Josef Klafka, Jon Pekarek, Timmy Li, Michelle Liang
CMSC 12300
Amazon Reviews Final Report

**Link to Github Repository:** https://github.com/mliang1825/CS123

**Link to Dataset:** http://jmcauley.ucsd.edu/data/amazon/links.html

**Access to Google Buckets:**
https://console.cloud.google.com/storage/browser/cmsc123amazonreviews

**Description of Data Set:**
Originally, the data set we found was 18 GB and contained approximately 83 million Amazon reviews across all product categories. The original format of the data set is a JSON file. Given that our group will be looking at all possible pairs of the review data, we opted to run all of our code on a sub-data set of 94.4 MB that contained all reviews within the category of baby products, which contains 153,919 reviews. Each review within the JSON file contained the reviewer ID, asin (ie. product ID), reviewer name, helpfulness rating, the review text, overall rating score, and the summary of the review.

**Hypotheses you tested:**

1. Is there a significant percentage of Amazon reviews that use similar words and phrases? The motivation behind this hypothesis is that there has recently been news articles that claim that a lot of companies are paying customers to buy their products on Amazon and post a positive review using a certain template the company has created. We wanted to test this on a subset of Amazon Reviews to see just how many reviews are similar to each other given different thresholds.
2. Are most similar reviews written for the same product? This hypothesis is a continuation of our motivation for the previous hypothesis. If a large percentage of similar reviews are written for the same products, then those products may be the ones where their companies are paying customers to leave positive reviews.
3. Are some similar reviews written for different products actually written by the same person? The motivation behind this hypothesis is that we wanted to see whether people who write reviews naturally use similar phrases and words to describe different products.

**Algorithms Used:**
We used a “reverse-top k” algorithm for obtaining the most common words in a file of reviews so that the words from the reviews could be filtered. In the main part of our project, we used a similarity matrix with ideas from natural language processing. We constructed a matrix, where

each row corresponded to a review and each column corresponded to a word in the vocabulary. Each entry then represented how many of that word (in that column) the review (in that row) had. This was a rudimentary similarity matrix that allowed us to directly compare the words used in different reviews and how many of that word was used.

**Big Data Approaches Used:**

For our project, we used MapReduce in several ways. First, we used MapReduce to find the least occurring 60,000 words in the dataset. These are the words that we will be looking at within each review, and the reason we used the least occurring 60,000 words is that we wanted to filter out the very common insignificant words that would be within reviews (eg. “the” and “a”). Also, we didn’t want to do comparisons with very common descriptive words (eg. “good” and “nice”). The second way we used MapReduce was to look at each review and get the counts of the number of occurrences of the 60,000 words within each review. We wrote the data about word counts for each review to a sparse matrix from this MapReduce work. The third way we used MapReduce is in the pairing step. We listed out the (column, value) tuples and iterated over the two rows in locked steps. Then we used Mapreduce to count the number of pairs with matching product IDs and reviewer IDs below a certain threshold.

**Runtimes:**

As for our extrapolated runtimes, obtaining the least occurring 60,000 vocabulary words in a sub data set, run using the maximum number of core instances (8) took 1.5-2 hours. In comparison, when we tried to run this code on the 18 GB data set, the code didn’t stop running after more than 24 hours. Creating the sparse matrix of review word counts took less than an hour, and took less than an hour when run on a local machine only as well. Though this step did not take very long, increasing the file size used in this step greatly increases the time needed for the next step where we would compare all possible pairs of reviews. Because we were thinking ahead for this pairing comparison step, we choose to stay with the file size we decided to go with. Our next step of doing the pairing comparisons in pairings.py used 20 nodes on a sparse matrix with about 150,000 lines and took about one hour. Threshold_counts.py used 15 nodes and ran on the output of pairings.py (a 4.6 GB file) and ran for about 2 hours.

