FINE-TUNED RECOMMENDER SYSTEM ON AMAZON REVIEWS DATASET

Aritra Majumdar
aritram21@vt.edu
Virginia Tech

Rhea Saxena
rhea2809@vt.edu
Virginia Tech

Romil Khimraj Balar
romilkhimraj@vt.edu
Virginia Tech

ABSTRACT

There exists a “similar product” section for each amazon product. There also exists multiple reviews for each product on amazon. Currently, we cannot assume that the similar or “recommended” product is one that is the best for the job/ reviewed well.
We wish to provide “filtered” recommendations based on our analysis of the reviews under that product. Thus, from the list of products most similar to, or frequently bought along with a given product, we can filter out the ones that are truly well-reviewed.
What we could see through our preliminary analysis was that the products suggested would not always be the “best” in their category, therefore by combining product relationship details and review information, we could generate better recommendations.

1. INTRODUCTION

The Internet has revolutionized the shopping experience, and the pandemic has further solidified the need for ecommerce. We no longer need to walk down aisles searching for the products we need, we only need to type a couple of words and we are presented with more choices than we would ever need. When we walk into a store all products of the same type/ used together and their variations will often be found together and sectioned accordingly, this gives a buyer all the options in one place and also urges them to consider other similar/ complementary products, this task is often performed by recommender systems in ecommerce. Recommender systems help avoid the need to keep scrolling to long lists of the same item and emulate the experience of looking around a section in a store, by recommending similar products to a user.

However, when we can’t hold products to gauge their quality and usefulness we often depend on others' reviews to make the right choice. But, most recommender systems, specifically retail based commerce give equal weightage to click traffic, compared to user reviews, if not more. This while boosting the seller’s agenda is not very consumer friendly. This need to also give quality to recommendations has motivated our work.

In our work we use a dataset of amazon reviews,discussed further in 4.1 , which has metadata that have fields called also_buy and also_view. Based on research of how most ecommerce systems work [1][2], it is safe to assume that the above mentioned fields can be used as the basis of creating a similar/ recommended product network. We use this as the base of our work and further implement algorithms to fine-tune these similar product recommendations, such that our final recommendations not only account for the ‘click traffic’ needed and various other parameters that go into a large scale recommendation engine, like Amazon’s, but also give high priority to the individual reviews for each product. Our goal is for our model to give quality recommendations which will help users trust the recommendations, as the recommended product would have meaningful positive reviews and help the click turn into a buy without wasting the consumers time.

2. BACKGROUND/RELATED WORK

Many previous works [3][4] have used the centrality measures in a network to make recommendations, degree centrality is based on the number of links held by each node.It shows how many direct connections each node has to other nodes. It is useful when identifying connected nodes which may hold the most information, which is why it is a good measure to build recommender systems. Degree centrality specifically is also appealing as it is not computationally as expensive as the other measures and can help the speed of recommendations.

In our work, we structure the sentiment analysis as an unsupervised learning problem and choose VADER[5] which is a valence-based lexicon based sentiment analysis algorithm that is capable of detecting both intensity and polarity aspects of sentiments. It uses an expansive lexicon (~7500 features) , linguistic rules and rule-based modifiers to generate sentiment scores. It works like a dictionary that assigns a predetermined sentiment score between -4 to 4 to each feature. It outputs a compound score as the "normalized, weighted composite score" which summarizes the sentiment intensity. It also provides pos, neg, neu scores which represent the ratios of proportions of text that fall in each category. For our work we choose to work with the compound score as it provides an overall view.

3. APPROACH

Every unique ID of the product (ASIN) will be treated as a node in a network. Using the related products field, similar products that have been bought or viewed along with the current product will be connected using edges. Every edge will consist of weights that represent the priority for recommendation. When a particular node in the network is accessed (i.e., product is bought or viewed), nodes that are directly connected to the particular node will serve as a recommendation list, and a product will be recommended based on the weight of the edges. To provide quality recommendations, for every unique product, we will do a sentiment analysis of the nodes in the recommendation list and create a probability list based on the overall sentiment of each product, where the nodes (products) which are considered the best have the highest probability of recommendation. Therefore, if a node is accessed, the top three products will be recommended.

