Analyzing Amazon Dataset Proposing Better Metrics

Unstructured Data Management

Harish Bharatham

(worked with 3 other teammates)

TOPIC INTRODUCTION

What is Amazon

Amazon is one of the largest online shopping websites in the world. The site is widely known for its wide selection of books, although the site has expanded to sell electronics, music, furniture, and apparel. Similar to eBay, users can also purchase and sell items using Amazon's online marketplace system. Amazon was founded in 1995 by Jeff Bezos and is based out of Seattle, Washington.

Amazon Reviews

Reviews by ordinary people have become an essential mechanism for selling almost anything online; they are used for resorts, dermatologists, neighborhood restaurants, high-fashion boutiques, churches, parks, astrologers and healers — not to mention products like garbage pails, tweezers, spa slippers and cases for tablet computers. In many situations, these reviews are supplanting the marketing department, the press agent, advertisements, word of mouth and the professional critique. But not just any kind of review will do. They have to be somewhere between enthusiastic and ecstatic. – NY Times

Amazon allows users to submit reviews to the web page of each product. Reviewers must rate the product on a rating scale from one to five stars. Amazon provides a badging option for reviewers which indicate the real name of the reviewer (based on confirmation of a credit card account) or which indicate that the reviewer is one of the top reviewers by popularity. Customers may comment or vote on the reviews, indicating whether they found a review helpful to them. If a review is given enough "helpful" hits, it appears on the front page of the product. In 2010, Amazon was reported as being the largest single source of Internet consumer reviews.

However, off late Amazon has been receiving a lot of backlash for having fake reviews on the website. The sellers seek shoppers on dozens of networks, including Amazon Review Club and Amazon Reviewers Group, to give glowing feedback in exchange for money or other compensation. The practice artificially inflates the ranking of thousands of products, experts say, misleading consumers. This is one aspect we address in this project by using SAS Text Miner.

In this project, we have focussed on the recommendation/ review system of Amazon.com. We have utilized the knowledge gained in this course to create a pseudo database schema in MongoDB as well as Neo4j (in the form of graph data model). Next, we have presented an overview of the dataset available by executing MongoDB queries on the dataset to get a general overview of the data. Based on these queries (but not limited to them) we have come up with some findings/ patterns in the data and have executed MongoDB queries for them as well. We have then attempted to formulate some interesting analytics metrics for the Amazon website specifically related to the product reviews and reviewers. We have also identified the queries required to run for identifying helpfulness of reviews and also identifying common set of reviewers who review a common set of products. Lastly, we have used SAS text miner to identify fake reviews among the dataset and then we have designed a template of the webpage using the analytics metrics.

DESIGN

1. MongoDB Pseudo Database Schema
   Before defining the pseudo database, we will define the constraints and indexes that we will be using for the Amazon dataset. Following the constraints defined:
   a. Constraint: The overall of a review must be either 1, 2, 3, 4 or 5.

```
db.createCollection("review",
  {
   validator:
    { "overall": { $in: [1,2,3,4,5] }
    }
  })
```

   b. Constraint: The price of a product must be greater than $0.

```
db.product.aggregate("product",
{
        validator:
        { "price" : {$gt : 0}}
        }
})
```

   c. Index: We can create a textual index on the reviewText of reviews.

```
db.review.createIndex( { reviewText: "text" } )
```

Pseudo Database Schema

For the Amazon dataset provided, we propose the following pseudo database schema.

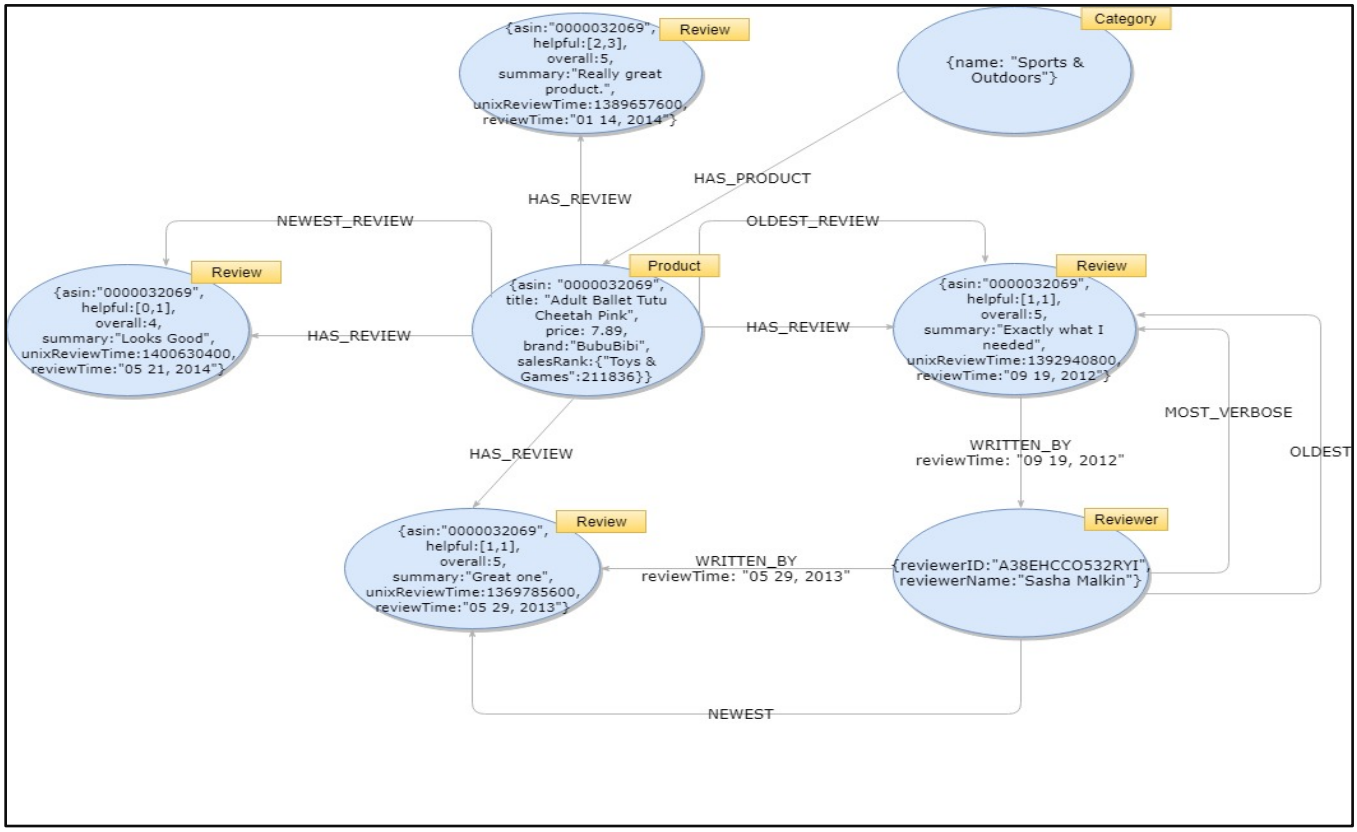
Embedding Reviews in Products

Placing orders and recommendation system for products are the two pillars of the Amazon website. Since, we are dealing with reviews in this project, we think that reviews of a particular product should be embedded within the specific product document inside the product collection. Whenever a reviewer clicks on a product for purchasing, the product page opens and all reviews for that product are displayed along with a review histogram. Therefore, queries involving retrieving the reviews for a product will run faster and more efficiently since we will get all the data in the same collection. We will not need to reference multiple collections resulting in increase in the running time of the query. Basically, we will be having an array of the review documents inside the corresponding product document. Below is a sample of the product collection after embedding reviews.

```
{
"asin" : "0001048791",
"salesRank" : { "Books" : 6334800 },
```

"imUrl" : "http://ecx.images-amazon.com/images/I/51MKP0T4DBL.jpg",
"categories" : [["Books"]], "title" : "The Crucible: Performed by Stuart Pankin, Jerome Dempsey & Cast"
reviews:[
{
"reviewerID" : "AA8M6331NI1EN",
"asin" : "B00000JBLH",
"reviewerName" : "ZombieMom",
"helpful" : [0,0],
"reviewText" : "Bought this for my boss because he lost his. He loves this calculator & would not be caught without it. It really helps him with his day to day work & he is the CEO of my company. If the CEO swears by them then they must be a great little calculator.",
"overall" : 5,
"summary" : "Fast shipping & great price for this awesome calculator",
"unixReviewTime" : 1384387200,
"reviewTime" : "11 14, 2013",
"category" : "Office"
},
{
"reviewerID" : "AA8M6331NI1EN",
"asin" : "B00000JBLH",
"reviewerName" : "ZombieMom",
"helpful" : [0, 0],
"reviewText" : "Bought this for my boss because he lost his. He loves this calculator & would not be caught without it. It really helps him with his day to day work & he is the CEO of my company. If the CEO swears by them then they must be a great little calculator.",
"overall" : 5,
"summary" : "Fast shipping & great price for this awesome calculator",
"unixReviewTime" : 1384387200,
"reviewTime" : "11 14, 2013",
"category" : "Office"
}, …
]}

2. Neo4j Graph Data Model



Description of Graph Data Model

We have identified the following entities based on our analysis of the Amazon dataset.

1. Product
2. Review
3. Reviewer
4. Category

We have taken these entities to be the Node Types which are represented in the labels for each node in our model. Based on the Cypher query language, each node consists the respective attributes/ properties separated by commas and enclosed in curly braces.

The relationships between the nodes are represented using directed lines. The direction of the relationship basically gives more semantics to the relationship and tells us as that we can only read the relationship in that direction. Few of the relationships also have some properties. For example, WRITTEN_BY relationship has the date property which gives the date on which that particular user wrote that particular review.

Note: In our illustration of graph data model, we have listed only a few properties for each node due to space constraints and to keep the clarity of the model intact. We have mentioned all the properties of each node in the following paragraphs.

While preparing our graph data model, we have considered that the queries that we have written so far can be answered using our model. Few examples of the queries and their corresponding relationships in our data model are:

1. Query: Identify date of oldest review per reviewer.

   This query is answered by the OLDEST relationship between the User and Review nodes. The query groups the User collection on the basis of reviewerID. Now, taking each reviewerID we can easily find its oldest review using OLDEST relationship.
   Note: We have further extended our model to identify the newest review given by a reviewer, which can be answered using the NEWEST relationship between Reviewer and Review nodes.
2. Query: Top 10 most verbose reviewers.

   To answer this query, we have created a relationship MOST_VERBOSE between User and Review nodes. This will give us the user who has written the longest text for a review. In future, if there comes another user with an even longer text for that particular review, this relationship will change to something like PREVIOUS or SECOND_MOST_VERBOSE (not shown in the model) and the relationship between the new User node and the Review node will become MOST_VERBOSE.
   In order to determine the oldest or newest reviewer for any product, we have the WRITTEN_BY relationship with 'date' property. We can easily sort the reviews based on their date and then retrieve the corresponding reviewer.

The following tables explain the details about the nodes and relationships used in our data model and the thought process behind this approach.

Nodes:

Node No	Label	Properties	Justification
Node 1	Review	reviewerID asin reviewerName helpful reviewText overall summary unixReviewTime reviewTime category	Review is an entity that we want to model and store information about. We want to use our graph data model for queries involving reviews, so we have made it as a Node. Example of query: Display top most useful reviews per business.
Node 2	Product	asin related title price salesRank imUrl brand categories	Product is an entity that we want to model and store information about. We want to use our graph data model for queries involving Product, so we have made it as a Node. Example of query: Display average number of reviews per product.
Node 3	Reviewer	reviewerID reviewerName	Reviewer is an entity that we want to model and store information

			about. We want to use our graph data model for queries involving Reviewer, so we have made it as a Node. Example of query: Identify the reviewer who has been reviewing for the longest period of time.
Node 4	Category	category_name	Category is an entity that we want to model and store information about. We want to use our graph data model for queries involving Category, so we have made it as a Node. Example of query: Display average review ratings per category. Note: This is not a previously asked query but might be a useful statistic to determine the credibility of a business.

Relationships:

Relationship Name	Nodes Involved	Properties	Justification
HAS_PRODUCT (Category HAS_PRODUCT Product)	Category1 Product1		Our graph data model includes product and the category to which that product belongs. We can create this relationship by matching the category name (of Category node) to the categories (of Product node). During graph traversal, we can simply go to the particular category whose products we are looking for and then traverse through only those relationships which emerge from that Category node and end at any Product node. This way we will be able to access all those products that belong to that particular category.
WRITTEN_BY (Review WRITTEN_BY Reviewer)	Review3 Reviewer1	reviewTime	Each review is written by a reviewer. This relationship will have an attribute known as reviewTime which denotes the date on which the reviewer posted the review.
HAS_REVIEW (Product HAS_REVIEW Review)	Product1 Review2		Every product has multiple reviews. A product may or may not have any reviews.

