

SEG304
Data Webpage Analysis with Apriori Algorithm
(wapa)

Student Name: Mark Tamer

Student Number: 6411572

<https://github.com/mtamer/wapa>

What I Implemented:

Wapa standing for “Webpage Analysis with Apriori Algorithm” is an application built in Python that allows users to enter search term(s), and using that term(s) the application crawls the Internet and retrieves the top 10 newest webpages in relation to that search term(s). We are also able to change the number of pages we want to visit. Meaning we can substitute the top 10 pages for the top 50 pages and so on. Hence one third of the application is actually building the WebCrawler itself. Once we have grabbed those website links, we visit the webpage and retrieve all the data on that page. After this I have implemented a set of regular expressions that parse all the “junk” out and grab only what’s important. Meaning we only grab the information located inside the paragraphs. Once this is done we split up all the words in the paragraphs leaving them as individual words. Therefore another one-third of this application is building the parser. Now with all this data we need to make sense of it. So we put all that valuable information per page (article) in it’s own dataset. Now in this example we have 10 datasets, and with each dataset we grab the top 100 distinct words. We do this in order to cut the search time and complexity of what happens next, which is the implementation of the Apriori Algorithm. Finally I have implemented the Apriori Algorithm to help make sense of all this data. It verifies what words coexist in the same dataset given a provided frequency, and then returns the results and displays them to the user.

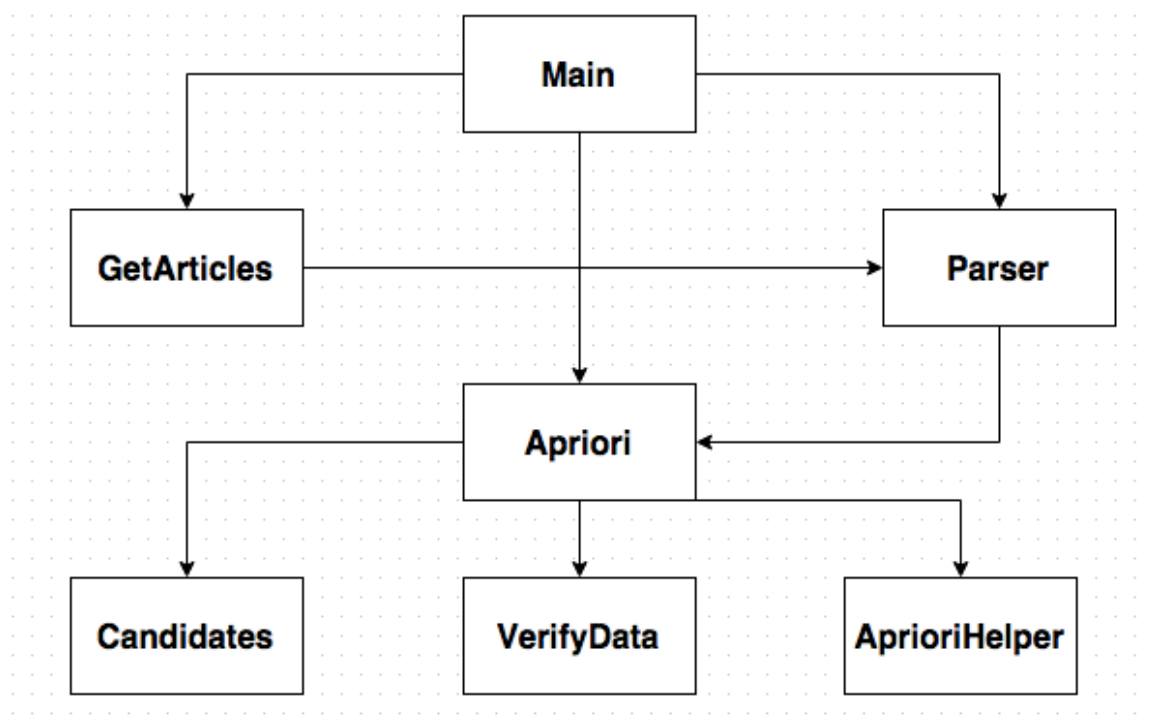
Why did I choose to Implement it:

The purpose of this project was for me to learn to program in a different language other than JAVA, but primarily learn about data mining algorithms and

patterns, and with this knowledge build a real world application. I implemented this application to primarily learn about how certain words affect those companies in terms of stock market price. I have a huge interest on how social media plays a role in the depreciation and appreciation a companies share price. For example when running our application and we enter the company “Target” as our search term and if the word “scandal” showed up more often in multiple datasets, we can than look at the stock market and see how this companies stock price is doing in terms of all the negative media results returned. This will also be the same for positive words.

