Project #1 report

Part 1: The Algorithms

In this project, I have written **four** algorithms for frequent pattern mining. I have written **Apriori algorithm**, **two imporved Apriori algorithms** and **FP-growth algorithm**. All the source codes are in attachments. All of the script can get the input of minimum support and the scale of data set and then output the frequent pattern based on such minimum support and the runtime.

Let me introduce the two imporvied Apriori algorithms.

The first one (Apriori_improved.py) splits the original dataset into several parts (in my script, I split it into 7 parts). And then using the some minimum support percentage do the Apriori algorithm in each part. The frequent patterns which appeared in most part are the frequent pattern itemsets.

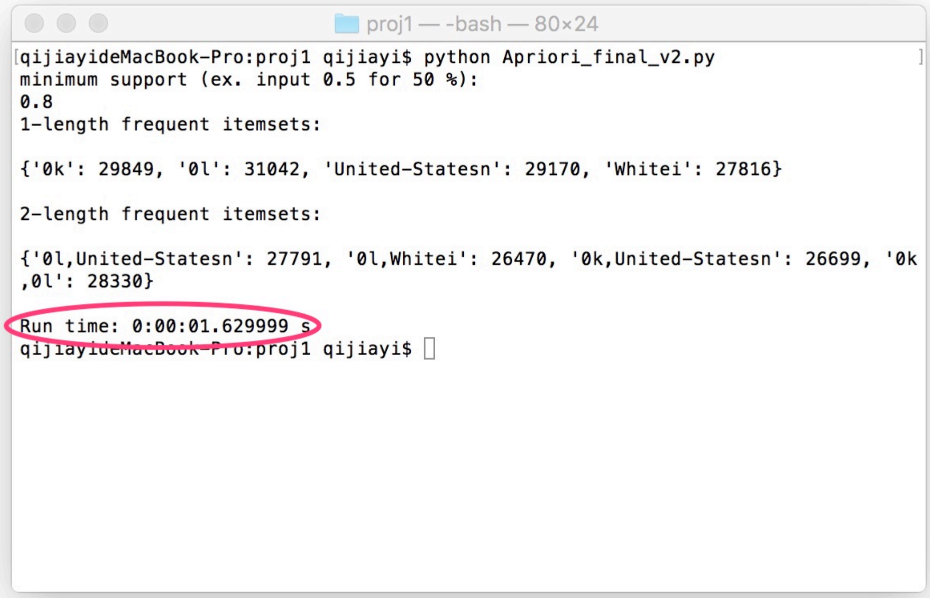
The second one (Apriori_improved_v2.py) using the 1-length frequent itemsets to pruning the original dataset (not the original data file). And then keep doing the Apriori algorithm.

The codes in attachments:

1. **Apriori_final_v2.py**
2. **Apriori_improved.py**
3. **Apriori_improved_v2.py**
4. **FP-growth_v4.py**

Part 2: Runtime analyzing for different minimum suport

In every algorithm, I using the python module **datetime** to figure out the total runtime by getting the time at the begin and the end.

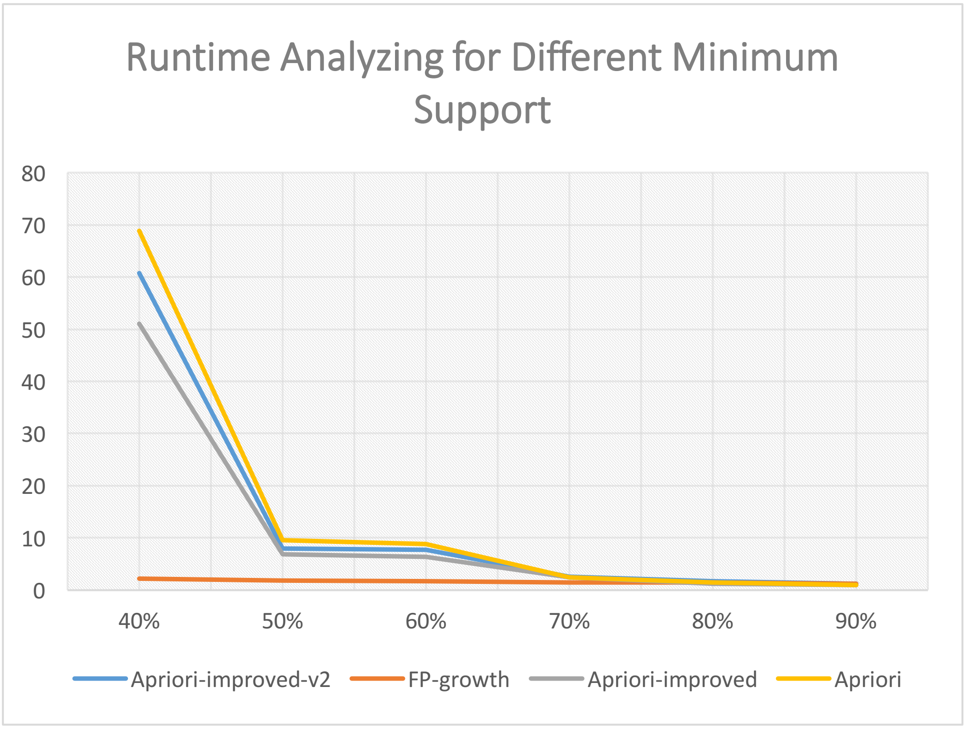


```
qijiayideMacBook-Pro:proj1 qijiayi$ python Apriori_final_v2.py
minimum support (ex. input 0.5 for 50 %):
0.8
1-length frequent itemsets:

{'0k': 29849, '0l': 31042, 'United-Statesn': 29170, 'Whitei': 27816}

2-length frequent itemsets:

{'0l,United-Statesn': 27791, '0l,Whitei': 26470, '0k,United-Statesn': 26699, '0k
,0l': 28330}

Run time: 0:00:01.629999 s
qijiayideMacBook-Pro:proj1 qijiayi$
```