4. EXPERIMENT

This section is structured, to go over in detail our experiment. We systematically go over our data collection strategies, implementation (consisting of building the product network, network analysis and sentiment analysis) and our overall analysis.

4.1. Data Collection

For this Project we utilize the UCSD Amazon Review Data (2018) [6]. This is an updated version of 2013 dataset [7] and includes reviews from May 1996 to Oct 2018.The original raw data set is quite large with 233.1 million reviews (34 GB), hence, we plan to use Pre-category data in which the dataset is broken into multiple categories and reviews and metadata for each are provided. We will be using the Electronics category as in current trends, we find multiple duplicates/ fakes of multiple electronic products which often turn out to be a waste of money. Effectively, our dataset will consist of metadata of 786,868 products and 20,994,353 reviews.
The metadata is a total of 14 fields:

- **asin** - ID of the product
- **title** - name of the product
- **feature** - bullet-point format features of the product
- **description** - description of the product
- **price** - price in US dollars
- **imageURL** - url of the product image
- **imageURL** - url of the high resolution product image
- **related** - related products (also bought, also viewed)
- **salesRank** - sales rank information
- **brand** - brand name

- **categories** - list of categories the product belongs to
- **tech1** - the first technical detail table of the product
- **tech2** - the second technical detail table of the product
- **similar** - similar product table

We plan to form our network based on the related products field, similar to the stanford dataset [8] of amazon metadata.
The reviews data has fields:

- **reviewerID** - ID of the reviewer
- **asin** - ID of the product
- **reviewerName** - name of the reviewer
- **vote** - helpful votes of the review
- **style** - a dictionary of the product metadata
- **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)
- **image** - images that users post after they have received the product

We use the review data, to provide only the top products based on overall reviews and their helpfulness scores. Due to duplicates and ease of computation, we kept the first instance of each product, leading to our final network having 82394 nodes and 325,137 edges.

4.2 Implementation

4.2.1 Building the product network

The metadata information is used for building our product network.
The metadata was found to have duplicates, hence, we use a list of unique product asin (unique reference numbers for each product) as our nodes.
The dataset as mentioned above has two fields 'also_buy' and 'also_veiw' that were used to create network edges. Each of these fields is a list of other product asins that were also bought/ also viewed when a given product was viewed.

- edges from each of the network nodes (product asins) to each of the nodes (product asins) listed as also_buy are created, with weight 1.
- edges from each of the network nodes (product asins) to each of the nodes (product asins) listed as also_view are created, with weight 2.

The edge weights show priority, the smaller the edge weight the better. As mentioned in the previous section our final product network has 82394 nodes and 325,137 edges.

4.2.2. Network Analysis

A preliminary analysis of our network reveals that the density of the graph is very low (9.57×10^{-5}) and the number of connected components is very high (11535). This implies that the nodes in the network exist in small clusters of groups. An analysis of the degree centrality in the network is provided in Fig 1, which shows that the average degree is extremely low, with the highest degree being 0.0142. A modularity score of 0.846 corroborated the presence of smaller clusters of nodes. As a result, it was postulated that the best way to assign a score to each of the nodes for the purposes of isolating important nodes from comparatively unimportant nodes was to use the degree centrality measure.

	Degree_Centrality
count	82394.000000
mean	0.000096
std	0.000281
min	0.000000
25%	0.000012
50%	0.000012
75%	0.000061
max	0.014200

Fig 1: An overview of the degree centrality of the network.

4.2.3 Building Recommendation System

Once the product network is established, for every node in the network, a degree centrality score is calculated. The degree centrality can be a measure of the amount of traffic that the node receives as it is the measure of how many edges a node is attached to. In our case, the traffic for a node can be considered as customers that are viewing or buying a product. A higher degree centrality means that the product has been bought along with a lot of different products or viewed every time after a customer buys a different product. In that case, it would imply that the product is essential in its own regard as it seems to have been bought or viewed related to other products a lot of times. Therefore, it was chosen to score the nodes based on the degree centrality score. Inspiration to use degree centrality as a scoring measure was drawn from the work of Waheed et al. 2019 as they have used this measure to recommend research papers [6].