Algorithm and Approach:

Diagram Explanation:



Main() Method Pseudo Code:

*User enter there search term (keyword) here.
With this keyword we call the method **getArticles()**.
When we get the articles we pass them through the our **parser()** method.*

Then we pass the data retrieved into our **apriori()** method.
Return our results.

getArticles() Method Pseudo code:

Visit Appropriate Query + **keyword**
Open the Query and read what's on the page
Remove all the data we don't need from the page
For all List tags in our page
 find all anchor tags in them
 Append all anchor tags in an Array
Return Array

parser() Pseudo Code:

Take in the result array returned by the **getArticles()** method.
Create Array to store the dataset
Create a dictionary of stopwords
Open all the URLs
Visit the URLs and store the webpage data
For specific data in the webpage data:
 Filter out the appropriate words
 Split the words
 Take Top 100 words provided
Store each article in it's own dataset
Return result array, which contains all these datasets

candidates() Method Pseudo Code:

Take in the dataset provided by the **parser()** method.
Initialize Array
For All items in our dataset
 Create list of candidate item sets of size one
Return Dictionary dataset

verifyData() Method Pseudo Code:

Create a dictionary
For all items in the dataset:
 Compare each item to another
 Check which Item meets the minimum support
Store items in a minimum support dictionary
Include the item and the frequency we see it in the dictionary
Return the data.

aprioriHelper() method Pseudo Code:

*Create an array
Calculate the number of frequent sets given the appropriate min support
Store the words in the array
Sort the array
Remove duplicate words*

apriori() method Pseudo Code:

*Create a list of terms using **candidates()** method each itemset is of length k
Verify data is correct using **verify()** method and that each item set is frequent
Use **aprioriHelper** method to reduce searching process and optimization
Keep frequent itemsets to create itemsets of length $k+1$
Return the results.*

What was Learned:

Throughout the implementation of my project I've learnt that everything around us, all this data can be really important to individuals who are seeking to make sense of it all. I have discovered how to build/implement/understand Algorithms as complex as the one used in this project "Apriori Algorithm" and the different methodologies to write code. Some technologies that I've learned would be the Python programming language and its various libraries and packages. I've learnt that Python is a beautiful language for quick and effective development and that it's an excellent language for automation, web apps, games, etc..

I've also learnt a lot about natural language processing and how closely it relates to data mining and machine learning. I have also learnt to implement a dictionary of stop words used in the application. The purpose of this is to disregard words we find have no meaning. I have learnt about frequent pattern matching and text matching which are also used in the application. Frequent pattern matching

being: the number of occurrences we see something show up in our results. Text matching being: statistical relevant patterns between datasets, which both methods are used in the application. I have also learned to build a very general and versatile WebCrawler that can be used for any terms we want. I have built a real world application and posted it online where others have begun to notice it and use it. I learnt design patterns and regular expression all while building the logic (backend) to the application. Finally I understand that testing and patience is key to success and that data mining is massive field with infinitely many possibilities. One thing that will always be with me while implementing this project is that research and testing is a must when working with so much data, especially when most data we retrieve is useless.

If I had more time what would I do:

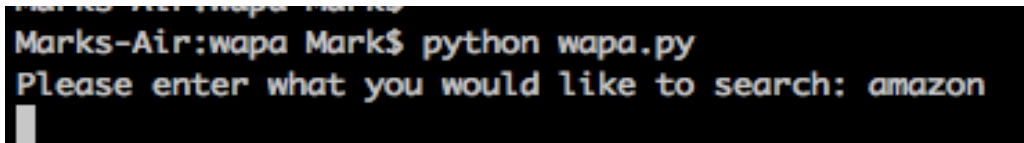
If I had more time I would have implemented a Graphical User Interface (GUI) that allows users to enter in their search term. From there I would have returned their results and displayed it on a graph in order to help the user to have a better understanding of the information and to provide a visual aspect to it. I would have also liked to do a time series analysis on the data being generated, as a different form of trying to understand the data presented.

What would I have done differently?

If I had to redo this project again I probably would have written it in a more hardware friendly language. The language would have been in GO, because I am able to boot up multiple distributive streams to be able to process more words more quickly using the Apriori Algorithm, resulting in a much quicker search.