I have run the scripts with different minimum supports (from 40% to 90%) and recorded the runtime in a table. You can see it below:

	40%	50%	60%	70%	80%	90%
Apriori	68.8s	9.5s	8.8s	2.4s	1.5s	0.9s
FP-growth	2.2s	1.8s	1.7s	1.5s	1.4s	1.2s
Improved Apriori	51.1s	6.9s	6.4s	2.4s	1.2s	0.9s
Improved Apriori v2	60.8s	8s	7.7s	2.5s	1.7s	1.2s

And here is the broken line graph according to the table above:



From the table and graph above, we can find out several conclusions.

A. FP-growth has the best and steady runtime among these four algorithm.
B. In this dataset, when the minimum support is greater than or equal to 70%, all of the four algorithm have almost the same runtime.
C. In this dataset, FP-growth has a better runtime than other three algorithms when the minimum support is under 70%.
D. In this dataset, Apriori algorithms, including the two improved algorithms, have a much longer runtime when the minimum support is lower than 50%.
E. The first improved algorithm has a better runtime than the second one. Both of the improved algorithms have a better runtime than the original Apriori algorithm.

To conclude, in this dataset, all of the four algorithms have the almost same runtime when minimum support is greater than or equal to 70%. And FP-growth have a steady and relatively short runtime with all the support threshold. As for the two improved algorithms, the first one is a better way to improve Apriori, although it's still have a much longer runtime than FP-growth when minimum support is under 70%.

Part 3: Runtime analyzing for different data size

Because we only have one dataset, to figure out how data size influence each algorithm, I divide the data set into different size and then analyze the runtime.

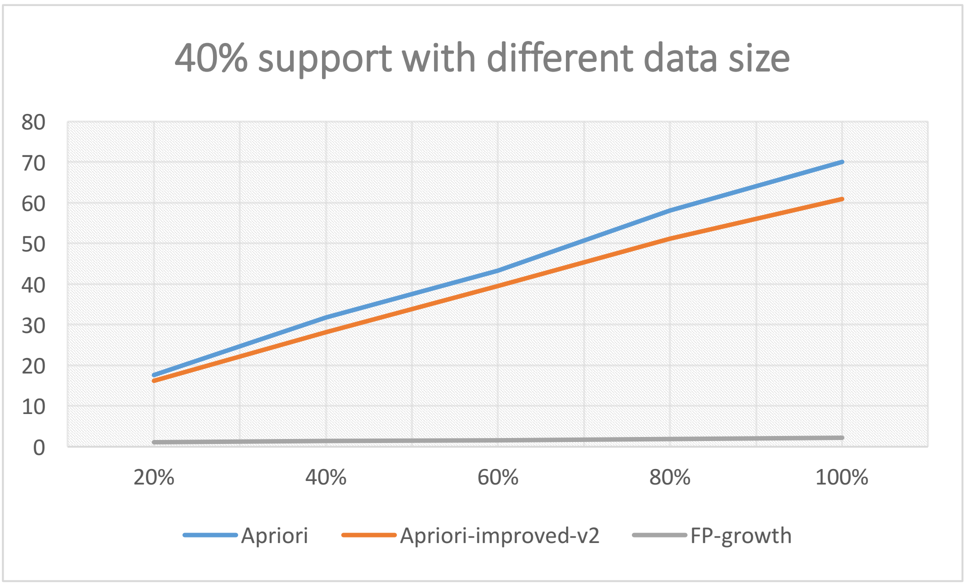
I just compare the performance with different data size for Apriori, the second improved Apriori and FP-growth. Because the result of first one (the one doing apriori in different subset of data set) would be different and doesn't make any sense.

