Homework 2

The version of Spark I use in the assignment is version 1.6.1

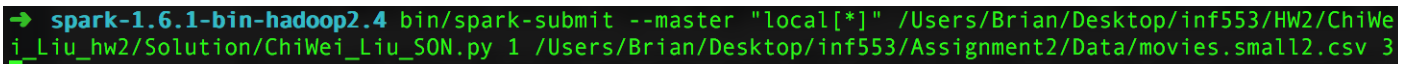
- ## Problem 1:

There will need three input argument to run the code, which are **Case Number**, **Input.csv** and **Support**. You should use you own file path for the input.csv and source code to run the code. For example:

1. Source code file path:
/Users/Brian/Desktop/inf553/HW2/ChiWei_Liu_hw2/Solution/ChiWei_Liu_SON.py
2. Input.csv file path:
/Users/Brian/Desktop/inf553/Assignment2/Data/movies.small2.csv
3. Also include --master "local[*]" before the source code file in the command line
4. You may switch the **case number** between the source code file path and Input.csv file path
5. You may switch the **support number** after the Input.csv file path
6. The output file will be generated under the folder of spark, and it will name as
ChiWei_Liu_SON_MovieLens.case1.txt if the running case is 1,
ChiWei_Liu_SON_MovieLens.case2.txt if the running case is 2.

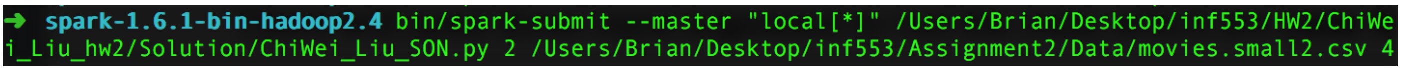
The command line will be exactly the same as following:

Case 1, Support 3:



```
➜  spark-1.6.1-bin-hadoop2.4 bin/spark-submit --master "local[*]" /Users/Brian/Desktop/inf553/HW2/ChiWe
i_Liu_hw2/Solution/ChiWei_Liu_SON.py 1 /Users/Brian/Desktop/inf553/Assignment2/Data/movies.small2.csv 3
```

Case 2, Support 4:



```
➜  spark-1.6.1-bin-hadoop2.4 bin/spark-submit --master "local[*]" /Users/Brian/Desktop/inf553/HW2/ChiWe
i_Liu_hw2/Solution/ChiWei_Liu_SON.py 2 /Users/Brian/Desktop/inf553/Assignment2/Data/movies.small2.csv 4
```

- ## Problem 2:

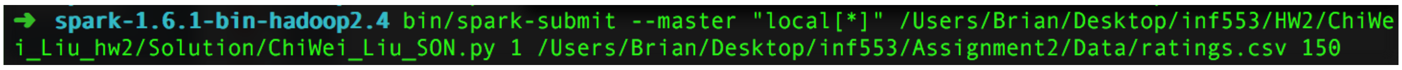
There will need three input argument to run the code, which are **Case Number**, **Input.csv** and **Support**. You should use you own file path for the input.csv and source code to run the code. For example:

1. Source code file path:
/Users/Brian/Desktop/inf553/HW2/ChiWei_Liu_hw2/Solution/ChiWei_Liu_SON.py
2. Input.csv file path:
/Users/Brian/Desktop/inf553/Assignment2/Data/ratings.csv
3. Also include --master "local[*]" before the source code file in the command line
4. You may switch the **case number** between the source code file path and Input.csv file path
5. You may switch the **support number** after the Input.csv file path

6. The output file will be generated under the folder of spark, and it will name as ChiWei_Liu_SON_MovieLens.case1.txt if the running case is 1, ChiWei_Liu_SON_MovieLens.case2.txt if the running case is 2.

