Project Report

Name	Email	NUID	Section
Ankur Shanbhag	shanbhag.an@husky.neu.edu	001731550	13250
Hardik Shah	shah.hard@husky.neu.edu	001744824	13250
Rahul Thakkar	thakkar.ra@husky.neu.edu	001743754	15702
Vardhman Jain	jain.var@husky.neu.edu	001728511	13250

Introduction and Dataset -:

In this project we are analysing the Amazon Reviews dataset to extract interesting information and run analytics on them. We have distributed the analysis over 4 major tasks of Join preprocessing, top-K data, disjoint graph sets between different categories and seasonal peaks from the user reviews. These tasks are explained in detail later.Our dataset size is ~65GB and in order to do preprocessing we implemented parallel job submission logic in order to pre process all the data within 15 minutes. Additionally, we made use of amazon s3 bucket sharing in order to share this data evenly amongst four buckets.

We are working with the Amazon review dataset. We have obtained the data set from the following link - http://jmcauley.ucsd.edu/data/amazon/links.html. This dataset contains reviews (ratings, text, helpfulness votes), product metadata (category information, price, brand) and links (also viewed/also bought graphs) of Amazon products, which spans to around 143.7 million records collected from May 1996 to July 2014. The data is in the JSON format.

Sample Data -:

Sample review -:


```
{
   "reviewerID":"A2SUAM1J3GNN3B",
   "asin":"0000013714",
   "reviewerName":"J. McDonald",
   "helpful":[2, 3],
   "reviewText":"I bought this for my husband who plays the piano. Great
purchase!",
   "overall":5.0,
   "summary":"Heavenly Highway Hymns",
   "unixReviewTime":1252800000,
   "reviewTime":"09 13, 2009"
}
```

Where -

- reviewerID - ID of the reviewer, e.g. A2SUAM1J3GNN3B
- asin - ID of the product, e.g. 0000013714
- reviewerName - name of the reviewer
- helpful - helpfulness rating of the review, e.g. 2/3
- reviewText - text of the review
- overall - rating of the product
- summary - summary of the review
- unixReviewTime - time of the review (Unix time)
- reviewTime - time of the review (raw)

Sample metadata -:

Metadata files include product descriptions, price, sales-rank, brand info, and co-purchasing links:


```
{
   "asin":"0000031852",
   "title":"Girls Ballet Tutu Zebra Hot Pink",
   "price":3.17,
   "imUrl":"http://ecx.images-amazon.com/images/I/51fAmVkTbyL._SY300_.jpg",
   "related":{
      "also_bought":["B00JHONN1S", "B002BZX8Z6"],
      "also_viewed":["B002BZX8Z6", "B00JHONN1S"],
      "bought_together":["B002BZX8Z6"]
   },
   "salesRank":{
      "Toys & Games":211836
   },
   "brand":"Coxlures",
   "categories":[["Sports & Outdoors", "Other Sports", "Dance"]]
}
```


Where -

- asin - ID of the product, e.g. 0000031852
- title - name of the product
- price - price in US dollars (at time of crawl)
- imUrl - url of the product image
- related - related products (also bought, also viewed, bought together, buy after viewing)
- salesRank - sales rank information
- brand - brand name
- categories - list of categories the product belongs to

In order to upload ~ 65GB of data on Amazon S3 we split the data into four parts of approximately 15GB each, uploaded on 4 different user buckets and gave each other read and write permissions using S3 bucket policies. These four user buckets were shared while running all the Hadoop tasks we have mentioned below.

Task Breakdown -:

Join (Preprocessing) -:

- We are interested in computing average of user rating for every product required for some of the task specified below.
- We have joined Metadata file (containing product information) with the Review file (containing user review information for products) based on the product id "asin" attribute to compute the average user rating for each product.
- We were able to achieve this in a single MapReduce program by taking advantages of MultipleInputs and GenericWritable classes. We have different Mappers for the Review data and Metadata. We used GenericWritable as a Mapper output values so that both of these Mappers can emit different type of Values: Text and a custom <double, int> pair writable for finding the average rating.
- We already had data segregated by the different categories so we submitted 24 MapReduce jobs: each per category to get even more parallelization and in turn avoiding extra calculation in sort and shuffle step of MapReduce.