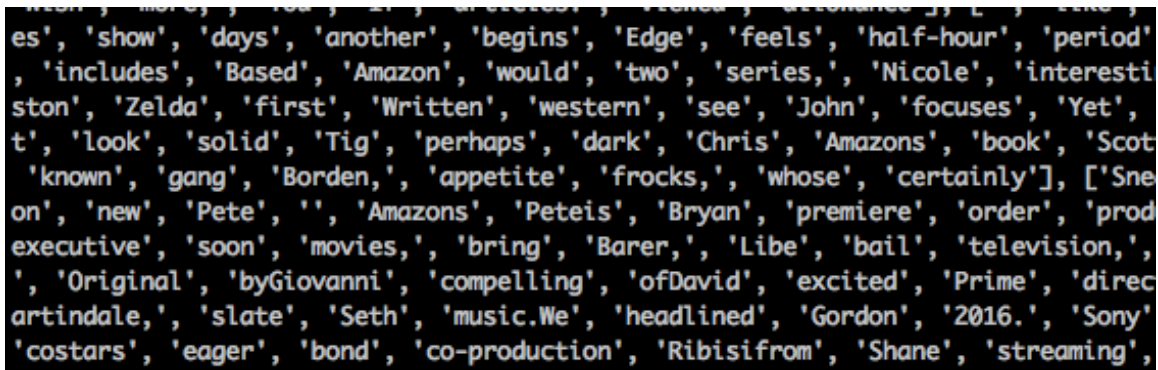
Application Testing:

I have tested the application out many times and it works for every term(s) being entered for our search key. The only problem is trying to look up terms in which the article(s) have lots of pictures, because this requires a lot of time to load the article and will cost us a lot when filtering them out as well.



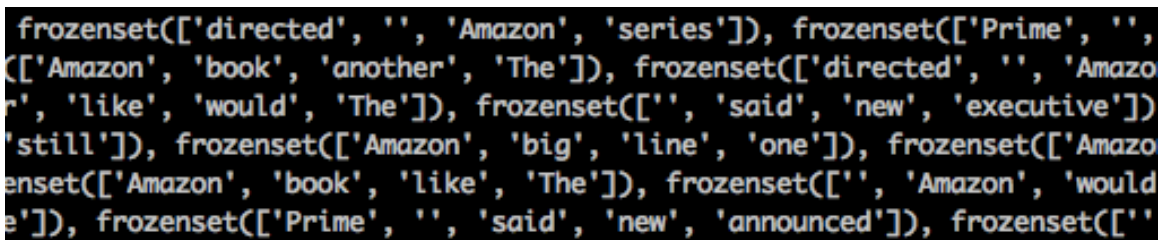
```
Marks-Air:wapa Mark$ python wapa.py
Please enter what you would like to search: amazon
```

Figure 1: Entering Search Term



```
es', 'show', 'days', 'another', 'begins', 'Edge', 'feels', 'half-hour', 'period',
, 'includes', 'Based', 'Amazon', 'would', 'two', 'series', 'Nicole', 'interesti
ston', 'Zelda', 'first', 'Written', 'western', 'see', 'John', 'focuses', 'Yet',
t', 'look', 'solid', 'Tig', 'perhaps', 'dark', 'Chris', 'Amazons', 'book', 'Scot
'known', 'gang', 'Borden', 'appetite', 'frocks', 'whose', 'certainly'], ['Sne
on', 'new', 'Pete', 'Amazons', 'Peteis', 'Bryan', 'premiere', 'order', 'prod
executive', 'soon', 'movies', 'bring', 'Barer', 'Libe', 'bail', 'television',
, 'Original', 'byGiovanni', 'compelling', 'ofDavid', 'excited', 'Prime', 'direc
artindale', 'slate', 'Seth', 'music.We', 'headlined', 'Gordon', '2016.', 'Sony'
'costars', 'eager', 'bond', 'co-production', 'Ribisifrom', 'Shane', 'streaming',
```

Figure 2: Top 100 words per Website



```
frozenset(['directed', 'Amazon', 'series']), frozenset(['Prime', 'Amazon', 'book', 'another', 'The']),
frozenset(['directed', 'Amazon', 'like', 'would', 'The']), frozenset(['Amazon', 'big', 'line', 'one']),
frozenset(['Amazon', 'book', 'like', 'The']), frozenset(['Prime', 'Amazon', 'would', 'announced']),
```

Figure 3: Apriori Algorithm

Technology/Methods used:

- Python
- Natural Language Processing Python (<http://www.nltk.org/>)
- Beautiful Soup (parsing HTML and XML):
(<http://www.crummy.com/software/BeautifulSoup/bs4/doc/>)
- Regex (Regular Expressions)

- Numpy (<http://www.numpy.org/>)
- Apriori Algorithm (https://en.wikipedia.org/wiki/Apriori_algorithm)
- Sublime (<http://www.sublimetext.com/>)
- Git (<https://git-scm.com/>)
- Frequent Pattern Matching
(https://en.wikipedia.org/wiki/Pattern_matching)
- Text Mining (https://en.wikipedia.org/wiki/Text_mining)
- Clustering (https://en.wikipedia.org/wiki/Cluster_analysis)