**Assignment 1 – Description of my code**

1. **Task 1: common word count**

|  |  |
| --- | --- |
| Working directory | WordCount folder |
| Input | stopwords.txt, task1-input1.txt, task1-input2.txt |
| Final Output | output/step4 |

|  |  |  |  |
| --- | --- | --- | --- |
| Steps | Input | Functionality | Output |
| Step 1 | stopwords.txt  task1-input1.txt | TokenizerWCMapper reads in input1.txt, and decomposes the file to tokens and removes the stopwords in the tokens.  Input: file text  Output: <word, 1>  IntSumReducer sums the token count and outputs word count in input1.txt  Input: <word, 1>  Output: <word, count> | output/step1 |
| Step 2 | stopwords.txt  task1-input2.txt | TokenizerWCMapper reads in input2.txt, and decomposes the file to tokens and removes the stopwords in the tokens.  Input: file text  Output: <word, 1>  IntSumReducer sums the token count and outputs word count in input2.txt  Input: <word, 1>  Output: <word, count> | output/step2 |
| Step 3 | output/step1  output/step2 | Mapper1 reads word count from file 1  Mapper2 reads word count from file 2  Input: <word, count>  Output: <word, count>  CountCommonReducer takes the smaller one of the 2 counts from 2 files (if both files have the word). And then outputs common word count.  Input: <word, count>  Output: <word, common word count> | output/step3 |
| Step 4 | output/step3 | SortMapper reads the common word count from step 3 output.  Input: <word, common word count>  Output: <common word count, word>  ReverseComparator is implemented to sort the output from Mapper in descending order based on the count.  SortReducer writes the top 15 sorted output to the file.  Input: <sorted common word count, word>  Output: <top 15 common word count, word> | output/step4 |

1. **Task 2: Recommendation system**

|  |  |
| --- | --- |
| Working directory | RecommendationSystem folder |
| Input | data.csv |
| Final Output | recommend/step5 |

|  |  |  |  |
| --- | --- | --- | --- |
| Steps | Input | Functionality | Output |
| Step 1 | data.csv | Step1\_ToItemPreMapper reads in data.csv and processes it to key-value pair.  Input: csv data  Output: <userId, itemId:score>  Step1\_ToUserVectorReducer aggregates the scores together for a user.  Input: <userId, itemId:score>  Output: <userId, itemId:score, itemId: score …> | recommend/step1 |
| Step 2 | recommend/step1 | Step2\_UserVectorToCooccurrenceMapper generates item pairs which occur for the same user.  Input: <userid itemid:score, itemid:score>  Output: <itemid1\_itemid2, 1>  Step2\_UserVectorToConoccurrenceReducer adds up the frequency of cooccurrence for each item pair.  Input: <itemid1\_itemid2, 1>  Output: <itemid1\_itemid2, frequency> | recommend/step2 |
| Step 3 run 1 | recommend/step1 | Step31\_UserVectorSplitterMapper reads step 1 output and generates user score for all possible combinations of item pairs.  Input: <userId, itemId:score, itemId: score …>  Output: <itemid1\_itemid2, userId\_score> | recommend/step3\_1 |
| Step 3 run 2 |  | This step is removed from my code | recommend/step3\_2 |
| Step 4\_1 | recommend/step2  recommend/step3\_1 | Step4\_PartialMultiplyMapper reads frequency of item pairs from step 2 and reads user\_score of item pairs from step 3 run 1.  Input: <itemid1\_itemid2, frequency> and <itemid1\_itemid2, user\_score>  Output: <itemid1\_itemid2, frequency> and <itemid1\_itemid2, user\_score>  Step4\_AggregateReducer stores all user\_score values into a hashset and finds out the frequency. And use frequency to multiply with each score for each of the users in the hashset.  Input: <itemid1\_itemid2, frequency> and <itemid1\_itemid2, user\_score>  Output: <item1\_item2\_userId, mul> | recommend/step4\_1 |
| Step 4\_2 | recommend/step4\_1 | Step4\_RecommendMapper processes the item pair key to a single item so that they can be passed to the same reducer to be summed up.  Input: <item1\_item2\_userId, mul>  Output: <item2\_userId, mul>  Step4\_RecommendReducer sums partial multiplication results together for an item for a user.  Input: <item\_userId, mul>  Output: <item\_userId, score> | recommend/step4\_2 |
| Step 5 | recommend/step4\_2 | Step5\_FilterSortMapper filters items for user 62 because my student ID is E0134062.  Input: <item\_userId, score>  Output: <item, score>  Step5\_FilterSortReducer uses a hashmap to store item score pairs and sort them using SortHashMap.  Input: <item, score>  Output: <item, score> in score’s descending order | recommend/step5 |