NEWEST_REVIEW (Product NEWEST_REVIEW Review)	Product1 Review2		In the graph data model, every product will have newest review. This traversal will help us answer queries such as: Identify the reviewer with the newest review for each product.
OLDEST_REVIEW (Product OLDEST_REVIEW Review)	Product1 Review3		Every product has an oldest review. This traversal will help us answer queries such as: Identify the reviewer with the oldest review for each product.
OLDEST (Reviewer OLDEST Review)	Reviewer1 Review1		Review is written by a Reviewer. Each reviewer will have an oldest review which will be the first review that he/she writes. In order to find such reviews faster, we have a direct relationship between Reviewer and Review named, OLDEST. This way the user will not need to traverse through all Review nodes for a Reviewer and check the reviewTime property and then determine the oldest Review. This approach of traversal will help us answer queries such as: Identify the oldest review for a reviewer.
NEWEST (Reviewer NEWEST Review)	Reviewer1 Review2		Review is written by a Reviewer. Each reviewer will have a newest review which will be the most recent or last review that he/she writes. In order to find such reviews faster, we have a direct relationship between Reviewer and Review named, NEWEST. This way the user will not need to traverse through all Review nodes for a Reviewer and check the reviewTime property and then determine the newest Review. This approach of traversal will help us answer queries such as: Identify the newest review for a reviewer.
MOST_VERBOSE (Reviewer MOST_VERBOSE Review)	Reviewer1 Review3		In our graph data model, every reviewer will have one review for which he/she has written the maximum text. Every time the reviewer enters another review with more text, this new review becomes the most verbose review and the previous one becomes the second most verbose review. By modelling in this way, we can

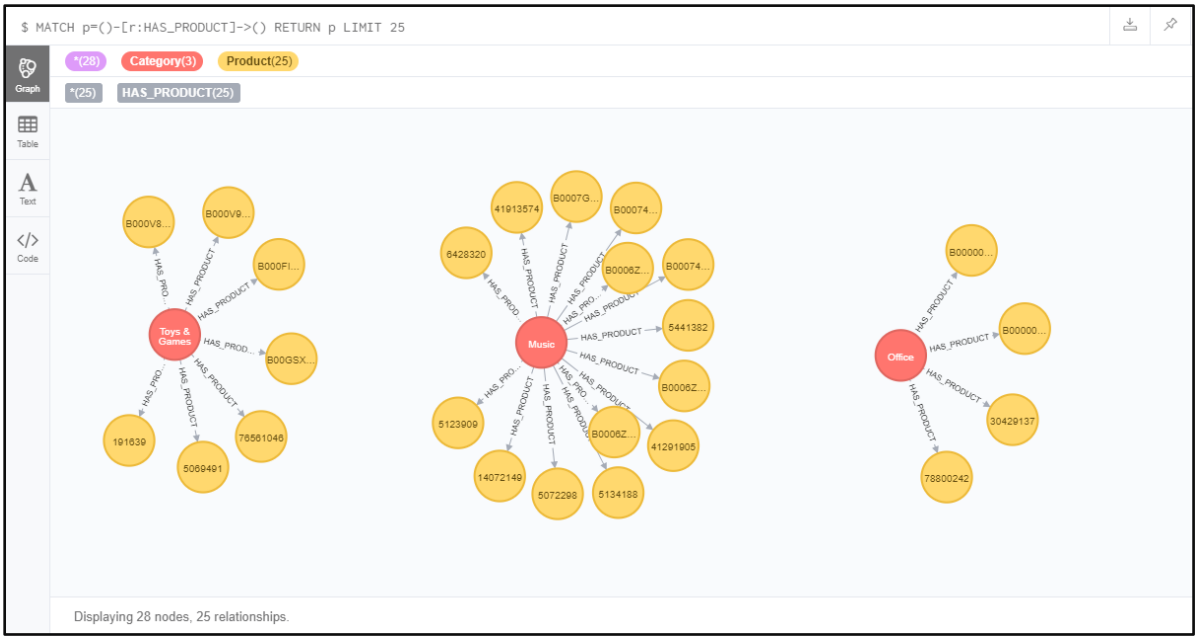
			answer queries such as: Most verbose reviewers.

Following are the CREATE queries for the three major relationships of our graph data model.

1. Category HAS_PRODUCT Product

   MATCH (category:Category), (product:Product)
   WHERE product.categories CONTAINS category.category_name
   CREATE (category)-[:HAS_PRODUCT]->(product)

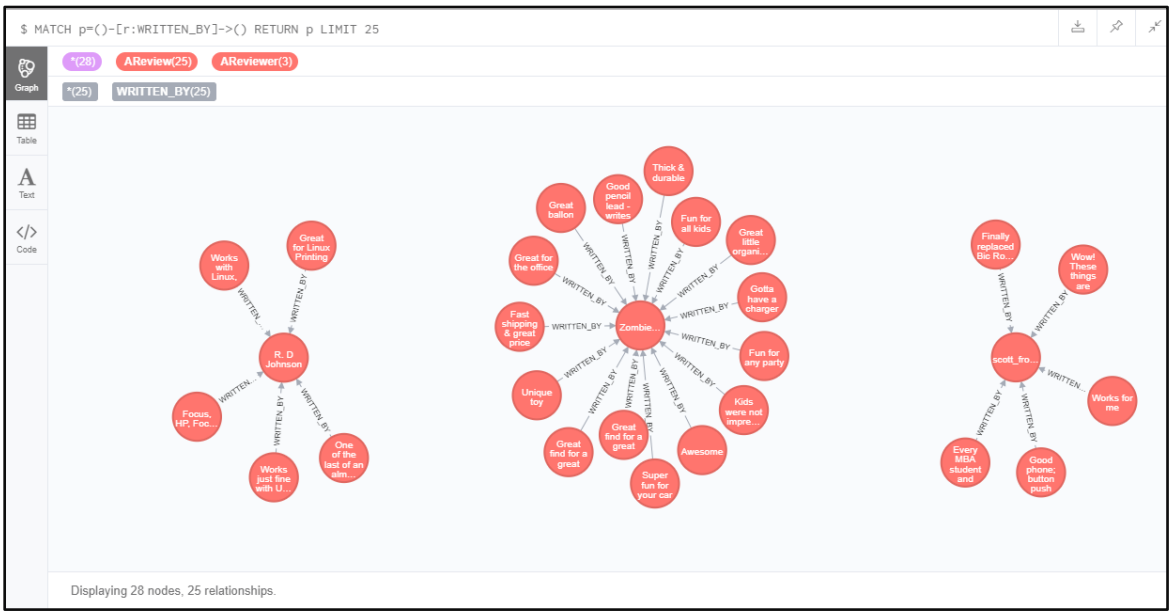
   Representative Sample:



2. Review WRITTEN_BY Reviewer

   MATCH (review:Review), (reviewer:Reviewer)
   WHERE review.reviewerID = reviewer.reviewerID
   CREATE (review)-[:WRITTEN_BY]->(reviewer)

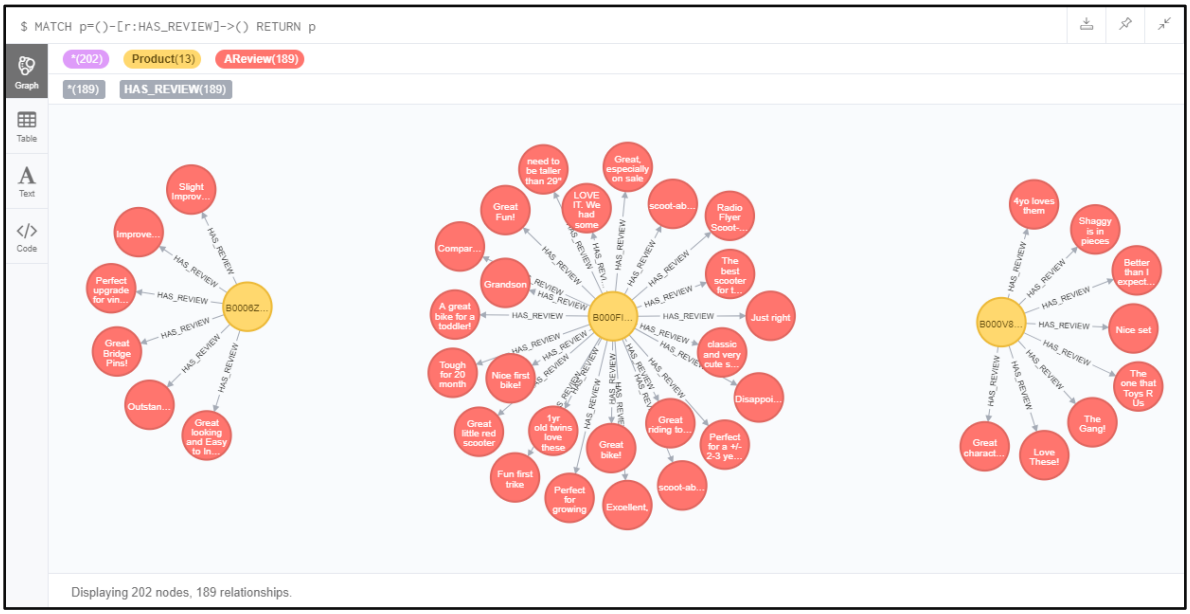
Representative Sample:



3. Product HAS_REVIEW Review

MATCH (review:Review), (product:Product)
WHERE review.asin = product.asin
CREATE (product)-[:HAS_REVIEW]->(review)

Representative Sample:



BASIC UNDERSTANDING OF THE DATA

1. MongoDB Queries

Instead of going with all the reviews for all categories, we have selected the “Office Products”, “Musical Instruments” and “Toys & Games” categories in order to limit our dataset to a reasonable size. We have also downloaded and imported the product metadata which gives all information about a particular product.
After importing the JSON files for the reviews of each of the categories, we have added a “category” attribute to each of these collections. We have the value of this attribute as “Office”, “Music” and “Toys & Games” respectively. Next, we have merged these individual review collections into a single “review” collection. We have used the following queries to do so.
For adding category attribute:

```
db.office_products.update({},{$set:{"category": "Office"}},{multi:true})
db.musical_instruments.update({},{$set:{"category": "Music"}},{multi:true})
db.toys_and_games.update({},{$set:{"category": "Toys & Games"}},{multi:true})
```

For merging the review collections for each category into a single review collection:

```
db.office_products.copyTo(“review”)
db.musical_instruments.copyTo(“review”)
db.toys_and_games.copyTo(“review”)
```

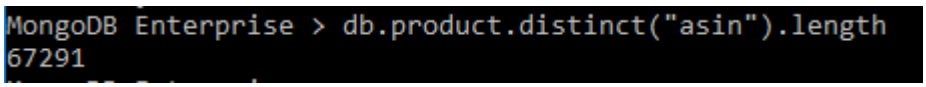
Following are the queries we have executed to get an overview of the dataset under consideration:

a. What is the overall number of products?

Wording of query

```
db.product.distinct("asin").length
```

Snapshot showing successful execution



Summary of results

To find the total number of products in the Amazon dataset, we have simply applied the distinct function on the asin field and then calculated the length of the collection. The length function returns the total number of documents in that collection.

b. What is the overall number of reviews?

Wording of query

db.review.count()

Snapshot showing successful execution

```
MongoDB Enterprise > db.review.count()
231116
```

Summary of results

In this query, we have used the count function on the review collection to calculate the total number of reviews in our dataset.

c. What is the overall number of reviewers?

Wording of query

db.review.distinct("reviewerID").length

Snapshot showing successful execution

```
MongoDB Enterprise > db.review.distinct("reviewerID").length
23998
```

Summary of results

In order to get the total number of reviewers, we have identified the distinct reviewer IDs in the review collection. After applying the length function on this, we get the total number of these distinct reviewers. It is important to apply the distinct function here because a reviewer can review more than one products also. So doing a simple count on the review collection will not give the correct number of reviewers for the products.

d. What is the overall number of reviews with ratings less than 3?

Wording of query

db.review.find({overall:{$lt:3}},{_id:0, asin:1, summary:1, overall:1}).count()

Snapshot showing successful execution

```
MongoDB Enterprise > db.review.find({overall:{$lt:3}},{_id:0, asin:1, summary:1, overall:1}).count()
14328
```

Summary of results

In order to get the total number of reviews having a rating less than 3, we have used overall:{$lt:3} as the where criteria for the find function. We have then applied the count function on the result obtained to get the total number of matched reviews. We have displayed the asin, summary and overall fields in the final output and we have also supressed the _id field.

e. What is the overall number of reviews with ratings more than 3?

Wording of query

db.review.find({overall:{$gt:3}},{_id:0, asin:1, summary:1, overall:1}).count()

Snapshot showing successful execution

```
MongoDB Enterprise > db.review.find({overall:{$gt:3}},{_id:0, asin:1, summary:1, overall:1}).count()
194599
```

Summary of results

In order to get the total number of reviews having a rating greater than 3, we have used overall:{$gt:3} as the where criteria for the find function. We have then applied the count function on the result obtained to get the total number of matched reviews. We have displayed the asin, summary and overall fields in the final output and we have also supressed the _id field.

f. What is the total number of reviews per product?