Along with calculating the degree centrality score, we have to also keep in mind that if a product is also bought with another product, or if it is viewed after buying another product, the weights between the edges for these relations will have to be different as each scenario has a different importance. A product that is viewed after buying another product intuitively seems less important than a product that was also bought with the product in question. Therefore, the final score for all the products were calculated by dividing the individual centrality scores with edge weights as the edge weights were inversely proportional to the importance of the product. These products and scores were stored in a recommendations dictionary such that all the products were keys and its corresponding values were a python dataframe containing the five product IDs with top five final scores. If a product did not have any related products then only a string explaining that there were no related products was stored. Figs 2 and 3 demonstrates the algorithm for convenience.

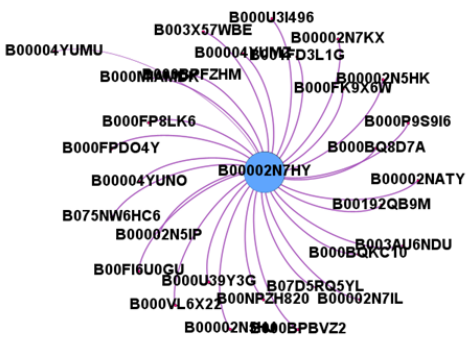

Fig 2: Network of queried products and its related products.

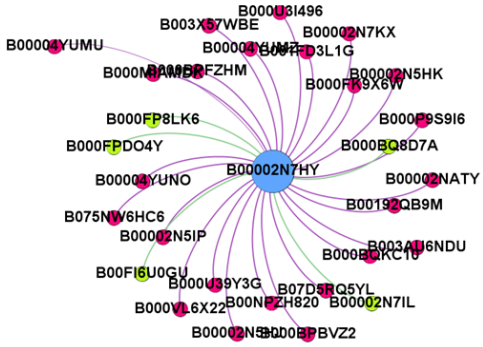

Fig 3: Top five products

For example, in Fig 2 it can be seen that the product whose recommendations we want to find has the ID B00002N7HY. It is the main node to which all the other nodes (product IDs) are connected to. In reality these nodes are also connected to many other nodes, not just B00002N7HY, but to find the recommendations for B00002N7HY we only need to consider the nodes directly connected to it. All these nodes contain a degree centrality score. Once the product is queried, the edge weights between the product and the connected nodes are found and the centrality score is divided by it for individual connected nodes. Thus, a list of scores is found for all the connected nodes. Once we have the list of scores, it is sorted according to the scores and five products with the best scores are selected. These are shown in yellow in Fig 3.

4.2.4 Sentiment Analysis

The shortlisted candidate products do represent a robust set of products that can be recommended to the user. The methods used to arrive at this borrow greatly from collaborative filtering and content filtering. We hypothesize that adding an overall sentiment as a factor into the recommendation mix will provide for more suitable recommendations.

We do so by generating a sentiment score for a product using all the reviews associated with that product. Therefore, for every product, we aggregate the sentiment associated with each product [9]. We then take the average of this score to reach an average sentiment score associated for each product.

A problem we found is that many products do not have any reviews associated with them. These could represent new products which we might want to recommend - suffering from a cold start problem [10] or off-brand/ fake products which people can easily filter out and ones we don't want to recommend. The solution for cold start problems which are based on collaborative filtering methods is to add content-based features. We achieve the same by assigning a sentiment score of 0 for all such products. In practice, this achieves a dual purpose:

1. If there are no other recommended products or only recommended products with negative sentiment (ones we wouldn't recommend anyway), we now recommend this product.
2. If there are other products which people can buy and they have positive reviews, those get recommended instead.

Therefore, we now are left with a shortlist of products which are sorted according to their sentiment score and recommended to the user. This forms our fine-tuned product recommendation. We can choose the top 3 from this list as our final recommendations.

4.3 Evaluation

Since the dataset itself belongs to Amazon, it can be assumed that Amazon would be most likely to recommend products that are related to a particular product, i.e., products that have been viewed as well as bought along with that product. However, the total number of related products that have been viewed or bought can be quite high. In these cases, the fine-tuned recommendation system chooses only the top three products to recommend based on sentiment and network scores. For a particular product ID (B002MPLYEW) , a comparison between the total number of related products that might be recommended by Amazon and the number of recommended products by our fine-tuned recommender is given in Fig 4.