Pseudo Code-:

ReviewJoin: ***edu.neu.mr.amazon.reviews***

```
ProductMetadataMapper (text):
        jsonObject = parse (text)
        replaceSpecChars (jsonObject)
        emit (jsonObject.asin, getText (jsonObject))

ProductRatingMapper (text):
        jsonObject = parse (text)
        emit (jsonObject.asin, <jsonObject.rating, 1>) // Used second parameter count=1 for combining results

ProductRatingAvgReducer (asin, iterable<genericWritable>):
        for each in iterable:
                if(genericWritable instaceof text)
                        metadata = (text)  genericWritable
                else
                        sum+= getSum (genericWritable)
                        count+= getCount (genericWritable)
        emit(asin, metadata | (sum/count))
```

- All the features required for tasks specified below has been extracted from this join output. This join output would be used further in two of the other major tasks listed below.
- The output directory names (e.g. "*category=Books*")are such that data can be easily partitioned for Hive external table creation.

- The sample for joined output is –

asin | avg_rating | review_count | category | price | also_viewed | bought_together | bought_after_viewing | brand | title

5555991584 | 4.6 | 130 | Digital Music | 9.49 | B000002LRR | B000002LRR | B002RV01QI,B000050XEI,B000002LRT | | | Memory of Trees

Job Statistics:

- 11 m3.xlarge machines for Joining 24 categories
- ~ 63 GB of input data Metadata and Review (split across 4 S3 buckets)
- ~ 2.2 GB of output data
- ~ 22min (1276s) for joining the Metadata and Reviews

m3.xlarge - 4 cores, 15GB RAM, 80 GB SSD

Top-K -:

- We created external table in Hive over the joined data computed above stored on S3 in a specific format (CSV or TSV).
- This table can now be queried to find the top-K products (top based on overall average review rating and the number reviews computed above) based on parameters like category, price and brand.
- Example – Sample HiveQL query to fetch top 100 Books sorted by review count and the first 5 rows of the output for the same

INSERT OVERWRITE DIRECTORY 's3://vardhman.jain/AmazonAnalytics/output-Hive-Books/" select concat(asin, "," , price,",", title, ",", avg_rating, ",", review_count) from (select asin, price, title, avg_rating, review_count from products where category='Books' distribute by review_count sort by review_count DESC limit 100) as s;

Sample Output-:

0439023483 , 4.99 , The Hunger Games (The Hunger Games Book 1) , 4.6 , 21400
0439023513 , 6.99 , Mockingjay (The Final Book of The Hunger Games) , 4.2 , 14114
0375831002 , 6.99 , The Book Thief , 4.6 , 12571
038536315X , 10.49 , Sycamore Row , 4.5 , 12564
0849922070 , 7.99 , Heaven is for Real Movie Edition , 4.5 , 10424

- The input size of ~240 MB signifies that partitioning reduced the data read considerably visible in the following log snippet taken from file *log*

*2015-12-07 23:50:07,797 INFO [main()]: exec.Utilities (Utilities.java:estimateNumberOfReducers(3126)) - BytesPerReducer=256000000 maxReducers=1009 totalInputFileSize=**252,586,462***

Job Statistics -:

- 4 m3.xlarge machines used to run the hive queries
- Time taken ~ 70 seconds

m3.xlarge - 4 cores, 15GB RAM, 80 GB SSD

Structures in Graph (Disjoint sets) -:

- We were interested in mining relations between different categories based on available product information. These relations can then be used to enhance the capability of recommendation system for the user.
- Every product in the metadata file has information about other products which were viewed or bought by the customers along with the product. This data is useful to understand information about user interest/preferences, correlations among the products and their demand. These relationships between products describe two important types of interest, namely those of substitute and compliment products. Substitute products are those which can be interchanged (like different pair of shoes) and compliments are those that might be purchased together (like a pair of shoes and socks). We can represent this information in form of a graph. Every product will form a vertex and all the viewed/bought products being represented as an adjacency list.
- Union-Find is an efficient algorithm which can be used to find disjoint sets of products from this graph representation. However, all the graph algorithms including Union-Find are iterative in nature and require thousands of chained MapReduce jobs to determine all the disjoint set of nodes from a large graph.
- Chaining many MapReduce jobs will result in long running program impacting overall performance of the algorithm.

Therefore we have used Apache Spark instead of plain Java MapReduce which has an advanced Directed Acyclic Graph (DAG) engine supporting cyclic data flow which is well suited for iterative algorithms given a sufficiently large size cluster to perform in-memory computations.

- Apache Spark has an inbuilt GraphX library suitable for performing iterative graph algorithms on large data sets. GraphX is a distributed graph processing framework on top of Spark. It provides an API for expressing graph computation which enable users to easily and interactively build, transform, and reason about graph structured data at scale. We have leveraged the GraphX implementation of Union-Find algorithm to determine connected components (products) over large data set.

The main challenge we faced was to represent our data in form of a graph which can then be used with GraphX API. This program was developed in Scala (which is compatible with Graphx) to represent every product as a vertex of the graph and edge providing information about products which were bought/viewed together.

- The calculated disjoint sets of related products using GraphX was followed by a MapReduce program which calculates the count of products belonging to each category in every disjoint set. Finally we mark categories with maximum counts in every cluster as correlated.

MapReduce job pseudo-code:

- Every map task will receive text record as input which holds informations about one product (along with its category) and an id number (indicating disjoint set).
- The map task will output data in following format:
  <Key: Disjoint set id> <Value: Product-category information>
- Reducer in its reduce() call will iterate over all the products for a disjoint set and maintain a HashMap<Category, Count>.as we have finite number of categories (only 24 categories).

- Finally we use MultipleOutputs to generate a separate partition file for all the disjoint sets. Reducer will output the data as follows:
  <Key: Category> <Value: Count>

We developed a single Java program which automated the process of calculating disjoint sets of products using Apache Spark and then invoke a MapReduce program to output categories

Finding seasonal (monthly) peaks -:

o We determined seasonal product popularity based on reviews density at different seasons of the year (months in our case).
o In order to do this we required two MapReduce jobs, which are mentioned as follows-
o The first MapReduce job reads the JSON input and converts it to CSV format and writes selected fields (<***asin,review_rating, unix_review_time***> in our case).The mapper reads the output and the final output file is generated by the reducer.
o We implemented another MapReduce job that did the following –

- The input to the mapper looks like <***asin, review_rating, unix_review_time***>
- Mapper filters out the reviews for the specified product for a range of years.
- The intermediate <key, value> generated by the mapper is

<***Key***:<***yyyymm, asin> Value: <review_rating_average, count***>>. The month is determined from the unix_review_time attribute in the Review record.

- Every reducer is responsible for calculating the average overall rating of the product for every month.

o We have implemented custom writable for map and reduce output and also custom practitioner that sends data for different months to different reducers.
o This module is invoked by user with range of years and product id "*asin"* specified as parameters to a java program.
o We finally generate twelve part files (one for each month) that contain the <<***Key***:***yyyymm>,<Value: average_rating,review_count***>> (average rating of month MM for all the years given in the range and number of reviews in that month).
o Additionally, for this task we wrote a sequential program that runs over the output files generated for the job and plots charts for each month of every year against its average rating and the count of reviews. This helped us in visualizing the trends from the data. We explored jFreeChart library in order to achieve this.

The source code corresponding to this module is as follows -
Package **edu.neu.mr.seasonalPeaks**

- SeasonalPeaksDriver.java
- SeasonalPeaksMapper.java
- SeasonalPeaksPartitioner.java
- SeasonalPeaksReducer.java

Package **edu.neu.mr.utils**

- DateProductWritable.java
- RatingCountWritable.java

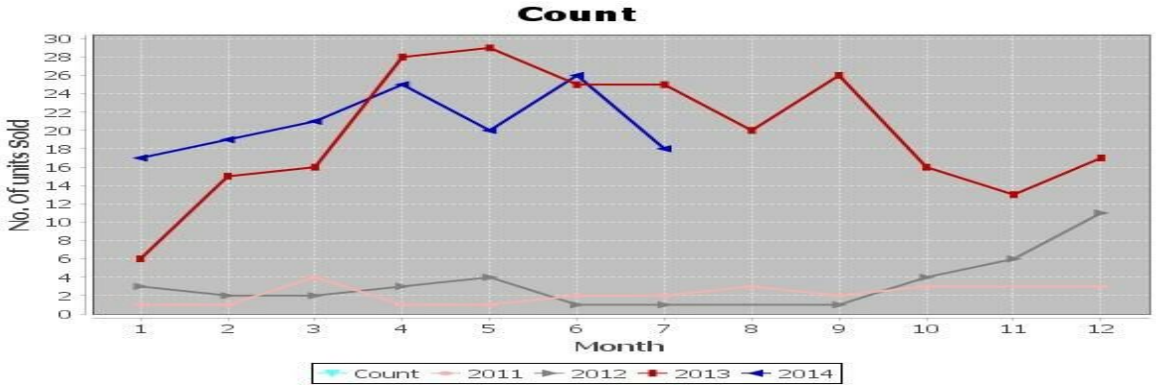
- JsonToCSVWithMEtadata.java
- Utility.java

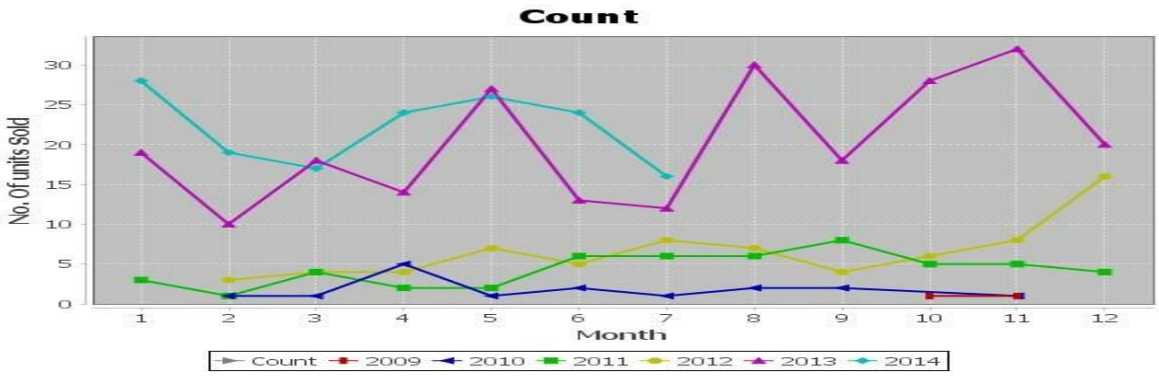
Case Study

- Seasonal peak of flu and allergy medicines comparison with google flu trends
- Harry Potter DVD sales peak comparison with DVD release date
- BBQ grill sales higher during the summer which is synchronous with BBQ trends among people.

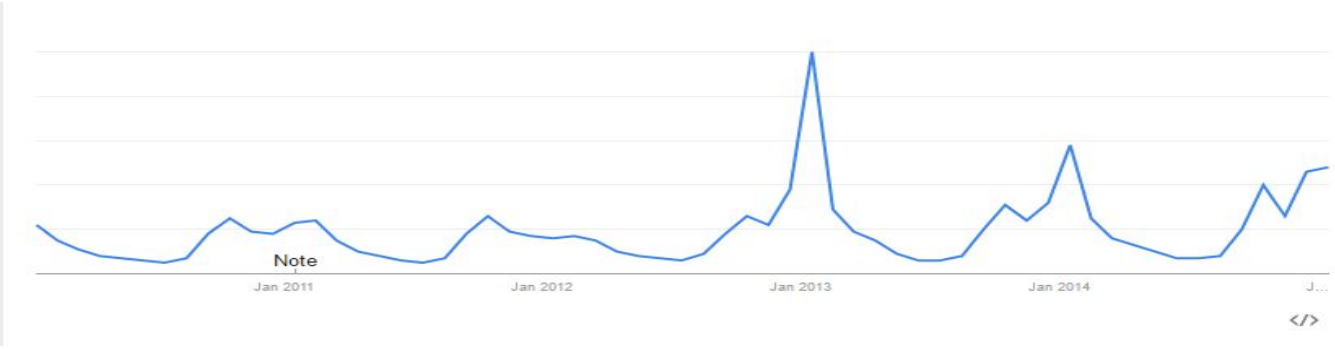
Flu and Cold Case Study

Citrazine, Oscillococcinum, Loraditine (flu medicines) sale shows rise for year 2013.



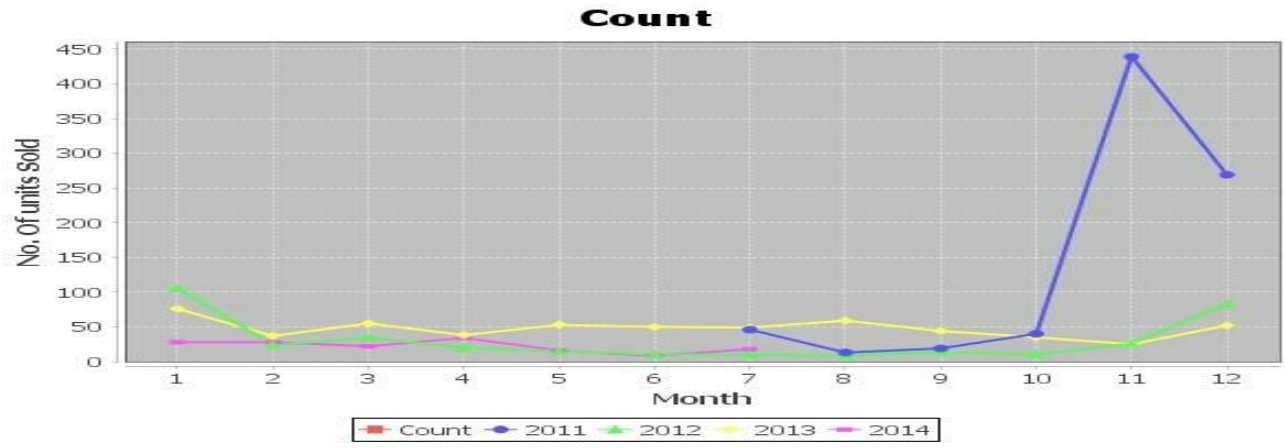


This matches with the fact that there was a sudden surge in the flu cases during the 2013-2014 duration (especially in the winter months - January to April). As seen in the following google trends graph.