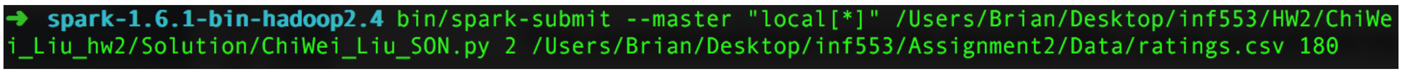
The command line will be exactly the same as following:

Case 1, Support 120:



```
➜  spark-1.6.1-bin-hadoop2.4 bin/spark-submit --master "local[*]" /Users/Brian/Desktop/inf553/HW2/ChiWei_Liu_hw2/Solution/ChiWei_Liu_SON.py 1 /Users/Brian/Desktop/inf553/Assignment2/Data/ratings.csv 150
```

Case 2, Support 180:



```
➜  spark-1.6.1-bin-hadoop2.4 bin/spark-submit --master "local[*]" /Users/Brian/Desktop/inf553/HW2/ChiWei_Liu_hw2/Solution/ChiWei_Liu_SON.py 2 /Users/Brian/Desktop/inf553/Assignment2/Data/ratings.csv 180
```

Execution Table:

CASE 1		CASE 2	
Support Threshold	Execution Time	Support Threshold	Execution Time
120	10.68 sec	180	309.19 sec
150	5.27 sec	200	184.56 sec

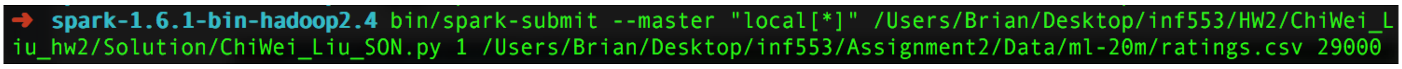
- ## Problem 3:

There will need three input argument to run the code, which are **Case Number**, **Input.csv** and **Support**. You should use you own file path for the input.csv and source code to run the code. For example:

1. Source code file path:
/Users/Brian/Desktop/inf553/HW2/ChiWei_Liu_hw2/Solution/ChiWei_Liu_SON.py
2. Input.csv file path:
/Users/Brian/Desktop/inf553/Assignment2/Data/ml-20m/ratings.csv
3. Also include --master "local[*]" before the source code file in the command line
4. You may switch the **case number** between the source code file path and Input.csv file path
5. You may switch the **support number** after the Input.csv file path
6. The output file will be generated under the folder of spark, and it will name as ChiWei_Liu_SON_MovieLens.case1.txt if the running case is 1, ChiWei_Liu_SON_MovieLens.case2.txt if the running case is 2.

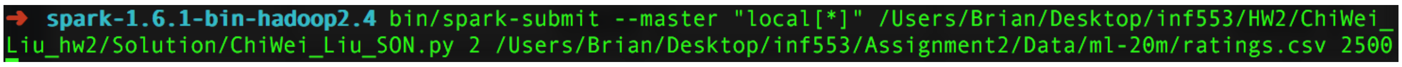
The command line will be exactly the same as following:

Case 1, Support 29000:



```
➜  spark-1.6.1-bin-hadoop2.4 bin/spark-submit --master "local[*]" /Users/Brian/Desktop/inf553/HW2/ChiWei_Liu_hw2/Solution/ChiWei_Liu_SON.py 1 /Users/Brian/Desktop/inf553/Assignment2/Data/ml-20m/ratings.csv 29000
```

Case 2, Support 2500:



```
➜  spark-1.6.1-bin-hadoop2.4 bin/spark-submit --master "local[*]" /Users/Brian/Desktop/inf553/HW2/ChiWei_
Liu_hw2/Solution/ChiWei_Liu_SON.py 2 /Users/Brian/Desktop/inf553/Assignment2/Data/ml-20m/ratings.csv 2500
```

Execution Table:

CASE 1		CASE 2	
Support Threshold	Execution Time	Support Threshold	Execution Time
29000	477.60 sec	2500	222.1 sec
30000	385.55 sec	3000	108.37 sec

Bonus: Describe why did we need to use such a large support threshold and where do you think there could be a bottleneck that could result in a slow execution for your implementation, if any. First of all, the benefit of using large support is to avoid finding too many trivial items that did not appears too many times; also, using large support can avoid slow execution time for the program to process. If the support is low, the program will have too many pairs and items needs to be compute, therefore it will take a lot of time to complete. In my program, the bottleneck of the support for case 1, large data set may be 5000 or lower, since the support is too low, the program needs very long time to process, which might be the bottleneck for user to complete in normal time. The bottleneck for case 2 may be 500 or lower.

In this project, I mainly use SON Algorithm and A-Priori to find the frequent item sets. In the SON Algorithm, I find the frequent singleton in the item sets, and use the frequent singleton to generate the combination of candidate pairs, and count the times that candidate appears in the baskets reach the support or not. Then use the result of frequent pairs to continue combine and generate the frequent triple… until we cannot generate any pair in the item set. Which means all frequent item are found in the item set.