I test the different size of data set (20%, 40%, 60%, 80% and 100% of the data set).

For 40%:

	20%	40%	60%	80%	100%
Apriori	17.6s	31.8s	43.2s	58.0s	70.0s
Apriori-improved-v2	16.1s	28.1s	39.5s	51.1s	60.9s
FP-growth	1.0s	1.3s	1.5s	1.8s	2.1s

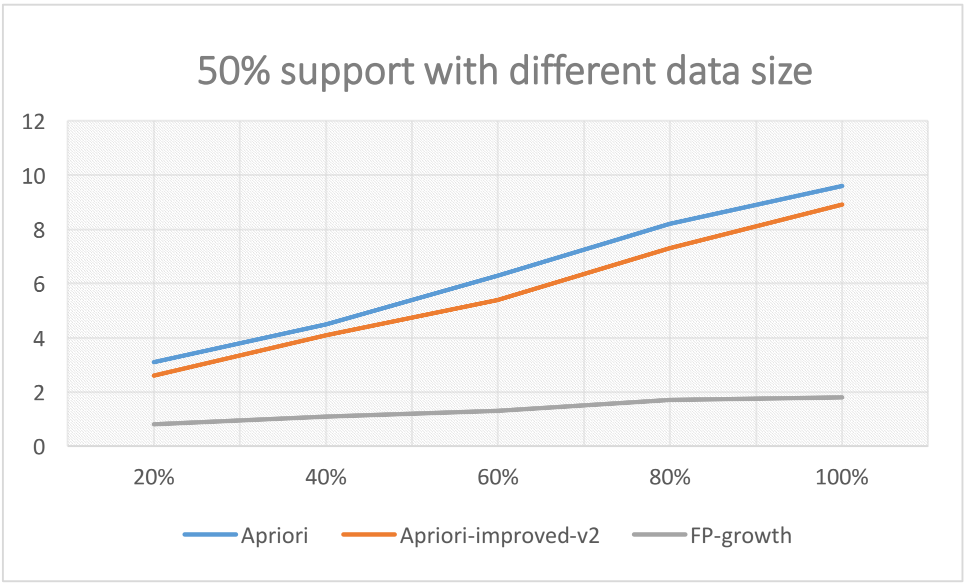
The broken line graph:



For 50% support:

	20%	40%	60%	80%	100%
Apriori	3.1s	4.5s	6.3s	8.2s	9.6s
Apriori-improved-v2	2.6s	4.1s	5.4s	7.3s	8.9s
FP-growth	0.8s	1.1s	1.3s	1.7s	1.8s

The broken line graph:



I didn't do the same things with higher support since the graph are very similar with the two above.

From these two graph, we can figure out that the runtime of Apriori-based algorithm will significantly increase with the increasing of data size while the runtime of FP-growth is almost steady with the increasing of data size.

The reason is obvious: Apriori-based algorithm will scan the data set lots of times so that the runtime will depend on the data size. But FP-growth algorithm only look through the data set two times.

Part 4: Conclusion and Further Work

Conclusion:

In this data set, FP-growth algorithm has a much better performance than the three Apriori-based algorithm. The data size and minimum support have little influences on the runtime of FP-growth algorithm. But data size and minimum support significantly influence the runtime of Apriori-based algorithms.

Further Work:

The two improved Apriori algorithms do have some improvements from the original one, but I have some ideas might make a big improvement. The idea is shown below:

When we find out a transaction contain a specific itemset, ***A*** (such as the 1-length frequent itemset), we will record the location of that transaction. And then whenever we are checking a joined itemset which contain ***A***, we can directly check the itemsets at such locations so that we don't need look through the whole data set anymore.

I think this would improve the Apriori to a large extent. But because of the time constraint (I have spent lots of time in FP-growth algorithm), I didn't write this algorithm into real program.

Also, there are plenty of other improving method, such as sampling the data set, which use accuracy to exchange a shorter runtime.