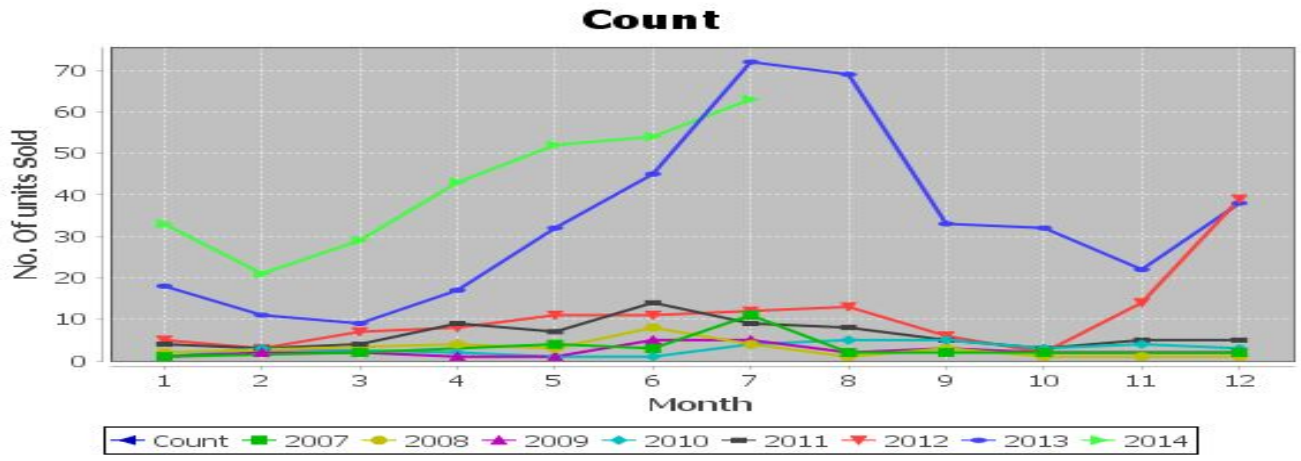
Movie DVD Releases

There is an obvious surge in sales of movie DVDs when they are released as seen in the following graph which show the sales surge for the Harry Potter movie DVDs immediately after they were released (November,2011) in the following case.



BBQ Grill Sales

BBQ grills tend to have a peak during the summer, which can be validated by the following graph



Job Statistics:

11 m3.xlarge machines for data conversion from JSON to CSV
~ 45 GB of input data (split across multiple 4 user buckets)
~ 2 GB of output data (refer log folder SeasonalPeaksPreprocessing)
10 minutes to process the data
3 m3.xlarge machines used to compute monthly averages
~ 2 GB of input data (on one s3 bucket)
~ 5 minutes to process data (refer log folder SeasonalPeaksProcessing)
Note: m3.xlarge Configuration - 4 cores, 15GB RAM, 80 GB SSD

Conclusion -:

We set out to analyze the review data and get some useful insight from it and make things like recommendation, demand based supply better and more efficient. This in turn has a direct impact on the user experience and consequently on the revenue of any eCommerce business. This project introduced interesting ways to improve the recommendation system by finding disjoint set in the also-bought network and by giving top-k products in the each category. The idea of finding seasonal peaks of a product over the years gives insight to what period that product is sold most which definitely helps better the solution to the demand-supply conundrum. The results we show above are promising and can be extended to ideas discussed in the future scope.

Future Scope -:

The dataset includes further information about the product like its sales rank within the category and the exact review by the user which we are not currently using in the tasks we have implemented. We could use this data to further do some interesting analytics that are mentioned below-:

- **Sentiment Analysis on the exact user review** -
    - We could extract sentiments from the user reviews to analyse what the user thinks about the product or service.
    - We can associate certain keywords inside the review with the overall rating and after considering all the reviews for the product we could tag the products with most frequently occurring keywords in the review. This can help us in finding overall emotion quotient for the product and can help in searching these keywords, for instance “excellent” , “disappointing” etc.
- **Improvement of Recommendation System -**
    - We can extend the disjoint graph sets task by using other information like sales rank and user ratings to improve the overall user based recommendation system (by using collaborative filtering). Recommendation systems are one of the most key features today in an eCommerce business that can help in improving sales and revenues by manifolds.

Citations -:

· Image-based recommendations on styles and substitutes
J. McAuley, C. Targett, J. Shi, A. van den Hengel
SIGIR, 2015
pdf

· Inferring networks of substitutable and complementary products
J. McAuley, R. Pandey, J. Leskovec
Knowledge Discovery and Data Mining, 2015
pdf

· “Granting Read-Only Permission to an Anonymous User”