Jason Lebov | CS 470 - HW #2

Apriori Algorithm *[MinSupport Threshold @ 650]* → Output *[Descending Order]*

flu (26186)
shot (23167)
flu shot (23150)
get (7647)
flu get (7642)
got (7396)
flu got (7388)
shot got (6963)
get shot (6959)
flu shot got (6956)
flu get shot (6955)
getting (3087)
flu getting (3086)
getting shot (2764)
flu getting shot (2763)
today (2729)
flu today (2726)
shots (2573)
flu shots (2573)
today shot (2365)
flu today shot (2362)
arm (2023)
flu arm (2022)
flu like (1940)
like (1940)
flu shot arm (1900)
shot arm (1900)
sick (1842)
flu sick (1841)
flu like shot (1692)
shot like (1692)
sick shot (1685)
flu sick shot (1684)
year (1534)
flu year (1530)
time (1404)
flu time (1403)
shot year (1402)
flu shot year (1398)
today got (1265)
flu today got (1264)
time shot (1253)
flu time shot (1252)
today shot got (1196)
flu today shot got (1195)
feel (1018)
flu feel (1018)
free (1013)
flu free (1013)
still (990)
flu still (989)
work (968)
flu work (968)
amp (959)
amp flu (959)
feel shot (918)
flu feel shot (918)
never flu (917)
never (917)
one (915)
flu one (915)
get got (892)
flu get got (890)
flu still shot (886)
still shot (886)
get shot got (839)
flu get shot got (838)
never flu shot (833)
never shot (833)
sick get (818)
flu sick get (818)
flu need (814)
need (814)
one shot (801)
flu one shot (801)
work shot (799)
flu work shot (799)
arm got (789)
flu arm got (789)
amp flu shot (773)
amp shot (773)
flu going (770)
going (770)
shot arm got (764)
flu shot arm got (764)
flu day (759)
day (759)
sick get shot (750)
flu sick get shot (750)
flu shot need (739)
shot need (739)
get year (697)
flu get year (696)
gotten (695)
flu gotten (694)
flu going shot (688)
going shot (688)
flu sick got (680)
sick got (680)
sore (675)
flu sore (675)
shot day (667)
flu shot day (667)
gotten shot (665)
flu gotten shot (664)
last (664)
flu like got (664)
like got (664)
last flu (663)
get today (656)
flu get today (655)

*Continued on Next Page

Report:

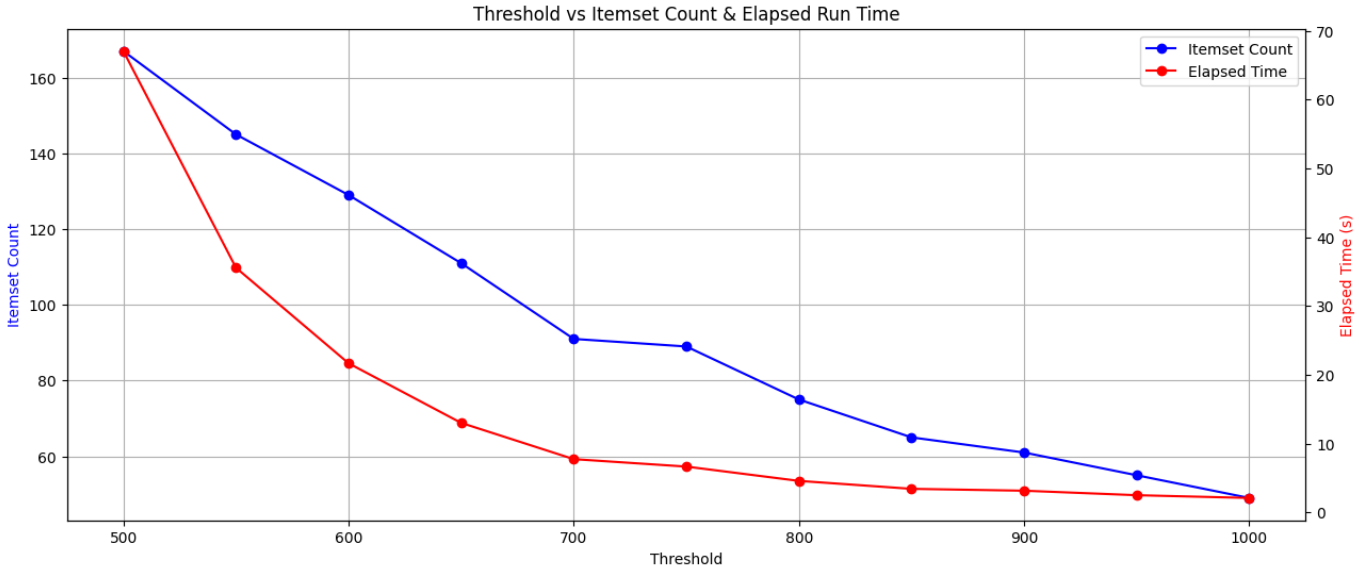
After running my implementation of the Apriori Algorithm on the 'data.csv' dataset (2014 Flu Shot Twitter Dataset), with a minimum **support threshold of 650**, the above *(page up)* is the resulting output. We can see that the term '*flu'* was the most frequent with **26186** occurrences and '*flu get today*' was the least with **655**.

