NgramCount

To extend the program to count n-grams, we modified the Drive file to receive the input string “-n N” and transfer to integer for Mapper, and we also updated the Mapper file to use a two-level nested loop to create every combination of N-gram keys. There are 2 indices to retrieve N continuous words. The first index iteratively move from the begin of the tweet to N words front of the end of the tweet, and the second index move from the first index to the next N words to generate every N-gram word.

HashTagSim

Runtime: 3,059,544 ms = 50.99 min

Top 5 hash tag pairs and their similarity:
3515229 #jobs #photography
3104763 #fb #ff
2362069 #fb #smem
1835633 #fb #photography
1793309 #1 #fb

HashTagSim
Optimization 1 New Algorithm of Calculating Every Hashtag Pair Similarity

- **Technique:**
  The original HashtagSim program only computes similarities between #job and other hashtags. To extend the program to all pairs of hashtag, the intuition way is to run #job firstly and then go through all other hashtags. The runtime of computing #job and other hashtags is 3568 ms, however, the total number of hashtag is 40132, for this large number of hashtag, the runtime of computing similarities between hashtags would be up to 3568ms * 40132 = 39.78 hours, even though the average of hashtags is ⅕ of #job (because #job has lots of co-occured words compared to other hashtags), it would request 8 hours to complete.

  Our method for all hashtag pairs' similarity is to take non-hashtag words as keys, and co-occured hashtags as value. Then use MapReduce to sum up the counts of co-occured hashtags. Thus, we can get the similarity contribution of each word to every hashtag pair in the value set. So we "mapped" every hashtag pair and their product of co-occured counts, and then "reduced" to the sum of similarity from all words. It is also the problem can be solve by MapReduce.

  For example:
  #a b c
  #a b #b
  #b #c d e
  #c e f

  The original method:
  MapReduce of getting hashtags feature vector:
  #a b:2;c:1;
  #b b:1;d:1;e:1;
  #c d:1;e:2;f:1;

  Mapping of similarity:
  Similarity of #a #b: common words: b:2*1=1
  Similarity of #b #c: common words: d:1*1=1; e: 1*2=2, sum=3

Our method:
MapReduce of getting words feature vector:
b #a:2;#b:1;
c #a:1;
d #b:1;#c:1;
e #b:1;#c:2;
f #c:1;

Mapping of similarities of each word to each hashtag pair:
Map 1st line: similarity of #a #b: 2*1 (The part of similarity of #a #b from b)
Map 2nd line: no hashtag pair
Map 3rd line: similarity of #b #c: 1*1 (The part of similarity of #b #c from d)
Map 4th line: similarity of #b #c: 1*2 (The part of similarity of #b #c from e)
Map 5th line: no hashtag pair

Reduction of total similarities of each hashtag pair:
2 #a #b
3 #b #c

While scanning hashtag pair, it's important to aware the sequence of hashtag pair. For example, #a #b and #b #a should be grouped into one hashtag pair while computing similarity. Therefore, we arrange the hashtag pair in alphabetical order to prevent the repeated hashtag like #b #a.

HashTagSim
Optimization 2 Remove Empty Strings ““ in Value

- **Technique:**
  We observed the redundant word ‘;:1’ in the end of each value which is generated during getting word feature vector. The possible reason could be the extra empty string ““ while Mapper read file and split the word in the beginning. This redundant words could cause the incorrect result of similarity, so we remove this during computing similarity to make sure the correctness of result and save the computing throughput.

- What is the observed speed up?
  77 mins / 51 mins = 1.51x

HashTagSim
Optimization 3 Setup the Number of Map/Reduce Task

- **Technique:**
  While using AWS EMR to run hashtagsim, we find out that “Map Tasks Running” in WebUI would be varied, could be 1, 3 or 4. While “Map Tasks Running” is equal to 1, the run time would be up to 2.5 hours, on the other hand, if “Map Tasks Running” is 3 or 4, the run time would be reduced to 1.2 hours. The reason for this difference is the setting number of Map task. If we don’t setup the number of Map task, the system would automatically choose the number of Map task by situation. We adopted the default value of maximum number of “Map Task” of c1.medium, 2, and “Reduce Task”:1. For 4 nodes, that is use 8 and 4 for “Map Task” and “Reduce Task”. However, these setting is not forced for Hadoop, so we still need to split the input file to make Hadoop use multiple Map/Reduce tasks. Based on the input file size of GetHashFeatureVector and GetHashtagSimilarity which is tweets1m.txt (94MB) and tmp/feature_vector/part-r-00000 (17MB), we divided into 8 input file for 8 map tasks, each needed to be at least 11.75MB, 2.125MB. This setting can reduce the runtime to about 51 minutes. In addition, it’s helpful for Reducer to parallel compute 8 output files by setting ReduceJobs=8.

- What is the observed speed up?
  164 mins / 51 mins = 3.22x