**Big Data Related Tools Used, But Not Covered in Lecture:**

1. Google Cloud Buckets - we used buckets to store the large files needed for this task and be able to easily work with them using Google Cloud Platform.
2. We learned how to attach auxiliary files in MapReduce so that the auxiliary files would be sent to each node while the algorithm was carried out, namely through the use of the “configure_options” method in MRJob with “self.add_file_option(‘--auxiliary’)”.
3. Sparse Matrix - we used a sparse matrix in order to reduce the size in memory and on disk of the reviews word counts file. Because reviews will have much fewer than the

60,000 unique words, a sparse matrix is an efficient way to store the data as word counts that are zero will not take up space, unlike in a regular matrix.

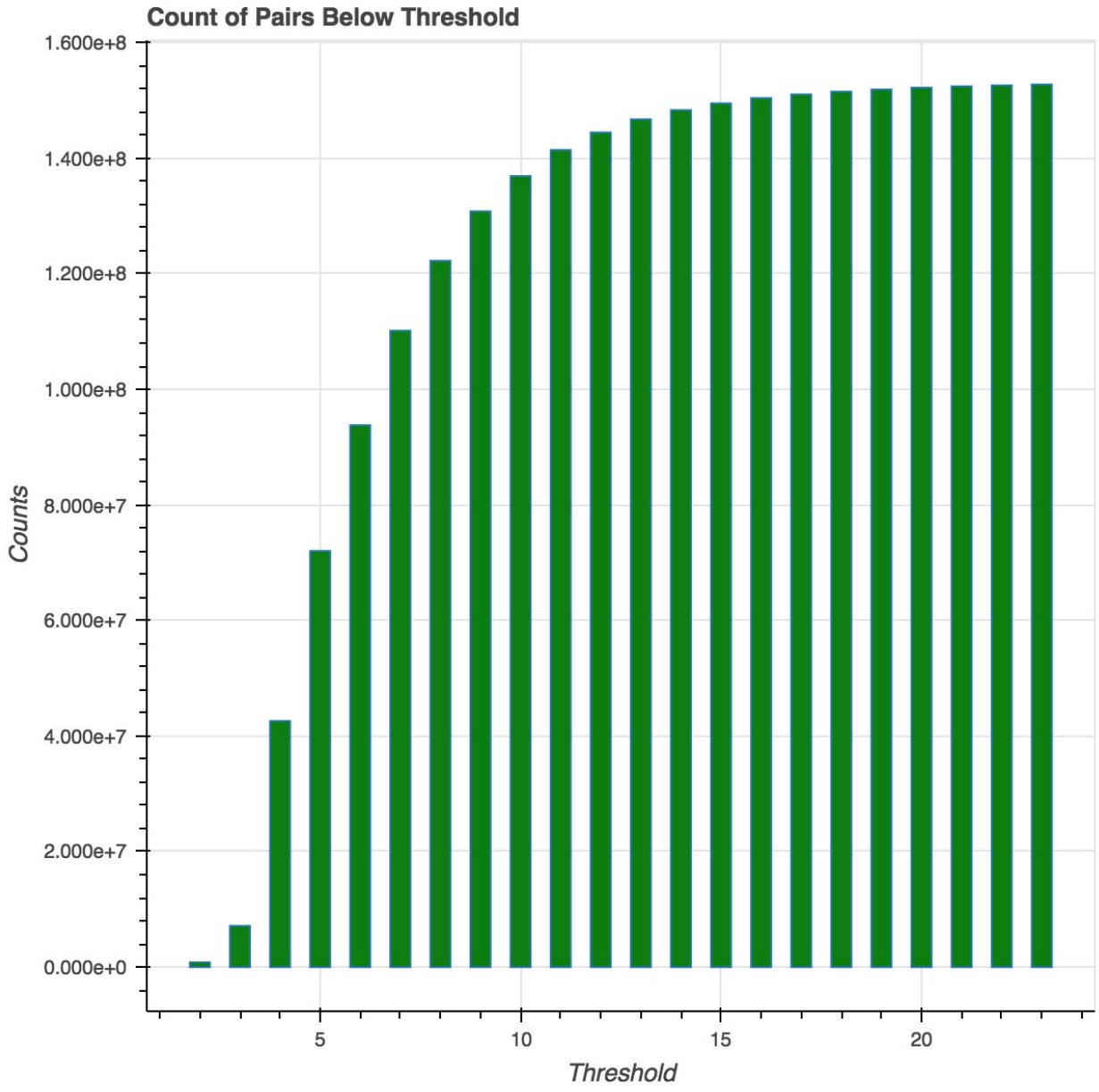
**Challenges Faced:**

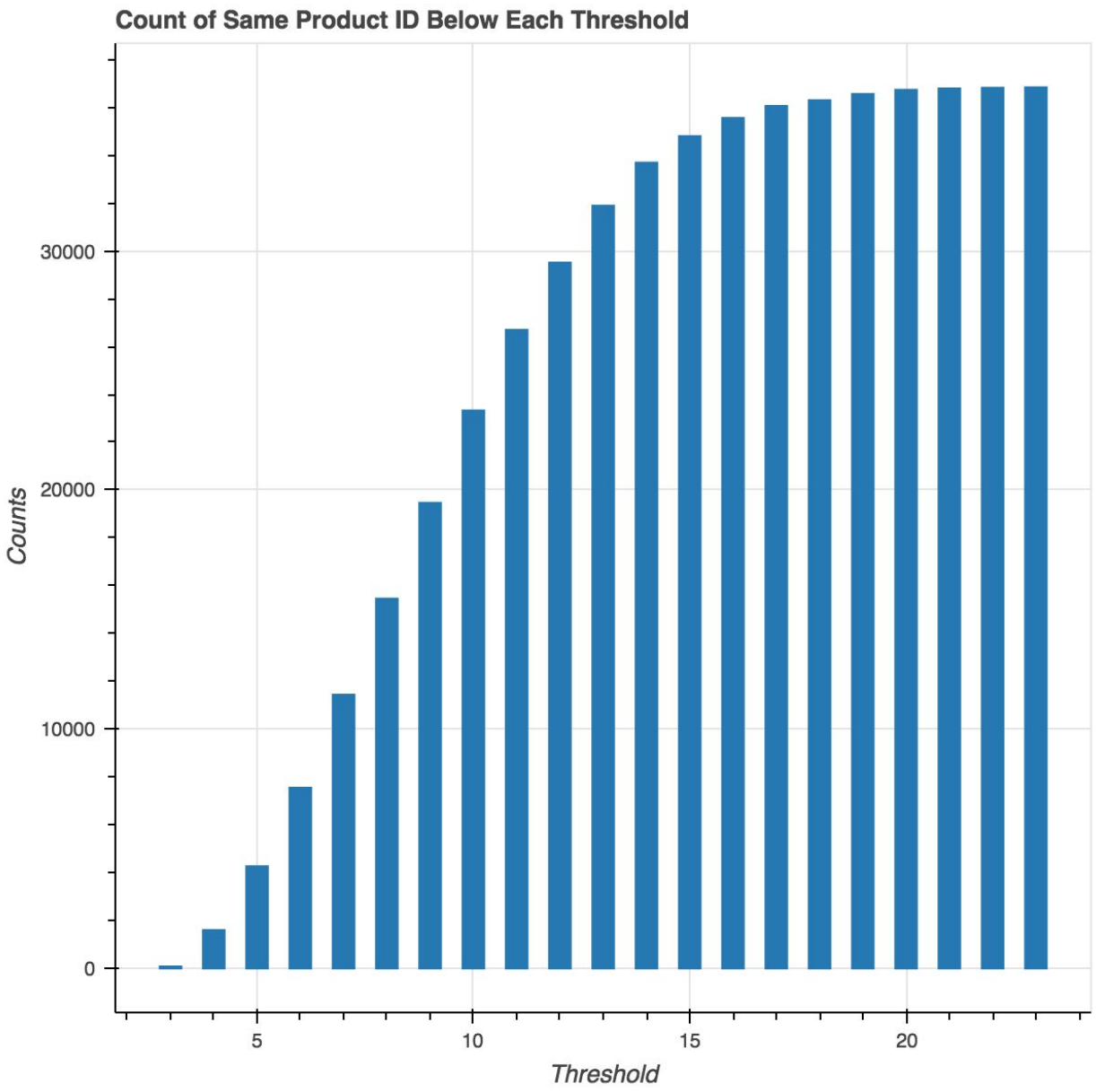
One challenge we faced was downloading and storing the 18 GB data set with the 83 million reviews. We quickly realized that the estimated downloading time for the data set on our laptops would take upwards of 48 hours, and our laptops also did not have enough storage to hold the data set. We overcame this challenge by using a Google Cloud bucket. With the bucket, we only had to upload the data set once and we were able to access the data set without downloading it again and again. Another challenge we faced was accessing the data set in the bucket when we wanted to run our code. There were a lot of problems we encountered when we were trying to share the bucket with all members of our team. Google Cloud has several ways to give other people access to buckets, and directly adding the gmail or username of another person's Google Cloud account did not work. However, we eventually figured out that we had to add a person's actual MapReduce account number in order for that person to gain access to the bucket, and the problem was resolved. The third challenge we faced was actually running our code on the 18 GB dataset with the 83 million Amazon reviews. Originally, we planned to use MapReduce to get the 50,000 least occurring words in the 18 GB dataset. However, the code wouldn't finish running even after letting it run for over 24 hours. Thinking ahead to our pairing comparison step where we would essentially have to look at 83 million * 83 million pairs of reviews, we decided to run our code on a sub data set instead. A challenge that we thought we would face but didn't end up facing was holding the sparse matrix in memory. Even though we were running the code on a smaller sub data set, we were afraid that complications would arise when we tried to store and read the sparse matrix. However, we didn't end up facing this problem, which we think is mainly because the sparse matrix only stores non-zero numbers. This helped us save a lot of space, since most of the reviews would have zero counts for most of the 60,000 words we looked at.

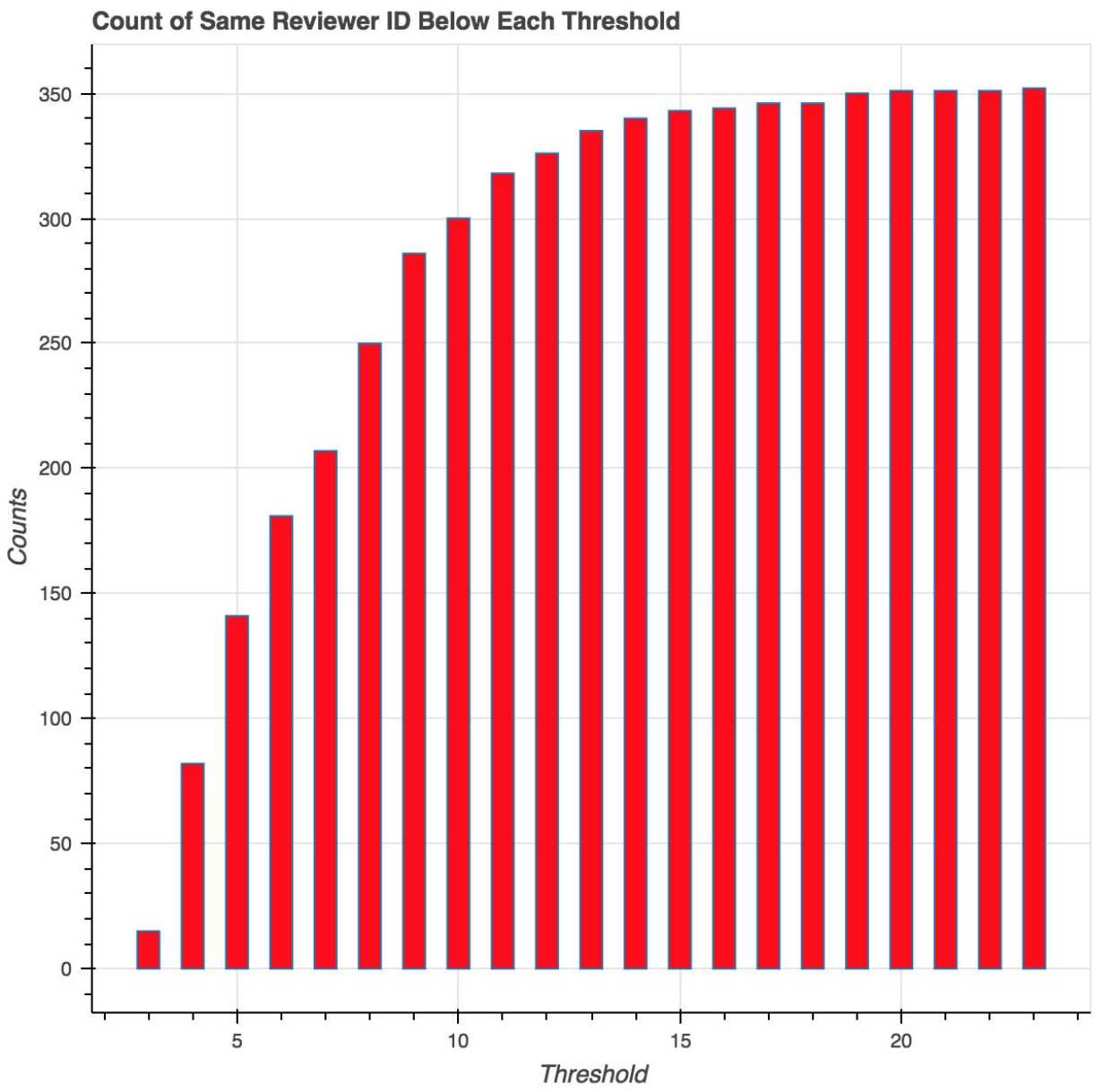
**Results:**

In the three bar graphs below, we show the final results that we saw in the pairing comparisons. For all three graphs, the x-axis represents the similarity thresholds. For each pair of reviews we looked at, we calculated a number that represents the distance between the two reviews. Intuitively, the closer the distance (ie. the smaller the number), the more similar the two reviews are. For all three graphs, we start at looking at the similarity threshold of 23, which is the average distance of all pairs of reviews. Then, we keep subtracting one from the current threshold to get all the other thresholds we are looking at (ie. 23, 22, 21, 20, etc). The y-axis of all three graphs represents the number of pairs found below a similarity threshold that meets a certain criteria. For the first green graph, we display the number of pairs that fall below each threshold. For example, we found approximately 70,000,000 pairs of reviews that have a distance of 5 or

below. For the second blue graph, we display the number of pairs that fall below each threshold, but also are pairs of reviews that have the same asin, or product ID. Thus, in other words, we display the number of pairs that are below each threshold and are reviews for the same product. For example, approximately 4,000 pairs of reviews are at a distance of 5 or lower and are for the same product. This relatively lower number of 4,000 in comparison to the 70,000,000 pairs of reviews at a distance of 5 or lower may mean that though there are companies that may pay people for fake reviews, this only occurs for a number of products, while most similar reviews may have a close distance because people in general tend to use the same words or phrases when leaving positive or negative reviews. Our third red graph displays the number of pairs that fall below each threshold and are written by the same person. For example, we see that approximately 140 pairs of reviews are at a distance of 5 or below and are written by the same person. Though our group originally suspected that the same person may use similar phrases or words when leaving reviews for different products, this may actually not be the case, as we see relatively low counts of pairs of similar reviews that are written by the same person. However, it should be noted that we only looked at reviews left for baby products, and if we were to expand our analysis to across more product categories, there may be more statistical proof surrounding our original intuition.


Count of Pairs Below Threshold
1.600e+8
1.400e+8
1.200e+8
1.000e+8
8.000e+7
6.000e+7
4.000e+7
2.000e+7
0.000e+0
Counts
5
10
15
20
Threshold


Count of Same Product ID Below Each Threshold
Counts
30000
20000
10000
0
5
10
15
20
Threshold


Count of Same Reviewer ID Below Each Threshold
Counts
Threshold
0
50
100
150
200
250
300
350
5
10
15
20