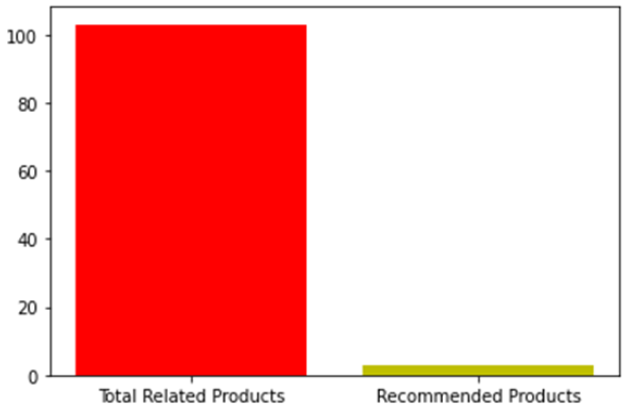

Fig 4: Comparison between total related products and products recommended by the fine-tuned recommender.

This saves the trouble for the customer to browse through all the recommendations and filtering them via ratings. Due to our advantage of sentiment analysis, the fine-tuned recommender only recommends products that have an overall positive sentiment based on reviews by previous customers. Table 1 clearly shows the difference between three products that might be recommended for the product in question by Amazon, and by our fine-tuned recommender system. The recommendations in Table 1 are based on the previous product ID in question, i.e., B002MPLYEW. The ratings in the table were calculated by finding the average ratings across all the reviews for the same product for both Amazon and Fine-tuned recommendation systems. It is important to note however that the recommendations that are being shown as Amazon recommendations may not necessarily be what Amazon recommends. However, for convenience of evaluating the performance of the recommendations system, it is assumed that Amazon may show these products as recommendations as they are related to the product in question. The usability of the Fine-tuned recommendation system is that it avoids even those recommendations.

Amazon Recommendations			Fine-Tuned Recommendations				
Product ID	Name	Rating	Product ID	Name	Sentiment Score	Network Score	Rating
B001KYE1PA	EdgeStar BWC120SS 103 Can and 5 Bottle Extreme Cool Beverage Cooler - Stainless Steel	2.63	B0170O0D82	EdgeStar KC2000TWIN Full Size Dual Tap Kegerator & Draft Beer Dispenser - Black	0.61	0.001	4.09
B007PNX1AG	EdgeStar CBR1501SG 24 Inch 148 Can Built-in Beverage Cooler	2.57	B00655HJJE	EdgeStar KC2000SSTWIN Full Size Stainless Steel Dual Tap Kegerator & Draft Beer Dispenser - Stainless Steel	0.6	0.0011	4.15
B0085BEMV4	EdgeStar BWC120SSLT 103 Can and 5 Bottle Freestanding Ultra Low Temp Beverage Cooler	3.802	B014LGBJVC	EdgeStar KC1000SS Craft Brew Kegerator for 1/6 Barrel and Cornelius Kegs	0.56	0.0016	4.24

Table 1: Comparison between possible Amazon recommendations and recommendations by Fine-tuned recommender.

From Table 1 it can be observed that the above related products might be recommended by Amazon despite it's low rating, whereas the Fine-tuned recommender recommends only the products which have high rating (or positive sentiment). As mentioned before, it may be that there are no reviews for a particular product, in which case the network scores will become the deciding factor in making a recommendation.

5. CONCLUSION

In our experiment, we use network metrics such as degree centrality in conjunction with the natural language processing technique of sentiment analysis to fine-tune recommendations provided by Amazon. We do this for one product category but the same can be extended to all products. From this effort, we learned to apply our network analysis skills to a large-scale network and see the effectiveness of our skills at this scale. We also learn the problems of scale, for example, our sentiment analysis task took 10 hours to complete for a 90k odd product which had 4 reviews each on average.

In terms of future work, we hypothesize that using the "helpfulness" score associated with each review along with the sentiment of the review would provide for more robust results. The helpfulness score provides important user feedback for the review itself and could make the sentiment of the review more meaningful. This could potentially remove bot or spam reviews. Including the "verified purchase" as a boolean qualifier for the review could also provide for more meaningful results especially in cases where we have too many reviews. However, in case of limited reviews we can forgo this filtering.