Wording of query

db.review.aggregate([{$group: {_id: "$asin", review_count: {$sum:1}}}])

Snapshot showing successful execution

```
MongoDB Enterprise > db.review.aggregate([{$group: {_id: "$asin", review_count: {$sum:1}}}] )
{ "_id" : "B00JBIVXGC", "review_count" : 13 }
{ "_id" : "B00FXKIG5I", "review_count" : 7 }
{ "_id" : "B00EPWAQS0", "review_count" : 5 }
{ "_id" : "B00C5B20QE", "review_count" : 5 }
{ "_id" : "B00BTGMI5O", "review_count" : 6 }
{ "_id" : "B00BLQTZDA", "review_count" : 6 }
{ "_id" : "B00BL6IYW8", "review_count" : 7 }
{ "_id" : "B00BHK2UHI", "review_count" : 6 }
{ "_id" : "B00B5OY8EE", "review_count" : 5 }
{ "_id" : "B00B1N06PO", "review_count" : 6 }
{ "_id" : "B00AZUAORE", "review_count" : 24 }
{ "_id" : "B00A716FB0", "review_count" : 11 }
{ "_id" : "B009S814U0", "review_count" : 5 }
{ "_id" : "B009MDMQJ4", "review_count" : 5 }
{ "_id" : "B0097WDJZA", "review_count" : 11 }
{ "_id" : "B0087UPSLQ", "review_count" : 8 }
{ "_id" : "B007T8OGLK", "review_count" : 16 }
{ "_id" : "B007T8KI6M", "review_count" : 8 }
{ "_id" : "B007K3EGXW", "review_count" : 8 }
{ "_id" : "B007K3EGWI", "review_count" : 5 }
Type "it" for more
MongoDB Enterprise >
```

Summary of results

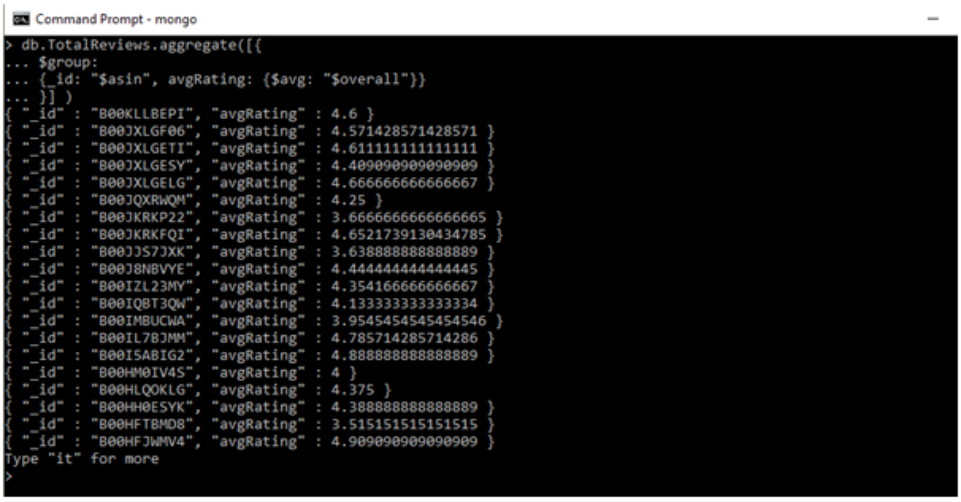
Here we are the total number of reviews for each product and for that first we are grouping all the reviews by asin (product ID) field and then calculating the total number of documents for each product by using $sum function.

g. What is the average rating per product?

Wording of query

```
db.review.aggregate([{
$group:
{_id: "$asin", avgRating: {$avg: "$overall"}}
}] )
```

Snapshot showing successful execution



Summary of results

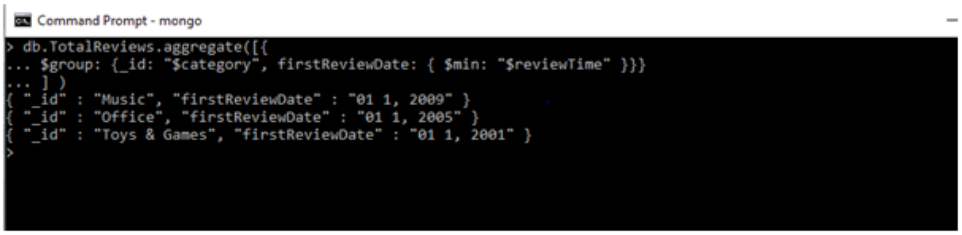
Here we are querying out the top 10 products which received most positive reviews. We are grouping the total reviews by asin (product id) which is unique for each product and then we are calculating the average of overall for each product by using $avg function. After that we are sorting the results in descending order and limiting the results to top 10.

h. What is the date of the first review per category?

Wording of query

```
db.review.aggregate([{
$group:
{_id: "$category", firstReviewDate: { $min: "$reviewTime" }}
}])
```

Snapshot showing successful execution



Summary of results

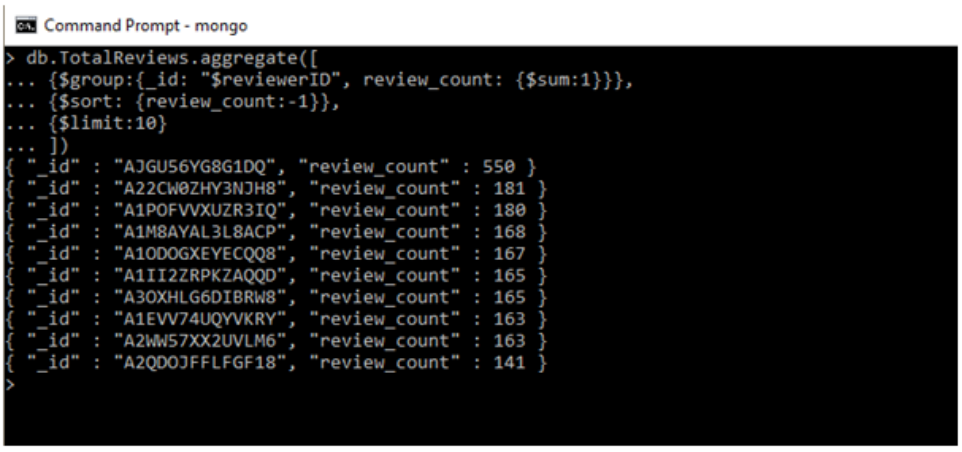
Here we are querying the first review per category and the date of first review per category. We are grouping the total reviews by category field and then taking the minimum of reviewTime field by using $min function.

i. What is the top 10 most prolific reviewers?

Wording of query

```
db.review.aggregate([
{$group:{_id: "$reviewerID", review_count: {$sum:1}}},
{$sort: {review_count:-1}},
{$limit:10}
])
```

Snapshot showing successful execution



Summary of results

Here we are querying out the 10 most prolific reviewers i.e. with the most number of reviews. We are grouping the total reviews by reviewer id which is unique for each reviewer and then

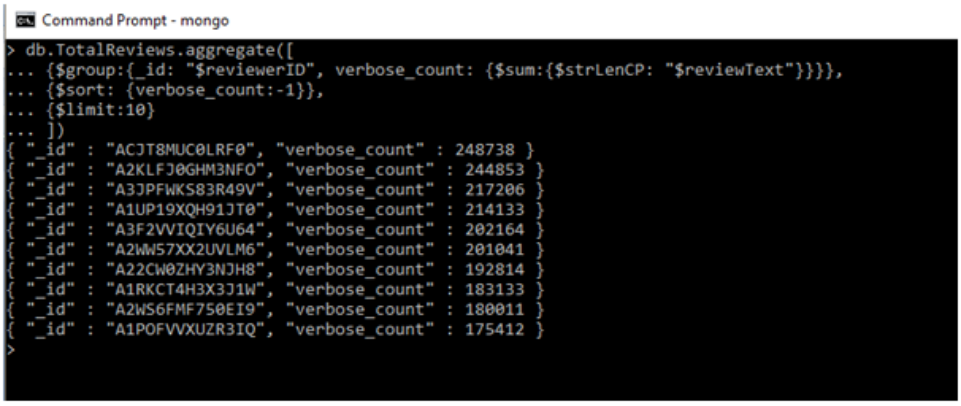
counting the number of documents for each reviewer. After that we are sorting the results in descending order and limiting the results to top 10.

j. What is the top 10 most verbose reviewers?

Wording of query

```
db.review.aggregate([
{$group:{_id: "$reviewerID", verbose_count: {$sum:{$strLenCP: "$reviewText"}}}},
{$sort: {verbose_count:-1}},
{$limit:10}
])
```

Snapshot showing successful execution



Summary of results

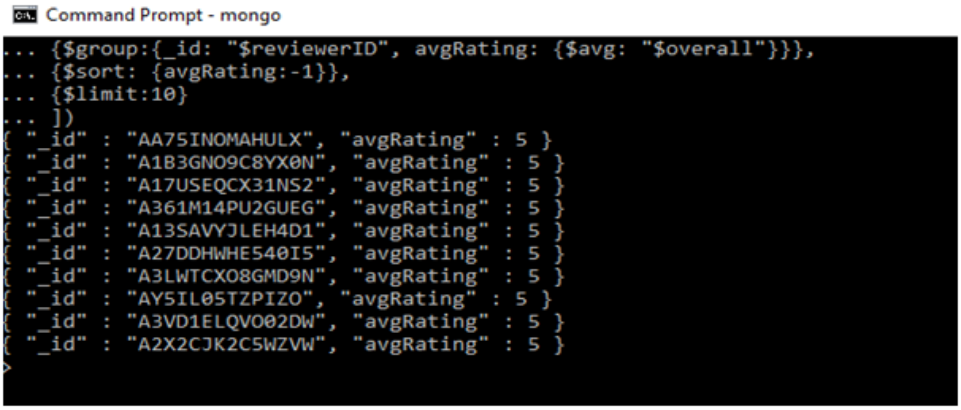
Here we are querying out the 10 most verbose reviewers i.e. with the max amount of review text. We are grouping the total reviews by reviewer id which is unique for each reviewer and then we are calculating the length of review text for each reviewer. After that we are sorting the results in descending order and limiting the results to top 10.

k. What is the top 10 most positive reviewers?

Wording of query

```
db.review.aggregate([
{$group:{_id: "$reviewerID", avgRating: {$avg: "$overall"}}},
{$sort: {avgRating:-1}},
{$limit:10}
])
```

Snapshot showing successful execution



Summary of results

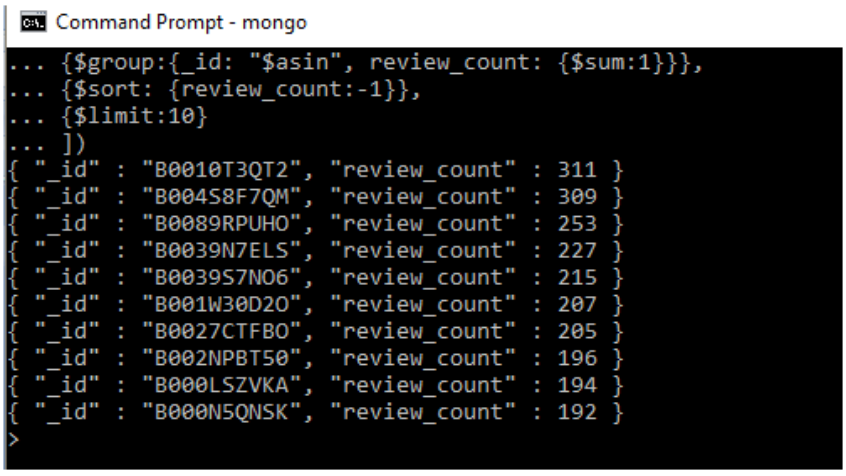
Here we are querying out the 10 most positive reviewers. We are grouping the total reviews by reviewer id which is unique for each reviewer and then we are calculating the average of overall field for each reviewer. After that we are sorting the results in descending order and limiting the results to top 10.

l. What are the top 10 products with most number of reviews?

Wording of query

```
db.review.aggregate([
{$group:{_id: "$asin", review_count: {$sum:1}}},
{$sort: {review_count:-1}},
{$limit:10}
])
```

Snapshot showing successful execution



Summary of results

Here we are querying out the top 10 products with most number of reviews. We are grouping the total reviews by asin (product id) which is unique for each product and then we are calculating the total number of documents for each product. After that we are sorting the results in descending order and limiting the results to top 10.

m. What is the top 10 most negative reviewers?

Wording of query

```
db.review.aggregate([
{$group:{_id:{reviewerId: "$reviewerID", name: "$reviewerName"}, avgRating: {$avg:
"$overall"}}},
{$sort: {avgRating:1}},
{$limit:10}
])
```

Snapshot showing successful execution

Summary of results

Here we are querying out the 10 most negative reviewers. We are grouping the total reviews by reviewer id which is unique for each reviewer and then we are calculating the average of overall field for each reviewer. After that we are sorting the results in ascending order and limiting the results to top 10.

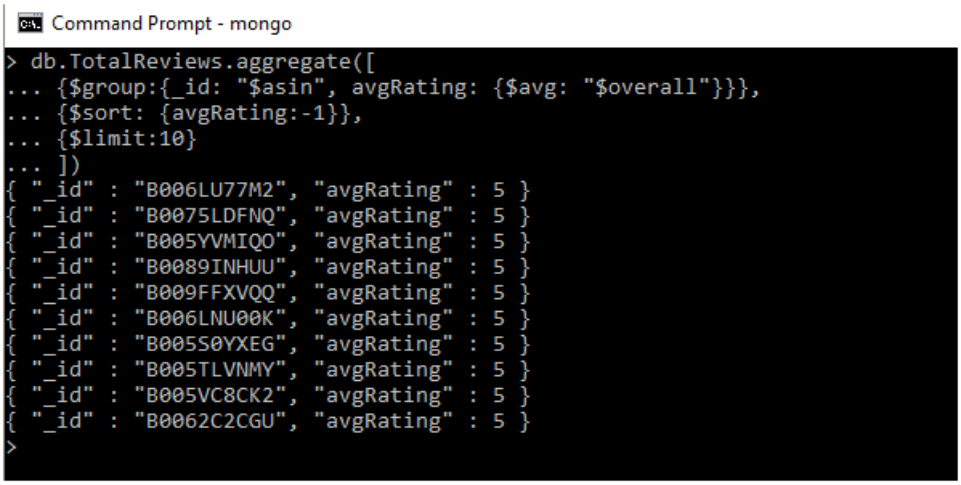
n. What are the top 10 products with most number of positive reviews?

