaI/edit)

**Chapter Summary and Major Learning Points**

The attached link is the reading notes I take on ppt, reading, and in the lecture.

* [Chapter 5 Reading and Lecture Notes](https://docs.google.com/document/d/1o1y42cVO3PqKK1pz0TgDDPT3d609PxORCwFJo7CRMDE/edit)

This is a bit wordy so I will include some of the major learning points that I have

* Structure of market basket transactions and how to process them in different forms
* The usefulness of association analysis in the business world
* The concept and how to calculate support, confidence, lift, etc., and frequent itemset
* Apriori principle and Apriori algorithm, and how the algorithm is implemented using a dictionary and set
* Two major tasks of association analysis
* Different ways of generating frequent-itemsets and pros and cons of each of them (Fk-1 \* F1, Fk-1 \* Fk-1, F1\*F1)
* Pruning techniques
* Python package that can be helpful in association analysis

**Learning Summary and Self-Assessment:**

I found this chapter particularly interesting because other than CS, I also major in Business, so I focus a lot of attention on consumer preference, consumer behavior. In marketing, the most important concept is to build a mutually beneficial relationship with customers. Knowing their behavior and habits is very important, and this utilizes association analysis. It can powerfully discover the hidden relationships in a complicated transaction set.

Association analysis can also be applied to the biology field to diagnose diseases, protein sequence, and all other areas. It is powerful, and I feel excited to learn it by implementing algorithms.

I still have one confusion that given I have implemented the Apriori algorithm and reading some google articles, I don’t think the apriori algorithm is efficient in the high dimensional datasets. Are there any better ideas around association analysis that are powerful to process big datasets?

Overall, I think I had a decent understanding of the material and gained lots of insights in doing the coding. The implementation of Apriori is especially difficult given I have to walk through different functions available for set and dictionary to design a structure that can help me to do specific tasks. But I am happy that I can implement it. I think I have an A to A- comfort level in association after finishing DMP. But I lean toward A because I think I was able to crack the algorithm and find some interesting conclusions.