A clear bottleneck for our recommendation pipeline is the sentiment analysis process. We initially implemented a state-of-the-art BERT transformer based model but found the inference times to be unreasonably high. We now feel that we have struck a balance between efficiency and accuracy. However, using certain smarter filtering techniques as mentioned above and the use of a faster model, we could achieve more robust results.

REFERENCES

1. https://www.rejoiner.com/resources/amazon-recommendations-secret-selling-online
2. https://www.dynamicyield.com/article/amazon-recommendations/
3. W. Waheed, M. Imran, B. Raza, A. K. Malik and H. A. Khattak, "A Hybrid Approach Toward Research Paper Recommendation Using Centrality Measures and Author Ranking," in IEEE Access, vol. 7, pp. 33145-33158, 2019, doi: 10.1109/ACCESS.2019.2900520.
4. Anahita Davoudi, Mainak Chatterjee, Social trust model for rating prediction in recommender systems: Effects of similarity, centrality, and social ties,https://doi.org/10.1016/j.osnem.2018.05.001
5. C.J. Hutto, Eric Gilbert,"VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text" ,https://comp.social.gatech.edu/papers/icwsm14.vader.hutto.pdf
6. https://nijianmo.github.io/amazon/index.html
7. http://snap.stanford.edu/data/web-Amazon-links.html
8. http://snap.stanford.edu/data/amazon-meta.html
9. Fang, X., Zhan, J. "Sentiment analysis using product review data" . Journal of Big Data 2, 5 (2015). https://doi.org/10.1186/s40537-015-0015-2
10. Lika, Blerina & Kolomvatsos, Kostas & Hadjiefthymiades, Stathes. (2014). "Facing the cold start problem in recommender systems." Expert Systems with Applications: An International Journal. 41. 2065-2073. 10.1016/j.eswa.2013.09.005.

SUPPLEMENTARY MATERIAL

Demo: https://drive.google.com/file/d/1zo_hdAYjK57n46SvwaNgO16eHjmgWcWe/view?usp=sharing

```
degree_centrality = nx.degree_centrality(G)
bar = list(G.nodes)
recommendations = {}
for node in tqdm(bar):
  scores = []
  connections = list(G.neighbors(node))
  if len(connections)<3 and len(connections)!=0:
    recommendations[node] = connections
  elif len(connections)>=3:
    for con_node in connections:
      degree = degree_centrality[con_node]
      weights = G.get_edge_data(node,con_node)['weight']
      scores.append([con_node, degree/weights])
    scores = pd.DataFrame(scores, columns= ["Items","Scores"])
    scores = scores.sort_values(by="Scores",ascending = False)
    scores = scores.head()
    recommendations[node] = scores.values.tolist()
  elif len(connections) == 0:
    recommendations[node] = "No related product!"
```

Code for calculating degree scores

```
i=1
final_recommendations_dict={}
for key,value in tqdm(recommendations.items()):
  # if i%999==1:
  #   print(i)

  if value == "No related product!":
    final_recommendations_dict[key]=["No related product!"]
    continue

  recommended_products = value
  final_recommendations=[]
  for recommended_product in recommended_products:
    reviews = list(df.loc[df['asin']==recommended_product[0]]['reviewText'])
    if len(reviews)>0:
      reviews = [str(r) for r in reviews]
      revs=[]
      review_score=0
      for r in reviews:
        words = word_tokenize(r)
        wordsFiltered = []
        for w in words:
          if w not in stopWords:
              wordsFiltered.append(w)
        ss=sid.polarity_scores(' '.join(wordsFiltered))
        review_score+=ss['compound']
      review_score/=len(reviews)
      # review_score= sum([res['score'] for res in pipe(reviews)])/len(reviews)
    else:
      review_score= 0
    final_recommendations.append((recommended_product,review_score))
  final_recommendations.sort(key = lambda x: x[1], reverse = True)
  final_recommendations_dict[key] = final_recommendations
  i+=1
```

Code for Sentiment Analysis