Based on these results we can determine that there is a pattern in the types of words that were used at the time on Twitter when discussing the flu. Many of these words seem to have to do with the flu shot, and/or getting the flu virus. Most of the words that are seen mainly include the word '*flu*' but also include groupings such as '*flu got*' and '*flu shot*', along with other words that fit into this context such as '*sick*' and '*arm*'. All these words seem to relate to a person who had gotten the virus and/or has gotten the flu shot. Also, most of these words appear to be action words indicating a common discussion on Twitter regarding flu-related actions or experiences. Additionally, many of the words revolve around time such as '*today*' and '*day*' which leads me to believe these tweets occurred during, or around flu season. Lastly, many of the words relate to symptoms of the flu such as '*sore*' and '*sick*'. Overall, all these words and word grouping are flu-related, and I would assume that they were posted during flu season to describe symptoms of the virus or the act of getting the vaccine.

For my implementation, I chose to use a **minimum support threshold** of **650** because it enabled me to maximize the number of itemsets found in the data, while also being considerate of the time the method took to execute.

*Continued on Next Page

As seen by this graph:



After running my Apriori implementation for thresholds ranging from 500 *(the minimum threshold outlined in the assignment instructions)* to 1000 and plotting both the Itemset Count (Blue Line) and Elapsed Time (Red Line) vs each Threshold, I was able to analyze the impact of each. A threshold of 500 took the longest time, about a minute, to execute and returned around 165 itemsets. Although this is good, I felt it took a bit too long to run and therefore I wanted to increase the minimum support value to help. After analyzing the graph, I determined that a value around 650 would be optimal given it took about 15 seconds to run and still outputted about 110 itemsets. This allowed me to ensure I was still getting enough frequent itemsets to be able to find insight and gain knowledge, while also ensuring time would not be an issue.

When coding my algorithm, I made sure to utilize a few different measures to help speed up my runtime. Many of these measures included different Python, mainly Python pandas functions/operations that allowed me to work with the data optimally. Additionally, I tried to use the least number of for-loops as I felt was possible to avoid iterating through every sample in the data.

A major lesson I learned from this assignment was to plan and think through your implementation before beginning the coding process. For this assignment, I spent about a day thinking through how I wanted to implement this algorithm and what I felt that would look like in code. By doing this step before coding, I was able to see any areas where I might struggle and think through different fixes. This made the coding and debugging step much more efficient as I already knew exactly what I wanted to accomplish, and how I was going to accomplish it. Additionally, I learned many new Python operations to help with my implementation such as the *Combinations* Function along with many others.