Wording of query

```
db.review.aggregate([
{$group:{_id: "$asin", avgRating: {$avg: "$overall"}}},
{$sort: {avgRating:-1}},
```

{$limit:10}
])

Snapshot showing successful execution



Summary of results

Here we are querying out the top 10 products which received most positive reviews. We are grouping the total reviews by asin (product id) which is unique for each product and then we are calculating the average of overall for each product by using $avg function. After that we are sorting the results in descending order and limiting the results to top 10.

o. What are the top 10 products with most number of negative reviews?

Wording of query

db.review.aggregate([
{$group:{_id: "$asin", avgRating: {$avg: "$overall"}}},
{$sort: {avgRating:1}},
{$limit:10}
])

Snapshot showing successful execution

```
> db.TotalReviews.aggregate([
... {$group:{_id: "$asin", avgRating: {$avg: "$overall"}}},
... {$sort: {avgRating:1}},
... {$limit:10}
... ])
{ "_id" : "B000AR84B0", "avgRating" : 1 }
{ "_id" : "B0017867QK", "avgRating" : 1 }
{ "_id" : "B002VUCA00", "avgRating" : 1.1111111111111112 }
{ "_id" : "B003B1WF7Q", "avgRating" : 1.3333333333333333 }
{ "_id" : "B009F7OUE6", "avgRating" : 1.375 }
{ "_id" : "B001TE5C1S", "avgRating" : 1.4 }
{ "_id" : "B001FVSZAK", "avgRating" : 1.4 }
{ "_id" : "B000SDU7ZW", "avgRating" : 1.4 }
{ "_id" : "B003AN6H42", "avgRating" : 1.4 }
{ "_id" : "B001PO6IL0", "avgRating" : 1.5 }
>
```

Summary of results

Here we are querying out the top 10 products which received most negative reviews. We are grouping the total reviews by asin (product id) which is unique for each product and then we are calculating the average of overall for each product by using $avg function. After that we are sorting the results in ascending order and limiting the results to top 10.

FINDINGS AND INTERPRETATION

We have the following findings from the Amazon dataset provided.

a. Comparison between the overall number of positive and negative reviews as well as per category.

   First, we calculated the total number of reviews having a rating greater than or equal to 3, which gives us the total number of positive reviews. Next, we calculated the overall number of reviews having a rating less than 3 i.e. negative reviews.

   Query:

   db.review.count({overall:{$gte:3}}) => 216788
   db.review.count({overall:{$lt:3}}) => 14328

```
MongoDB Enterprise > db.review.count({overall:{$gte:3}})
216788
MongoDB Enterprise > db.review.count({overall:{$lt:3}})
14328
MongoDB Enterprise >
```

   Next, we have calculated the number of positive and negative reviews per category. We have done this for all 3 categories which we have considered.

   For category “Office”,
   db.review.find({category:"Office", overall:{$gte:3}}).count() => 50402

db.review.find({category:"Office", overall:{$lt:3}}).count() => 2856

```
MongoDB Enterprise > db.review.find({category:"Office", overall:{$gte:3}}).count()
50402
MongoDB Enterprise > db.review.find({category:"Office", overall:{$lt:3}}).count()
2856
MongoDB Enterprise >
```

For category “Music”,
db.review.find({category:"Music", overall:{$gte:3}}).count() => 9794
db.review.find({category:"Music", overall:{$lt:3}}).count() => 467

```
MongoDB Enterprise > db.review.find({category:"Music", overall:{$gte:3}}).count()
9794
MongoDB Enterprise > db.review.find({category:"Music", overall:{$lt:3}}).count()
467
MongoDB Enterprise >
```

For category “Toys& Games”,
db.review.find({category:"Toys& Games", overall:{$gte:3}}).count() => 156592
db.review.find({category:"Toys& Games", overall:{$lt:3}}).count() => 11005

```
MongoDB Enterprise > db.review.find({category:"Toys& Games", overall:{$gte:3}}).count()
156592
MongoDB Enterprise > db.review.find({category:"Toys& Games", overall:{$lt:3}}).count()
11005
MongoDB Enterprise >
```

b. Identification of the product category which is being reviewed for the longest period of time.

We identified the category of products which is being reviewed for the longest period of time. From this we can interpret that this particular category has been around for the longest period of time and has been constantly receiving reviews. This may indicate that the popularity and sales of that product category because only if the product (belonging to that category) is being is being bought by customers will it receive more number of reviews. The dataset available consists of reviews only till the year 2014; the newest review date we have in our review collection is 23rd July 2014.

Just on a side note, the reviews go back to 28th July 2014. This implies our dataset consists of reviews from the year 2000 to 2014 for the categories, Office Products, Musical Instruments and Toys & Games.

Query:

1. We have first converted the review time for each review from String to Date format using the below function. This was required to be done because we preferred to display the oldest and newest review times in date format instead of the unix review times (epochs).

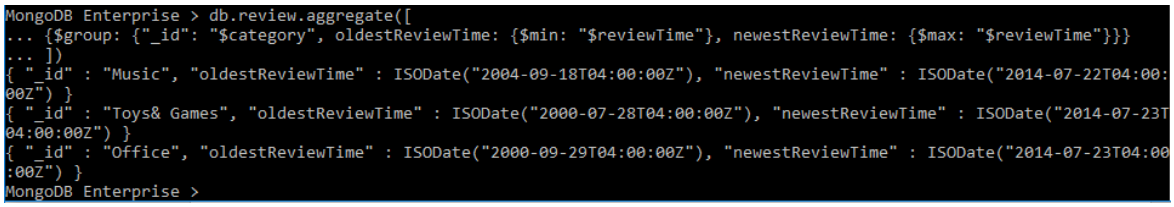
```
db.review.find().forEach(function(element){
  element.reviewTime = new Date(element.reviewTime);
  db.review.save(element);
})
```

This query simply converts the strings to dates and has no visible output.

2. Next, we simply grouped the review collection by category and then calculated the oldest review date and newest review date using the $min and $max functions respectively.

```
db.review.aggregate([
{$group: {"_id": "$category", oldestReviewTime: {$min: "$reviewTime"},
newestReviewTime: {$max: "$reviewTime"}}}
])
```

Output:



c. Identification of the product category which has the most number of recommended reviews.

We have formulated a definition for “recommended reviews” for a particular product. This definition is that if a reviewer precisely writes something like “I recommend this product to anyone.”, we consider it as a review which actually recommends buying the product. Similarly, if the review has words “not recommend”, then we will consider this review as one which discourages buying the product. Next for each category, we calculate the total number of reviews which recommends buying the overall products in that category. It is evident from the below query that the Toys & Games category has the most number of recommended products based on the corresponding reviews.

Query:

```
db.review.aggregate([
{$match: {$text: {$search:"recommend Recommend -\"not recommend\""}}},
{$group:{_id:"$category", recommendedReviews:{$sum:1}}}
])
```

```
MongoDB Enterprise > db.review.aggregate([
... {$match: {$text: {$search:"recommend Recommend -\"not recommend\""}}},
... {$group:{_id:"$category", recommendedReviews:{$sum:1}}}
... ])
{ "_id" : "Office", "recommendedReviews" : 7901 }
{ "_id" : "Music", "recommendedReviews" : 1162 }
{ "_id" : "Toys& Games", "recommendedReviews" : 19555 }
MongoDB Enterprise >
```

d. Identification of the product which has the most number of “also bought” and “also viewed” products.

For each product, Amazon shows some related products that were either viewed or bought by other customers. Amazon basically uses this strategy to increase its sales. If the user buys the current product as well as one or more of the “also viewed” and/or “also bought” products, this results in an increase in number of sales from Amazon’s perspective. Therefore, it will be helpful if we are able to determine the products which have the highest number of related (also bought/ also viewed) products.
In the below query, we assume that if the product has at least 10 also viewed and at least 10 also bought products, we consider it to have a high number of related products. The output shows the asin for the products which are considered to have a high number of related products.
Amazon can probably highlight the products which have high number of related products in order to lead the users on those product’s pages and increase their probability of buying their related products as well.

Query:

db.product.find({"related.also_viewed" : {$exists:true},
$where:'this.related.also_viewed.length>10', "related.also_bought" : {$exists:true},
$where:'this.related.also_bought.length>10'}, {asin:1, _id:0})

```
MongoDB Enterprise > db.product.find( {"related.also_viewed" : {$exists:true}, $where:'this.related.also_viewed.length>1
0', "related.also_bought" : {$exists:true}, $where:'this.related.also_bought.length>10'}, {asin:1, _id:0} )
{ "asin" : "0000032069" }
{ "asin" : "0000032034" }
{ "asin" : "0000031852" }
{ "asin" : "0000032050" }
{ "asin" : "0000031909" }
{ "asin" : "0000031895" }
{ "asin" : "0000031887" }
{ "asin" : "0005123909" }
{ "asin" : "0005119367" }
{ "asin" : "0005134188" }
{ "asin" : "0061997102" }
{ "asin" : "0071436588" }
{ "asin" : "0000031909" }
{ "asin" : "0000031852" }
{ "asin" : "0000032034" }
{ "asin" : "0000032050" }
{ "asin" : "0000031895" }
{ "asin" : "0000031887" }
{ "asin" : "0005123909" }
{ "asin" : "0005119367" }
Type "it" for more
MongoDB Enterprise >
```

ANALYTICS

MongoDB Metrics

Based on our analysis, we have come up with a few metrics which could be useful to include on Amazon's webpages for individual products and reviewers. These metrics are not available in Amazon's website currently.

A) Performance over time metric

Reason for choosing the metric:

This metric is shows the performance of each product with time. One of the problem with amazon reviews is the date of the reviews is not properly displayed to the user to show how old a particular review is. So we decided to show a graph that displays the rating of the product from the time it was launched.

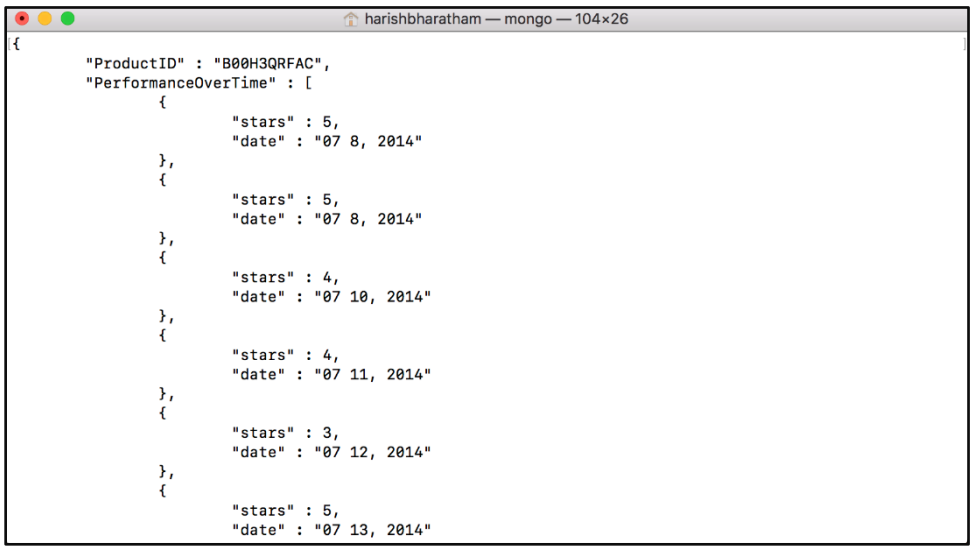
Wording of the query:

db.Reviews.aggregate([{$sort:{"unixReviewTime":1}},{$group:{_id:"$asin",PerformanceOverTime:{$push:{stars:"$overall",date:"$reviewTime"}}}},{$project:{ProductID:"$_id",PerformanceOverTime:{"$slice":["$PerformanceOverTime",10]},_id:0}},{$limit:5}],{allowDiskUse:true}).pretty()

Summary:

We used aggregate pipeline to group the data based on productID, and $push function to create an array of ratings with their time. We have sorted the data based on the review date in ascending order, in this way we will show the ratings of the product from the first review. This data can be used to display a graph on the product page.

Screenshot of Mongodb output:



Screenshot of proposed metric on amazon page:

B) Sort by Helpful votes

Reason for choosing the metric:

With this metric we have decided to sort the reviews based on helpful votes of reviews. One of the feature that amazon lacks in sorting is that the reviews cannot be sorted based on the usefulness of the reviews.

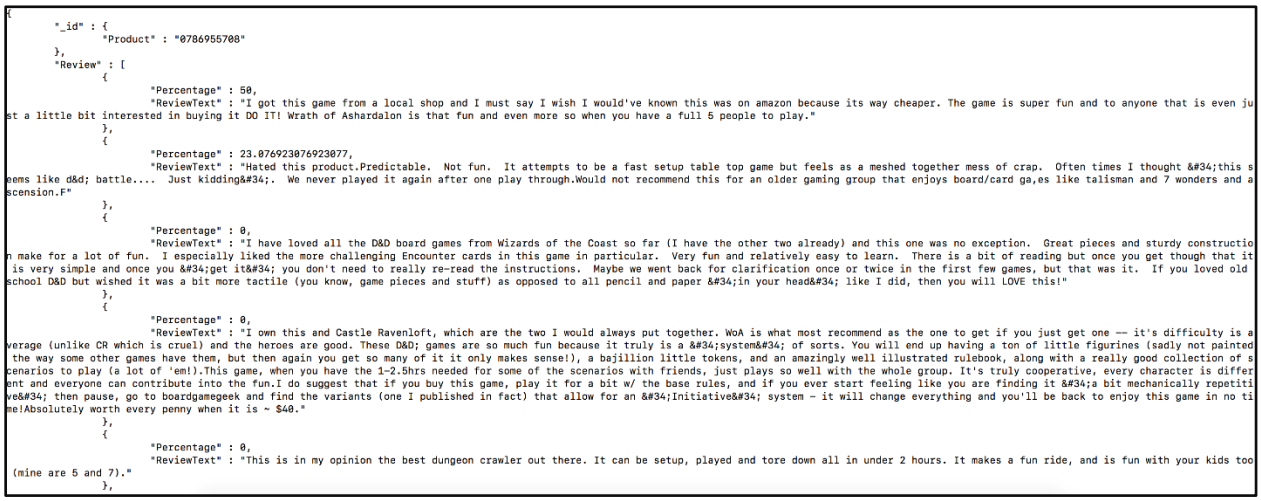
Wording of the query:

db.Reviews.aggregate([{$group:{"_id":{"Product":"$asin"},"Review":{$push:{Percentage:{$multiply:[{$cond:[{$eq:[{"$arrayElemAt":["$helpful",1]},0]},0,{$divide:[{"$arrayElemAt":["$helpful",0]},{"$arrayElemAt":["$helpful",1]}]}]},100]},"ReviewText":"$reviewText"}}}},{"$project":{"Review":"$Review"}},{$sort:{"Review.Percentage":-1}},{$limit:5}],{allowDiskUse:true}).pretty();

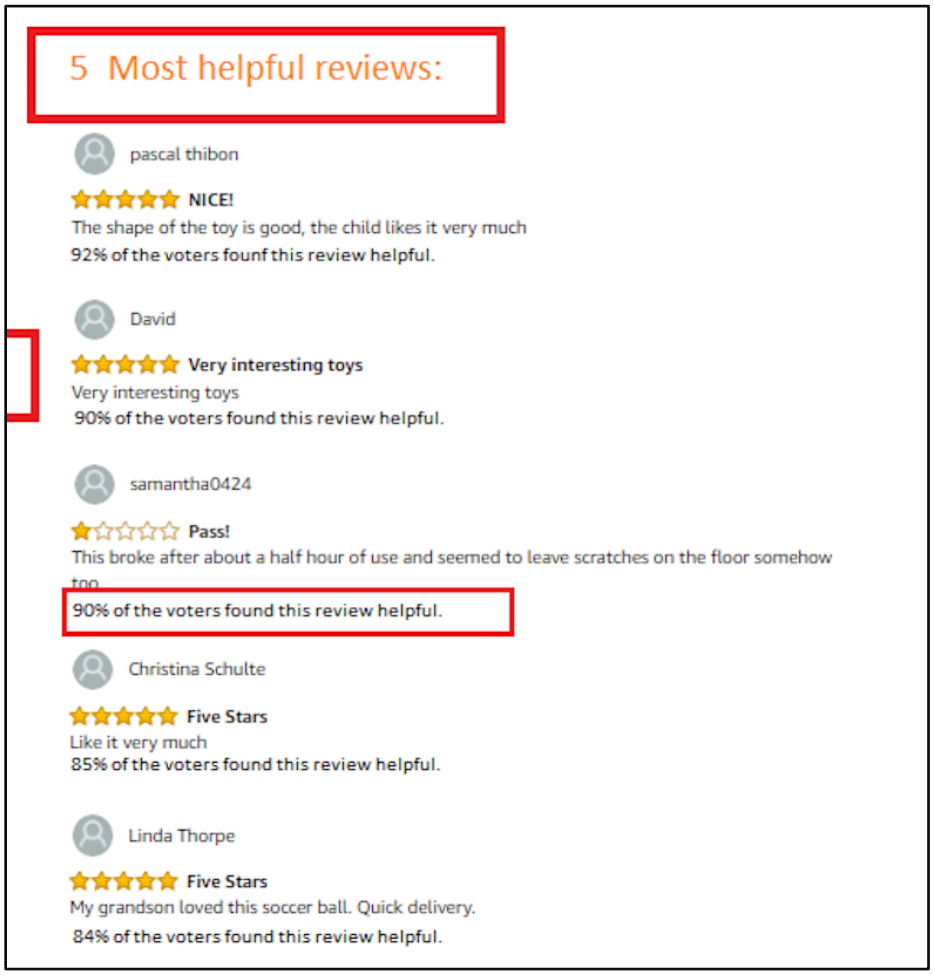
Summary:

We have decided to use aggregate pipeline to group reviews based on the productID and using $push to create an array of the review text and a percentage of helpfulness field that we calculated based on the data helpful votes data. We sorted the data based on this percentage, thereby displaying the most helpful review on the top.

Screenshot of Mongodb output:



Screenshot of proposed metric on amazon page:



C) Most Popular products under the same Category:

Reason for choosing this metric:

Amazon's website shows similar items for a product. These items being shown may not be very popular or may not be highly rated. We want to recommend the best products to the user. Therefore, we sorted the products in the same category based on the highest rating and then sorted again based on the most number of reviews. This will allow the user to view the most popular products under the same category. In this way, the user can make better informed decision while purchasing any product. According to us, rating and number of reviews will define popularity of product.

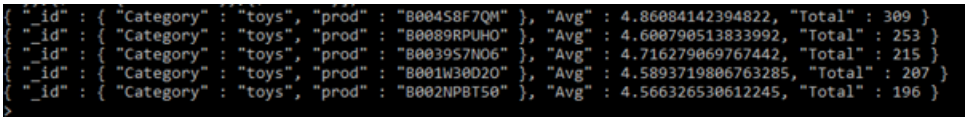
Wording of Query:

db.allreviews.aggregate([{$match:{"category":"toys"}},{$group:{"_id":{"Category":"$category","prod":"$asin"},"Avg":{$avg:"$overall"},"Total":{$sum:1}}},{$sort:{"Avg":-1}},{$sort:{"Total":-1}},{$limit:5}])

Summary:

For a given Category “Toys”, we have found out the average rating for each product and then calculated the total reviews available for each product. We sorted the results based on the average rating and the average number of reviews available for a particular product in the given category.

Screenshot from MongoDB:



Screenshot from our proposed Amazon webpage:

D) Most sold products under the same Category:

Reason for choosing this metric:

We would like the users to view the most sold products under the category that they are browsing. This will enable the users to make better informed decisions while choosing a product for buying.

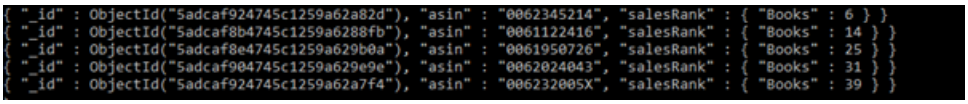
Wording of Query:

db.products.find({"salesRank.Books":{$exists:true}},{"salesRank.Books":1}).sort({"salesRank.Books":1}).limit(5)

Summary:

For a given category, we have displayed top 5 products with the least sales rank. Some of the documents in our collection did not have a sales rank attribute, so we excluded those documents from our query.

Screenshot from MongoDB:



Screenshot from our proposed Amazon webpage:

E) Percentage of helpfulness of each review:

Reason for choosing this metric:

Every review has helpful votes. These are displayed in Amazon. But in our webpage, we would like to display percentage of helpfulness of a review because there might be cases when a review has got 100 votes out of which there are only 5 helpful votes. So, the percentage of helpfulness is only 5 %, while there can be a review which received totally 5 votes out of which all 5 are Helpful. So, this review has 100% helpfulness. Hence, we would like the user to view this metric to judge whether to take a review into consideration or not. This will prevent the user from getting biased by a highly positive or highly negative review, which was otherwise not considered Helpful by majority of the users.

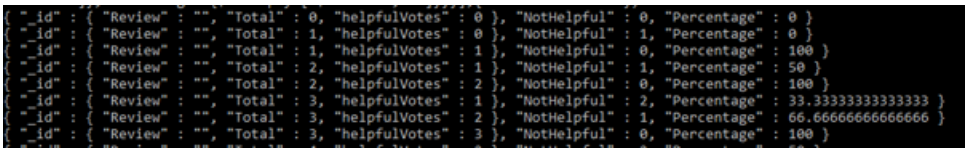
Wording of Query:

db.allreviews.aggregate([{$group:{"_id":{"Review":"$reviewText","Total":{"$arrayElemAt": ["$helpful", 1]},"helpfulVotes":{"$arrayElemAt": ["$helpful", 0]}}}},{$project:{"Decimal":{ $cond: [{ $eq: ["$_id.Total", 0] }, 0,{$divide:["$_id.helpfulVotes","$_id.Total"]}]}}},{$project:{"NotHelpful":{$subtract:["$_id.Total","$_id.helpfulVotes"]},"Percentage":{$multiply:["$Decimal",100]}}}],{allowDiskUse:true})

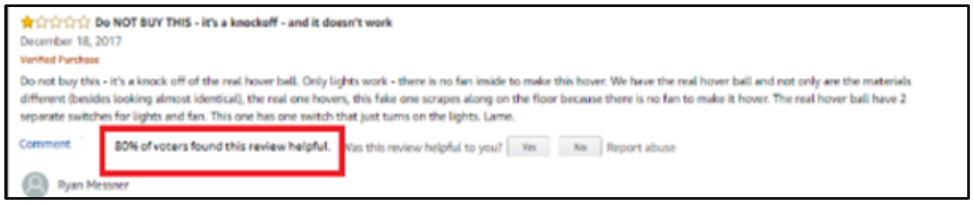
Summary:

In the json collection we have an array called helpful. The first element of the array is number of helpful votes and the second element is the total number of votes for a particular review. So, for each review, we have calculated helpful votes divided by total votes, multiplied by 100 to calculate the percentage. If a review did not have any votes, then we excluded those reviews from division to avoid divide by zero error and displayed percentage of helpfulness as zero for those reviews.

Screenshot from MongoDB:



Screenshot from our proposed Amazon webpage:



F) Top 5 recommended reviews for a product:

Reason for choosing this metric:

We have come up with our own definition of recommended reviews. We have defined a list of words and searched for those words in all the reviews for a particular product. Wherever we found a strong match based on high metascore, we displayed those reviews as recommended reviews. The reviews with the lowest metascore are considered as non-recommended reviews. We feel this will this will be a helpful criterion for the users to judge the quality of a product by taking into account the experience of other reviewers.

Wording of Query:

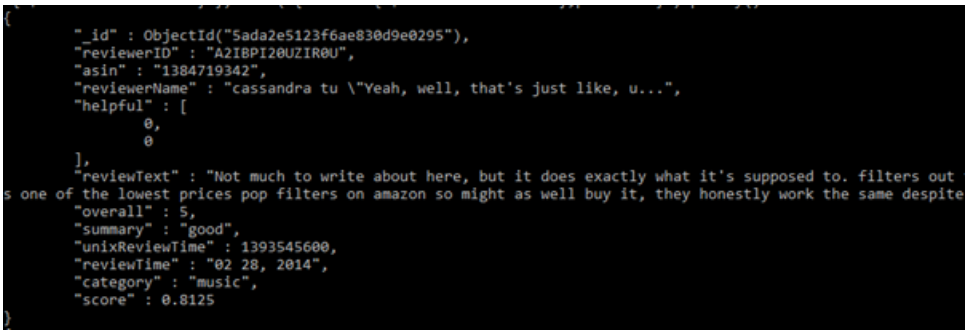
We first created an Index in reviewText field using the query: db.allreviews.createIndex({reviewText: "text"})

db.allreviews.find({ $text: { $search: "good bad worse excellent price quality worst great delivery fast slow service time cost poor" },"asin":"1384719342" },{ score: { $meta: "textScore" } }).sort({ score: { $meta: "textScore" } }).limit(5).pretty()

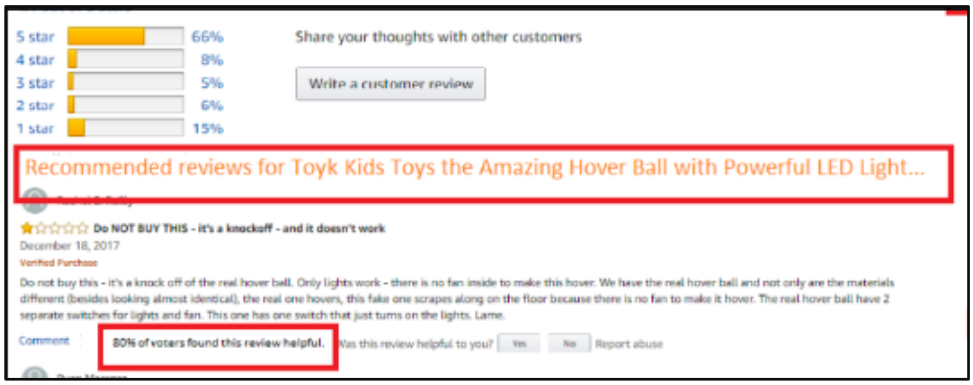
Summary:

For a given product, we searched reviews with the following words: "good bad worse excellent price quality worst great delivery fast slow service time cost poor". We sorted the obtained results based on metascore. So top 5 reviews with the highest relevance to these words are displayed.

Screenshot from MongoDB:



Screenshot from our proposed Amazon webpage:



G) Average rating given by a reviewer irrespective of products:

Reason for choosing this metric:

When a user reads a review and is strongly influenced by it, we feel it will help the user to also check whether the reviewer is a positive or negative reviewer in general. A reviewer may give only positive reviews or only negative reviews. So, this metric will help a prospective buyer to not be influenced by just one such review and to take other reviews also into consideration.

Wording of Query:

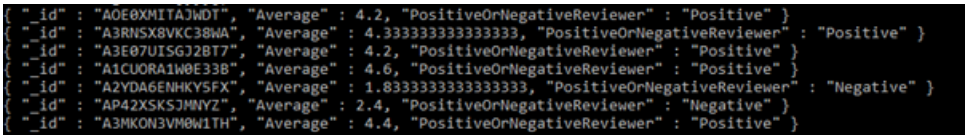
db.allreviews.aggregate([{$group:{"_id":"$reviewerID","Avg":{$avg:"$overall"}}},{$project:{"Average":"$Avg","PositiveOrNegativeReviewer":{$cond:[{$gte:["$Avg", 3]},"Positive","Negative"]}}}])

Summary:

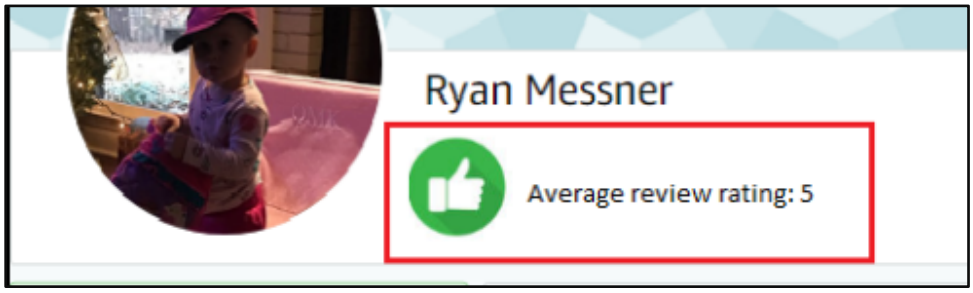
In this query, we have found out the average review given by each reviewer and based on that average, we have determined whether the reviewer is a positive or negative reviewer in general. If average rating

is greater than 3, the reviewer is considered to be a Positive reviewer. If the average rating is less than 3, the reviewer is considered to be a Negative reviewer in general.

Screenshot from MongoDB:



Screenshot from our proposed Amazon webpage:



H) Percentage of helpfulness of each reviewer:

Reason for choosing this metric:

In Amazon, only the total number of Helpful votes received by a reviewer is displayed. But in our proposed webpage, we would like to show the total number of Not Helpful votes and the percentage of helpfulness of a particular reviewer. This is because a reviewer may have received 5 helpful votes but may have received 20 not helpful votes, while another reviewer might have received 5 helpful votes and 0 not helpful votes. Hence, we would like to display the percentage of helpfulness by calculating total helpful votes received by a user divided by total votes received by a user.

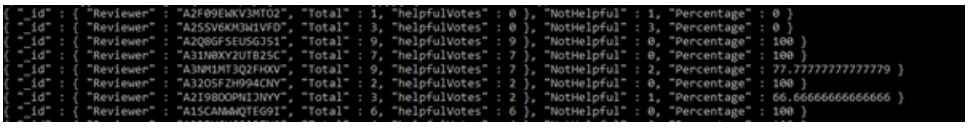
Wording of Query:

db.allreviews.aggregate([{$group:{"_id":{"Reviewer":"$reviewerID","Total":{"$arrayElemAt": ["$helpful", 1]},"helpfulVotes":{"$arrayElemAt": ["$helpful", 0]}}}},{$project:{"Decimal":{ $cond: [{ $eq: ["$_id.Total", 0] }, 0,{$divide:["$_id.helpfulVotes","$_id.Total"]}]}}},{$project:{"NotHelpful":{$subtract:["$_id.Total","$_id.helpfulVotes"]},"Percentage":{$multiply:["$Decimal",100]}}}],{allowDiskUse:true})

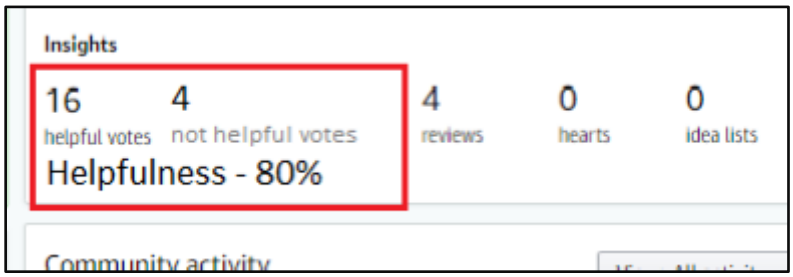
Summary:

We have calculated the total helpful, total not helpful and total votes received by every reviewer. Then we have divided the total helpful votes by total votes received and multiplied it by 100 to obtain the percentage. If any reviewer received 0 total votes, we excluded those reviewers to avoid divide by zero error and assigned zero as the percentage of helpfulness for those reviewers.

Screenshot from MongoDB:



Screenshot from our proposed Amazon webpage:



HELPFULNESS OF REVIEWS

Text based definition:

We have come up with our own definition of recommended reviews. We have defined a list of words and searched for those words in all the reviews for a particular product. Wherever we found a strong match based on high metascore, we considered those reviews as helpful. The reviews with lower metascores are considered as not helpful. We feel this will this will be a helpful criterion for the users to judge the quality of a product by taking into account the experience of other reviewers. We searched reviews with the following words: "good bad worse excellent price quality worst great delivery fast slow service time cost poor". While length can also be considered a criterion to define helpfulness of a review, we considered metascores as the parameter to define helpfulness of reviews. We have considered reviews having metascore greater than 5 as helpful.

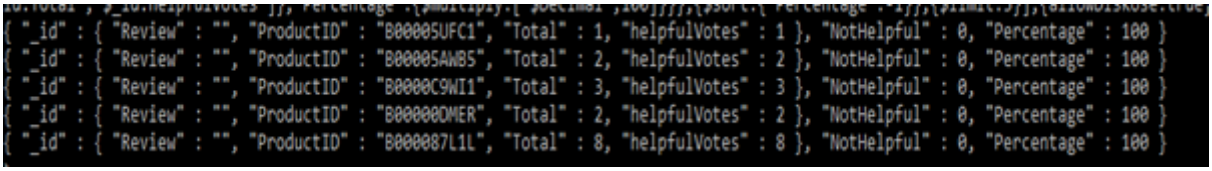
db.allreviews.find({ $text: { $search: "good bad worse excellent price quality worst great delivery fast slow service time cost poor" } },{ "_id":0,"asin":1,score: { $meta: "textScore" } }).sort({ score: { $meta: "textScore" } }).limit(5).pretty()

NOTE: For ease of comparison with the non-text-based approach, we have displayed only the top 5 products having the most helpful reviews.

```
{ "asin" : "B001YTK3XK", "score" : 10.278137943352384 }
{ "asin" : "B004HO58VQ", "score" : 8.778493804372712 }
{ "asin" : "B00CMQTVUA", "score" : 8.5383478303362 }
{ "asin" : "B00AVWKUWU", "score" : 8.533585379464286 }
{ "asin" : "B003YL412U", "score" : 8.388176037901129 }
>
```

Non-Text based definition:

In Amazon, every review gets helpful votes. We have calculated the percentage of helpfulness of each review. We consider percentage of helpfulness as a criterion for judging helpfulness of a review because there might be cases when a review has got 100 votes out of which there are only 5 helpful votes. So, the percentage of helpfulness is only 5 %, while there can be a review which received totally 5 votes out of which all 5 are Helpful. So, this review has 100% helpfulness.



From the above 2 screenshots, we can see that the results obtained from text based and non-text-based approaches do not match. This is because in the text-based approach, we have considered our own defined set of words as a measure of helpfulness. In the non-text-based approach, we have taken helpful votes as the measure of helpfulness. As helpfulness varies according to users' perception, therefore, the two approaches will result in different outcomes.

NEO4J – REVIEWERS REVIEWING COMMON SET OF PRODUCTS

Wording of query

```
MATCH (reviewer:AReviewer)<-[:WRITTEN_BY]-(:AReview)<-[:HAS_REVIEW]-(product:Product)
WITH reviewer, COLLECT(product.asin) AS commonProducts
WITH commonProducts, COLLECT(reviewer.reviewerID) as commonReviewers
WHERE SIZE(commonReviewers) > 1
RETURN commonProducts, commonReviewers
```

Snapshot showing successful execution

Summary of results

Out of all the paths in our graph model, we have matched the ones where we have a review being written by a reviewer and the product to which that review belongs. After that, given a reviewer, we have collected the list of products they have reviewed. Next, using this collection of products,

we have further collected the list of reviewers who have written these reviews. Then we have applied a condition that the size of the reviewers collection should be greater than one since we want to find the set of products being reviewed by a common set of reviewers and not just one reviewer. Finally, we return/ display the collections/ lists of the reviewers and the lists of products that they reviewed.

SAS – IDENTIFYING FAKE REVIEWS

SAS Text mining:

Issue of fake reviews:

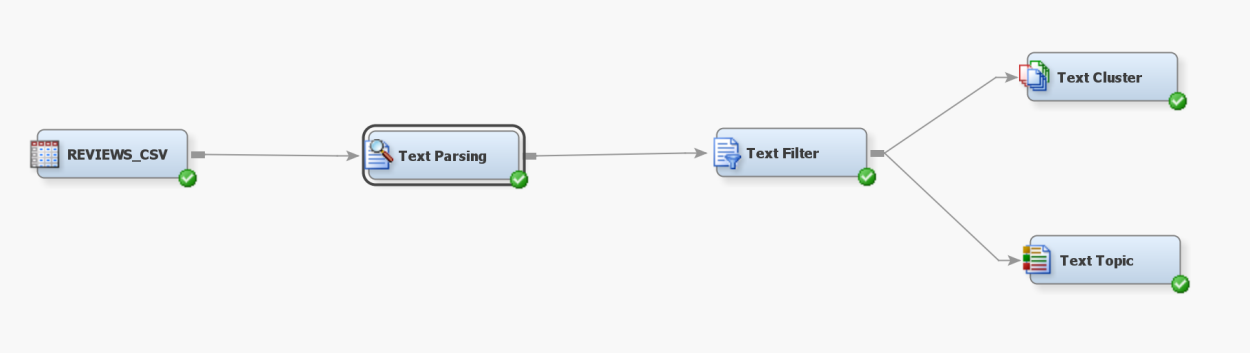
As the review systems are being extensively used by all the online businesses to provide the users best experience possible, some businesses, sellers and vendors of services and products are taking advantage of this system. In our analysis of the amazon review system, reportedly several sellers have been gaining fake positive reviews by offering free products and discounts to the customers. There are also cases where businesses target competitions and pay customers to provide fake negative reviews to their products. This is deteriorating the entire usability of the review system, providing bad experiences to users which, when reviewed are getting lost in the fraudulent reviews of that product. Reportedly, almost 100 facebook groups were found split up in several geographic locations and by categories, where merchants are continuously asking customers to provide fraud reviews.

To analyze the fake reviews in some of the amazon's reviews, we chose to use some extreme emotion carrying adjectives like excellent, worst, etc. Our explanation of doing this is that fake reviews generally tend to lack details of the product but just use words like this to represent the positiveness and high ratings. To implement our idea, we have converted the CSV file into a SAS table using SAS Enterprise Guide 7.1. In SAS Enterprise, we created a library and data source for this file and imported the table.

Data Dictionary:

id	ID	Nominal	No		No	.	.
asin	Input	Nominal	No		No	.	.
helpful	Input	Nominal	No		No	.	.
overall	Input	Interval	No		No	.	.
reviewText	Text	Nominal	No		No	.	.
reviewTime	Input	Interval	No		No	.	.
reviewerID	Input	Nominal	No		No	.	.
summary	Text	Nominal	No		No	.	.
unixReview	Input	Interval	No		No	.	.

Methodology:



To implement our idea, we have converted the CSV file into a SAS table using SAS Enterprise Guide 7.1. In SAS Enterprise, we created a library and data source for this file and imported the table.

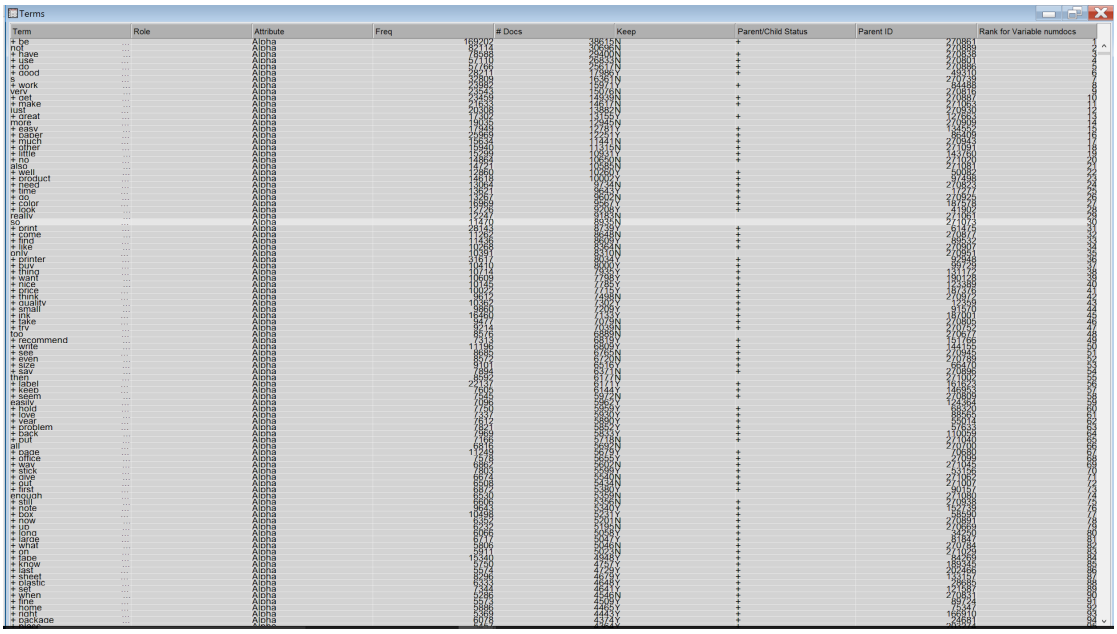
Text Parsing:

After we imported the data, we use the text parsing node to clean the data and make some necessary modifications for further processing. We used the properties panel of the text parsing node to:

1. Set the 'find entities' option to standard.
2. Set 'Detect parts of speech' to NO.
3. Ignore all the entities except Miscellaneous proper nouns (prop_misc).
4. Ignore all the parts of speech except adjectives.
5. Ignore numbers and punctuations.

After running the text parsing node, we generated the list of terms with their frequencies, to look and understand how it can be better processed by identifying the terms that are rarely used.

Output of text parsing node:

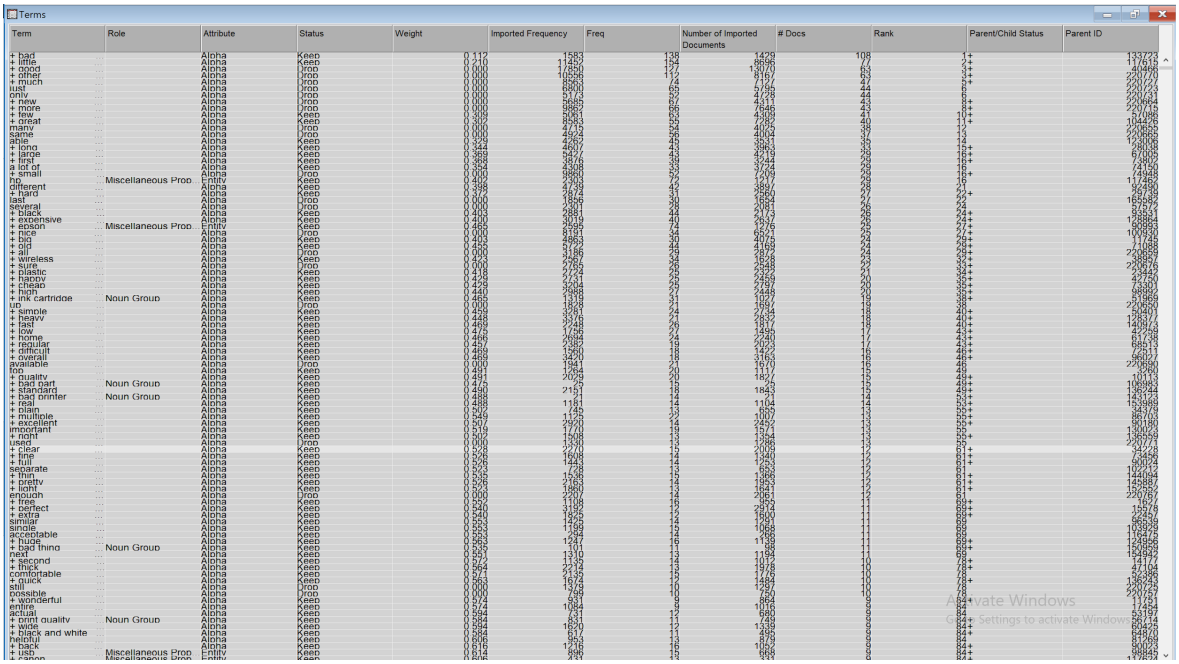


Text filter:

We added a text filter node, to drop the terms that have less weight, by entering the number of documents that word should be present in order to be considered. We used the 'filter viewer' in the properties of text filter node to keep words that would best fit for this analysis manually and combined some words and grouped them as synonyms. We performed spell check by selecting the 'check spelling' option as yes from the properties.

We also performed concept linking on some of the words, just to see if we can analyze something from the data. The outputs of the text filtering node and the concept linking are as below.

Text Filtering Output:



Manually keeping and dropping terms:

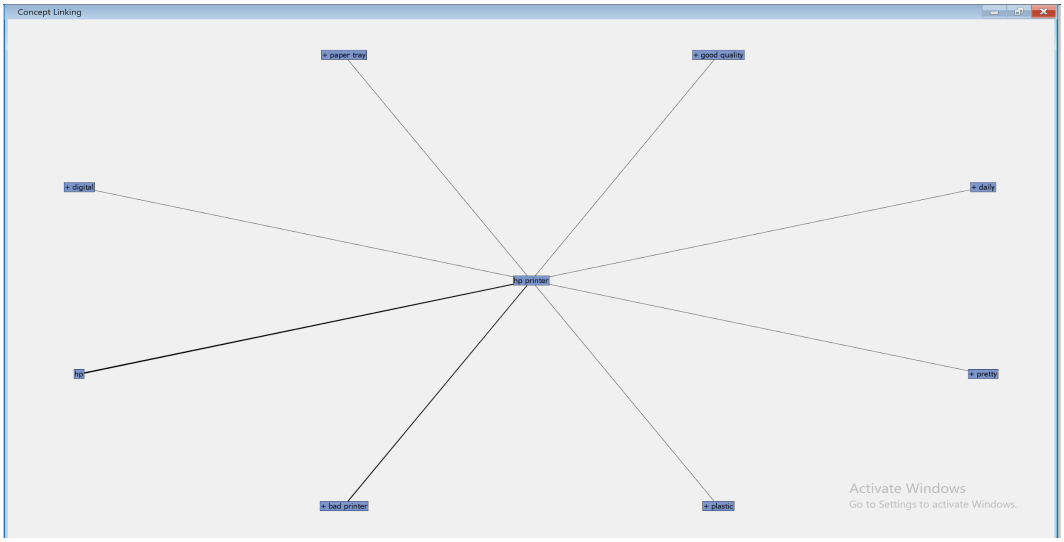
Terms

TERM	FREQ	# DOCS	KEEP	WEIGHT	ROLE	ATTRIBUTE
comparable printer						
expensive scanner	6	5	✓	0.854	Noun Group	Alpha
file storage	6	5		0.854	Noun Group	Alpha
inexpensive laser printer	6	5		0.854	Noun Group	Alpha
mock	6	5	✓	0.854		Alpha
excellent photo quality	6	5	✓	0.854	Noun Group	Alpha
mj	18	6		0.854	Miscellaneous ...	Entity
wf-7010	22	7		0.854	Miscellaneous ...	Entity
epson wp-4540	15	7		0.852	Miscellaneous ...	Entity
epson expression home xp-4...	9	6		0.852	Miscellaneous ...	Entity
lubricant	9	6		0.852		Alpha
introductory	9	6		0.852		Alpha
staple puller	9	6		0.852	Noun Group	Alpha
mp990	9	6		0.852	Miscellaneous ...	Entity
acroball	20	6		0.852	Miscellaneous ...	Entity
old at&t	15	6		0.852	Noun Group	Mixed
nx625	17	8		0.852	Miscellaneous ...	Entity
transitional notebook paper	12	7		0.851	Noun Group	Alpha
firmware version	11	6		0.851	Noun Group	Alpha
black leatherette	11	6		0.851	Noun Group	Alpha
claria ink	11	6		0.851	Miscellaneous ...	Entity
hl-5370dw	11	6		0.851	Miscellaneous ...	Entity
dear	15	8		0.851		Alpha
cl-241xl	15	6		0.851	Miscellaneous ...	Entity
metal arm	13	6		0.85	Noun Group	Alpha
battery pack	10	6		0.85	Noun Group	Alpha
workforce series	10	6		0.85	Miscellaneous ...	Entity
w x d	10	6		0.85	Miscellaneous ...	Entity
orange light	10	6		0.85	Noun Group	Alpha
unbroken	5	5		0.85		Alpha
avery ticket	5	5		0.85	Noun Group	Alpha
stress test	5	5		0.85	Noun Group	Alpha
everyday usage	5	5		0.85	Noun Group	Alpha
cheap kind	5	5	✓	0.85	Noun Group	Alpha
printer unit	5	5	✓	0.85	Noun Group	Alpha
fairly long time	5	5	✓	0.85	Noun Group	Alpha

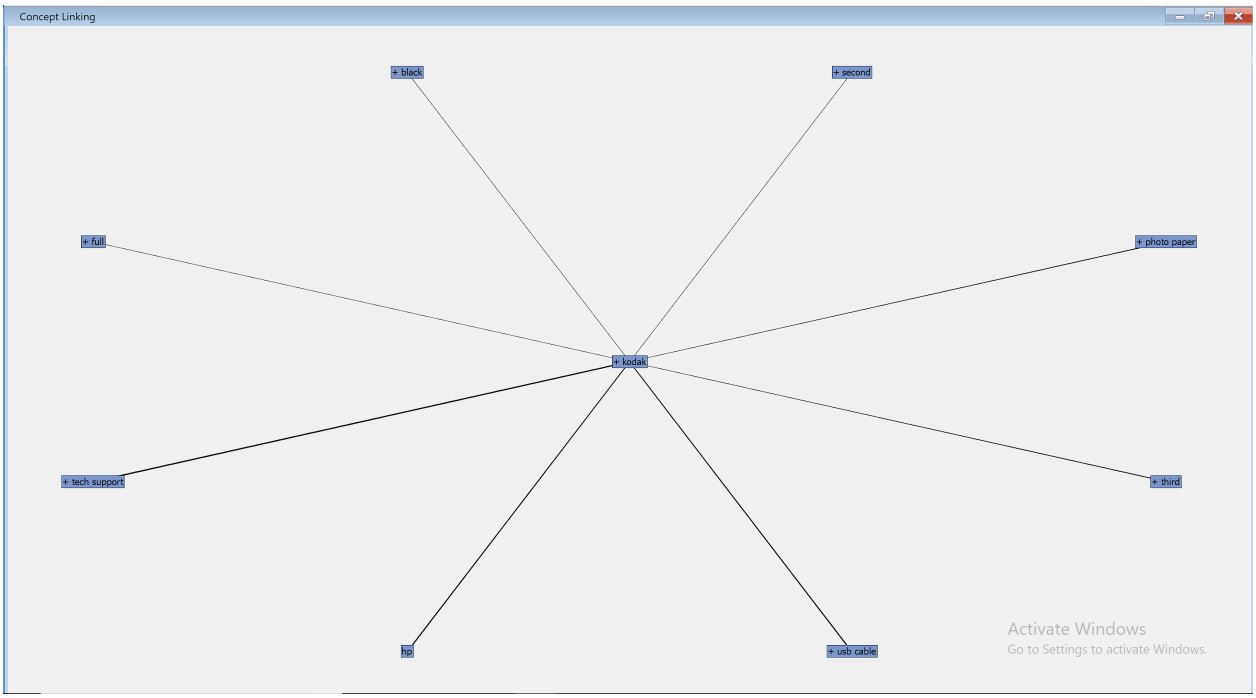
Concept linking:

We performed concept linking on the word ‘HP Printer’ and it shows that there’s a darker line towards the words ‘bad printer’ and ‘plastic’. We found this analysis interesting and performed this on other printer brands Epson and Kodak and inferred that epson had more stronger connection to words ‘wireless’ and ‘automatic’, indicating its features. Kodak had stronger connection to the word ‘tech support’, from which we inferred that the product might have more issues.

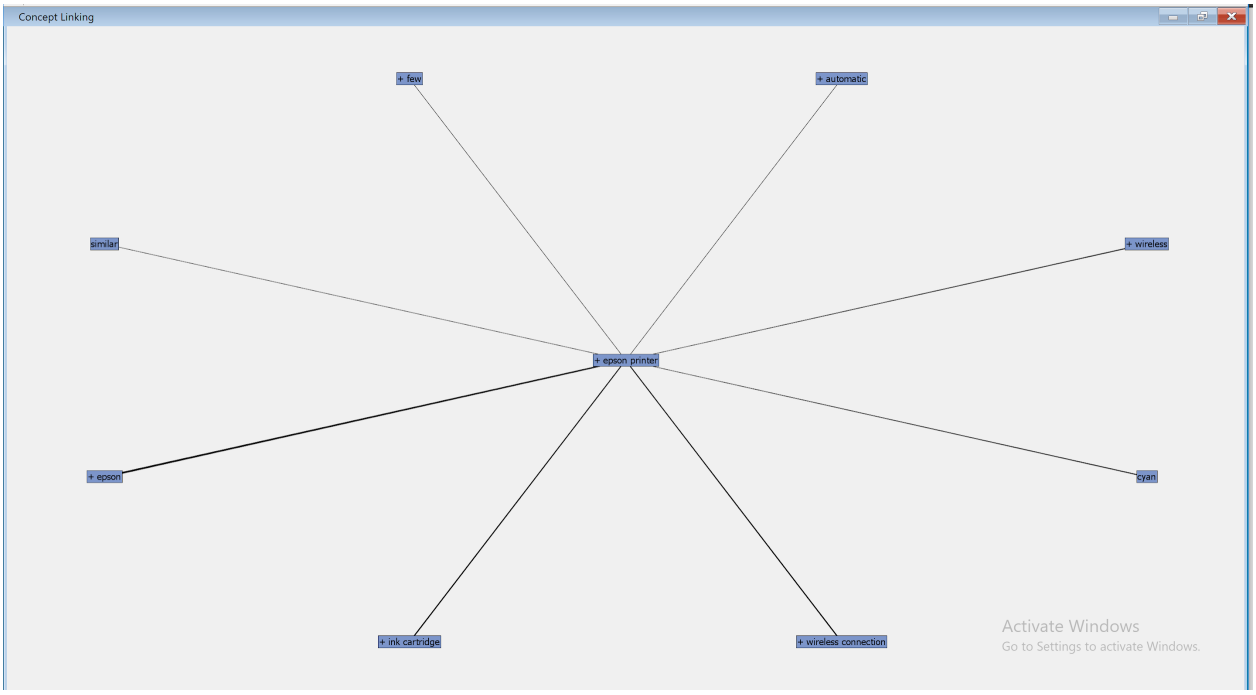
Concept Linking for HP:



Concept Linking for KODAK:

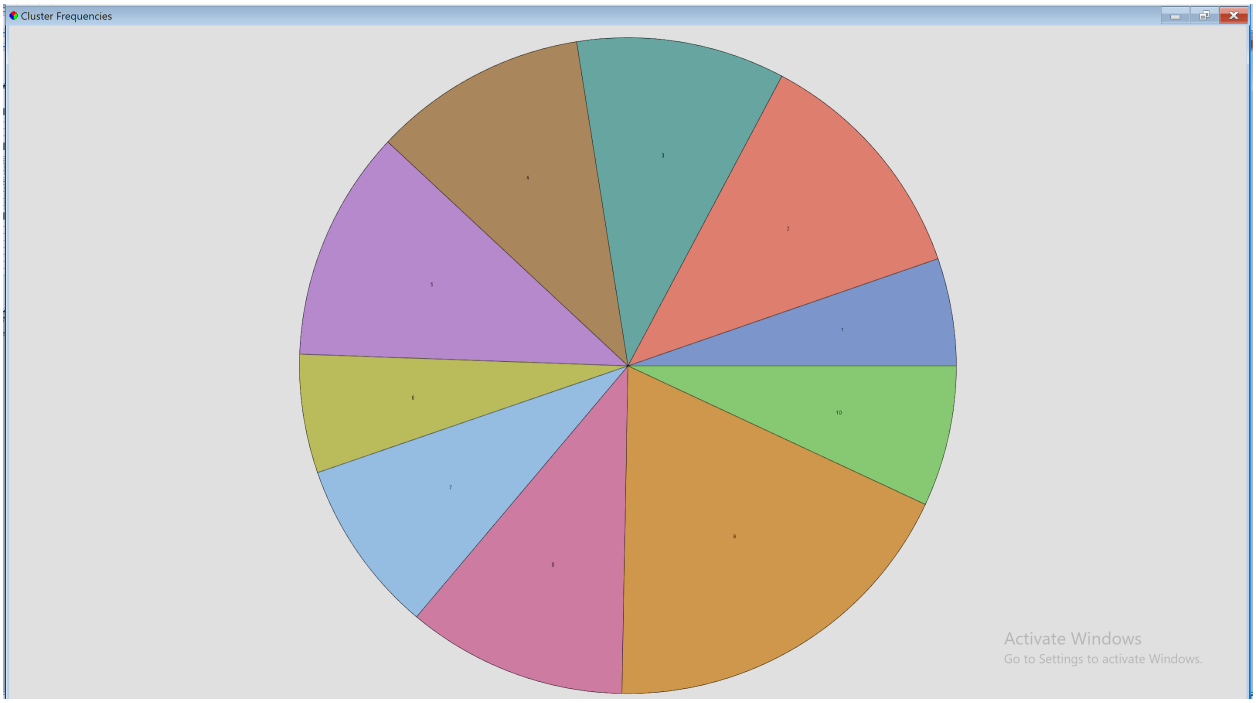


Concept linking for EPSON:



Text Clustering:

We attached a text clustering node to display the text in clusters to better understand the trend. We used the properties panel of the text clustering node to generate exactly 10 clusters. The outputs of this node are as below.



Clusters

Cluster ID	Descriptive Terms	Frequency	Percentage	Coordinate 1	Coordinate 2	Coordinate 3	Coordinate 4	Coordinate 5	Coordinate 6	Coordinate 7	Coordinate 8	Coordinate 9	Coordinate 10	Coordinate 11	Coordinate 12	Coordinate 13	Coordinate 14	Coordinate 15	Coordinate 16	Coordinate 17	Coordinate 18	Coordinate 19	Coordinate 20	Coordinate 21	Coordinate 22	Coordinate 23	Coordinate 24	Coordinate 25	Coo
1	+sticky +blue +scotch +dull +general +special +cutting worth +en	9	0.25	-0.0	-0.0	-0.0	0.10	-0.3	-0.3	-0.0	-0.0	-0.0	0.11	0.06	-0.0	0.03	-0.0	-0.0	0.10	-0.0	-0.0	-0.0	0.01	0.07	0.15	0.09	-0.0	-0
2	+good quality +battery +digital +daily +hp printer +entire +bad pri	21	0.34	-0.0	-0.0	-0.0	0.07	-0.0	0.05	-0.0	0.04	0.00	0.11	0.14	0.29	-0.0	-0	0.03	-0.0	0.12	-0.0	0.06	-0.0	-0.1	-0.0	-0.0	-0.0	-0
3	+writing +plastic +front +hard +large +standard +aware +favorite	18	0.30	-0.0	-0.0	0.08	-0.1	0.07	-0.0	-0.0	-0.0	-0.0	0.19	0.04	0.13	0.14	0.01	0.17	-0.2	0.15	-0.0	-0.0	0.11	-0.0	0.03	0.12	0.10	0.2
4	+cyan +epson +epson printer +impossible +ink cartridge +wireless	18	0.46	0.24	-0.0	-0.0	-0.1	0.02	0.12	-0.0	-0.1	-0.0	0.00	0.00	0.03	-0.1	0.03	0.06	-0.0	0.08	0.06	0.05	-0.0	-0.0	0.07	-0.0	-0.0	0.0
5	+slow +kodak +single +important +convenient +top +obvious +thin +	20	0.36	0.00	0.00	0.14	0.00	0.06	0.03	0.05	-0.1	0.04	0.03	0.04	-0.0	-0.0	-0.1	-0.0	-0.1	-0.1	-0.1	0.04	-0.0	-0.0	0.02	-0.0	0.0	0.0
6	+duplex print +duplex dual well +black and white +automatic +	10	0.47	-0.0	-0.0	-0.3	0.01	-0.0	-0.0	0.13	0.04	0.14	-0.0	-0.0	0.07	0.00	0.01	-0.0	0.04	-0.0	-0.0	0.01	0.03	-0.0	-0.0	0.00	0.0	
7	+cardboard +laser printer +lightweight +compact +dark +thick +'	15	0.34	-0.0	-0.0	0.1	-0.0	0.21	0.05	-0.0	0.07	0.15	0.15	-0.0	0.29	-0.1	0.04	-0.0	0.00	0.00	0.03	0.02	-0.1	0.20	-0.0	-0.1	0.0	
8	no +tech support' technical +fast +plain paper' +multiple +plain	19	0.43	0.15	-0.2	0.13	0.21	-0.0	0.01	-0.1	0.19	-0.0	0.08	0.10	0.04	0.09	-0.0	0.06	-0.1	0.04	-0.0	0.01	000	0.02	-0.0	0.04	-0.0	
9	comfortable +adjustable +long +wonderful +'long time' +little +de	32	0.35	-0.0	-0.1	0.08	-0.1	-0.0	-0.1	-0.1	-0.1	0.11	-0.0	0.04	0.15	0.02	0.06	-0.1	0.09	0.03	-0.0	-0.2	-0.0	0.06	-0.1	-0.0	0.0	
10	+free +hot +online total +regular +favorite local +'print head' +p	12	0.35	-0.0	-00	-0.0	-0.0	0.00	-0.0	-0.1	0.03	0.02	0.07	-0.0	0.05	-0.1	0.08	-0.0	0.03	-0.2	0.26	-0.0	0.27	-0.1	0.04	-0.1	0.1	

Text Topic:

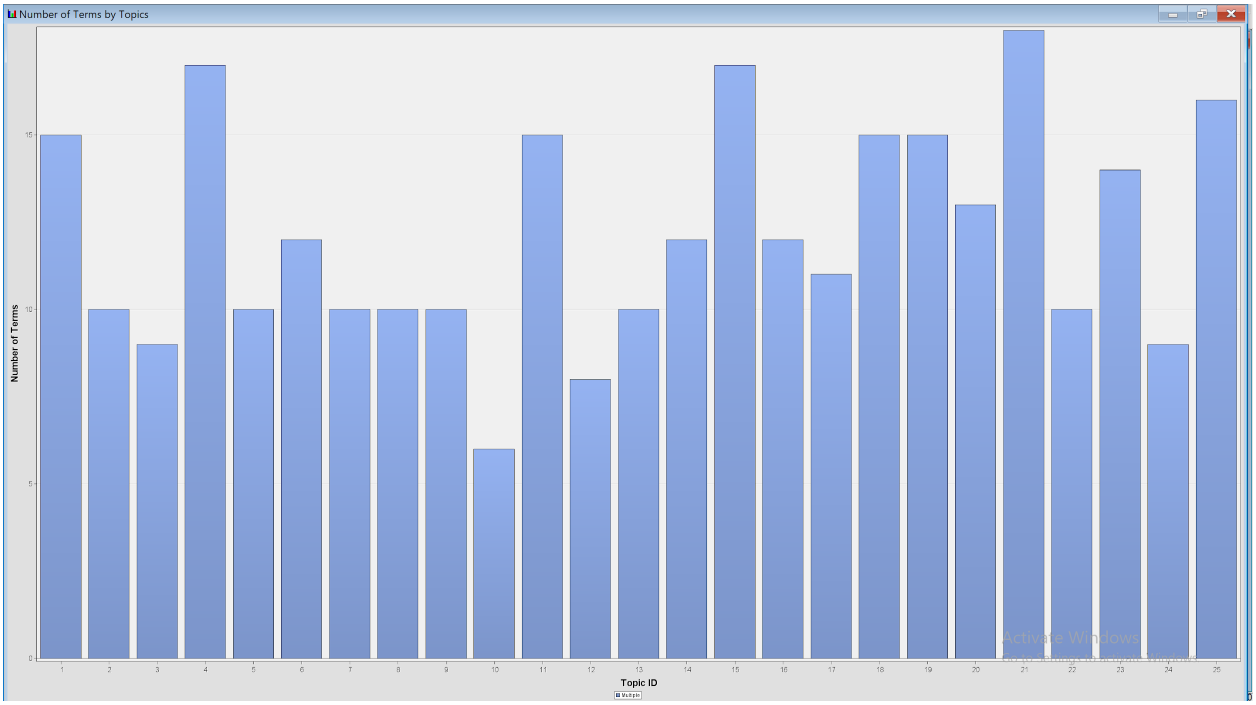
We used text topic node to combine words into topics that can help analyze the data further. We used the outputs to plot graphs between the topics and the documents, topics and terms.

Topics:

Topics

Category	Topic ID	Document Cutoff	Term Cutoff	Topic	Number of Terms	# Docs
Multiple	1	0.152	0.118	+low +heavy +dark actual +soft	15	17
Multiple	2	0.164	0.118	technical +tech support +multiple h	10	18
Multiple	3	0.172	0.112	+sticky +scotch +blue worth +second	9	12
Multiple	4	0.142	0.119	previous important +main +commo	17	18
Multiple	5	0.143	0.117	+duplex +duplex print +automatic +	10	8
Multiple	6	0.169	0.117	+pretty +good quality +digital +dail	17	11
Multiple	7	0.165	0.114	+laser printer +cardboard +tax mac	10	8
Multiple	8	0.144	0.117	+free +cheap +favorite +wonderful	10	12
Multiple	9	0.132	0.119	top +bottom +cardboard running +	10	20
Multiple	10	0.141	0.111	+writing traditional usual +fantastic	6	7
Multiple	11	0.150	0.118	+ink cartridge +epson printer single	15	19
Multiple	12	0.168	0.116	+hard +pretty +soft +many +negative	8	22
Multiple	13	0.165	0.114	+battery +cutting entire +dull +little	10	13
Multiple	14	0.147	0.118	+adjustable +wonderful +perfect +l	17	15
Multiple	15	0.149	0.119	+expensive impossible total +hand	17	22
Multiple	16	0.134	0.119	obvious +full +original +usb cable +	12	20
Multiple	17	0.136	0.118	+slow +real +cheap separate +front	11	18
Multiple	18	0.130	0.119	+inkjet +basic +pdf +crisp +lexmark	15	17
Multiple	19	0.183	0.118	+bad part +little +bad +lightweight t	15	21
Multiple	20	0.124	0.119	+bad thing +second +tall +huge +u	13	19
Multiple	21	0.135	0.119	+print head +compatible +head +te	18	15
Multiple	22	0.153	0.118	+back +tall +front +many +adjustable	10	15
Multiple	23	0.146	0.118	+scotch +difficult +desk +huge +ba	14	18
Multiple	24	0.183	0.117	hp hp printer +bad printer +ink cart	9	25
Multiple	25	0.185	0.118	+lightweight +great important +che	16	18

Terms by Topics:



Terms by Documents: