Big Data and Data Science Diploma Graduation Project

Inferring Networks of Substitutable and Complementary Products Model Implementation

Prepared by:

Abeer Mohamed

Ahmad Khalil

Ayman Yosry

Supervised by:

Dr. Inas Yassine

Contents

Introduction

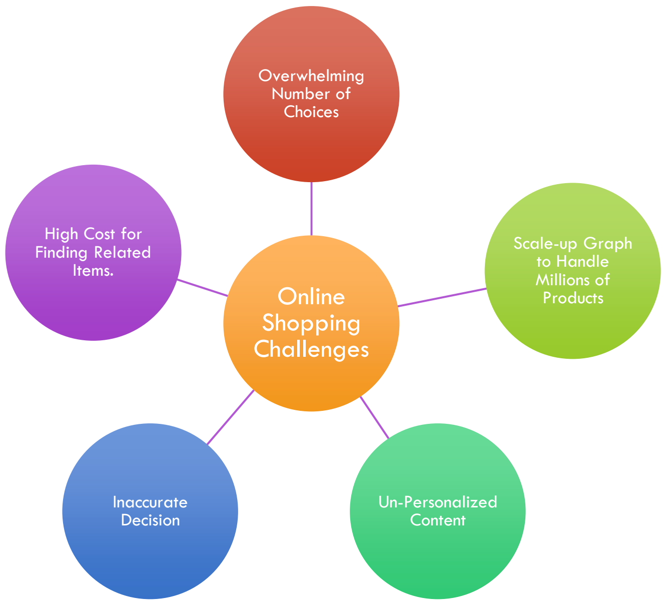
Today, every user of the World Wide Web can purchase almost any item being in any country of the world. As opposed to real shops, in the internet there are no place-limitations. Nevertheless people came across a new problem in the WWW. The amount of information and items got extremely huge, leading to an information overload. It became a big problem to find what the user is actually looking for. Search engines partially solved that problem, however personalization of information was not given. So recommender systems was a solution [1].

Recommender systems are tools for filtering and sorting items and information. They use opinions of a community of users to help individuals in that community to more effectively identify content of interest from a potentially overwhelming set of choices.

In this project we are implementing ready recommender model. The model is provided by Julian McAuley, Rahul Pandey, and Jure Leskovec. They develop a method to infer networks of substitutable and complementary products. The paper title is “Inferring Networks of Substitutable and Complementary Products”. Our contribution here that we implement this model on Spark and we deployed it using Graphx; in addition to different data preparation approach.

Problem Statement

On the Internet, where the number of choices is overwhelming, there is need to filter, prioritize and efficiently deliver relevant information in order to alleviate the problem of information overload, which has created a potential problem to many Internet users[3]. In addition to un-personalized content and services; the costs of finding and selecting items in an online shopping environment is expensive. The decision making process is inaccurate. The scaling the huge number of available products is much challenged.


Overwhelming Number of Choices
High Cost for Finding Related Items.
Online Shopping Challenges
Scale-up Graph to Handle Millions of Products
Inaccurate Decision
Un-Personalized Content

Recommender Systems

TRADITIONAL RECOMMENDER APPROACHES [2]

A. Content-based filtering

Content-based recommender systems work with profiles of users that are created at the beginning. A profile has information about a user and his taste. Taste is based on how the user rated items. Generally, when creating a profile, recommender systems make a survey, to get initial information about a user in order to avoid the new-user problem.

B. Collaborative filtering

Collaborative filtering became one of the most researched techniques of recommender systems. The idea of collaborative filtering is in finding users in a community that share appreciations. If two users have same or almost same rated items in common, then they have similar tastes. Such users build a group or a so called neighborhood. A user gets recommendations to those items that he/she hasn't rated before, but that were already positively rated by users in his/her neighborhood.

MODERN RECOMMENDATION APPROACHES [2]

A. Context-aware approaches

Context is the information about the environment of a user and the details of situation he/she is in. Such details may play much more significant role in recommendations than ratings of items, as the ratings alone don't have detailed information about under which circumstances they were given by users. Some recommendations can be more suitable to a user in evening and doesn't match his preferences in the morning at all and he/she would like to do one thing when it's cold and completely another when it's hot outside.

B. Semantic based approaches

Most of the descriptions of items, users in recommender systems and the rest of the web are presented in the web in a textual form. Using tags and keywords without any semantic meanings doesn't improve the accuracy of recommendations in all cases, as some keywords may be homonyms. That's why understanding and structuring of text is a very significant part recommendation. Traditional text mining approaches that base on lexical and syntactical analysis show descriptions that can be understood by a user but not a computer or a recommender system. That was a reason of creating new text mining techniques that were based on semantic analysis. Recommender systems with such techniques are called semantic based recommender systems.

In a modern recommender system, it is important to understand how products relate to each other[2]. For example, while a user is looking for mobile phones, it might make sense to recommend other phones, but once they buy a phone, we might instead want to recommend batteries, cases, or chargers. These two types of recommendations are referred to as substitutes and complements: substitutes are products that can be purchased instead of each other, while complements are products that can be purchased in addition to each other.

Here we develop a method to infer networks of substitutable and complementary products. We formulate this as a supervised link prediction task, where we learn the semantics of substitutes and complements from data associated with products [2]. The primary source of data we use is the text of product reviews, also makes use of prices, and brands. Methodologically, we build topic models that are trained to automatically discover topics from text that are successful at predicting and explaining such relationships. Experimentally, we evaluate our system on the Amazon product catalog.

Business Understanding

Recommender systems are ubiquitous in applications ranging from e-commerce to social media, video, and online news platforms. Such systems help users to navigate a huge selection of items with unprecedented opportunities to meet a variety of special needs and user tastes. Making sense of a large number of products and driving users to new and previously unknown items is key to enhancing user experience and satisfaction[4].

While most recommender systems focus on analyzing patterns of interest in products to provide personalized recommendations, another important problem is to understand relationships between products, in order to surface recommendations that are relevant to a given context. For example, when a user in an online store is examining t-shirts she should receive recommendations for similar t-shirts, or otherwise jeans, sweatshirts, and socks, rather than (say) a movie even though she may very well be interested in it. From these relationships we can construct a product graph, where nodes represent products, and edges represent various types of product relationships. Such product graphs facilitate many important applications: Navigation between related products, discovery of new and previously unknown products, identification of interesting product combinations, and generation of better and more context-relevant recommendations.

Despite the importance of understanding relationships between products there are several interesting questions that make the problem of building product graphs challenging[2]: What are the common types of relationships we might want to discover? What data will allow us to reliably discover relationships between products? How do we model the semantics of why certain products are related?—For example, the semantics of why a given t-shirt might be related to a particular pair of jeans are intricate and can only be captured by a highly flexible model. And finally, how do we scale-up our methods to handle graphs of millions of products and hundreds of millions of relations?

Inferring networks of product relationships. Here we are interested in inferring networks of relationships between millions of products. Even though our method can be used to learn any type of relationship, we focus on identifying two types of links between products: substitutes and complements. Substitutable products are those that are interchangeable—such as one t-shirt for another, while complementary products are those that might be purchased together, such as a t-shirt and jeans.

Data Understanding

We used a data extracted from amazon (Baby products data).

Three files:

1- meta_Baby contains the main data about the product
   a. asin (id)
   b. price, company
   c. also viewed products
   d. also bought products
   e. description
   f. title
2- qa_Baby contains questions and answers about the products
   a. asin (id)
   b. question
   c. answer
3- reviews_Baby_5 contains the reviews written by users on the product
   a. asin (id)
   b. review
   c. reviewer name
   d. review time
   e. rating

Notes:

1- Some data is incomplete, some products has no text.
2- Merging all the text will give a well sized text for the product
   so we merged description, title, questions, answers and reviews for each product to form a one large text for each product

Data Preparation

1- The Files we got are in the Dict python serialization format So first we Changed them into JSON format contains fields needed:
   a. asin (id)
   b. price
   c. company brand name
   d. also viewed products ids
   e. also bought products ids
   f. all text available on the product (reviews, title, description, questions, answers)

2- All the text aggregated into one large Doc for each product

3- Products has null or zero length doc are filtered out

4- Create the relations depending on also viewed and also bought data
   a. The also viewed data are considered as supplementary
   b. The also bought data are considered as complementary

5- Replicate some no related data size of products that has no also viewed nor also bought data is very small (just 10 %) we created a not related products data from those products
   a. for every product we selected 10 random products from the whole data
   b. create 10 not related products rows' from the ids of the products

Goals achieved by this step

1- Now data is ready for the Text Mining step

2- extra not used data in our algorithm are filtered out from the data

3- empty useless products (has no reviews nor description) are filtered out

Model

Our Model is built on, that is capable of modeling and predicting relationships between products from the text of their reviews and descriptions. At the model core, we combine topic modeling and supervised link prediction, by identifying topics in text that are useful as features for predicting links between products. Our model also handles additional features such as brand, price, and rating information, product category information, and allows us to predict multiple types of relations (e.g. substitutes and complements) simultaneously [2].

We made the training on a large corpus of products from Amazon, with connections derived from browsing and co-purchasing data. We also build a product graph, where for every product we recommend a list of the most related complementary and substitutable products. Finally, our model can be applied in 'cold-start' settings, by using other sources of text when reviews are unavailable. Overall, we find that the most useful source of information to identify substitutes and complements is the text associated with each product (i.e., reviews, descriptions, and specifications), from which we are able to uncover the key features and relationships between products, and also to explain these relationships through textual signals. Our model can help users to navigate, explore and discover new and previously unknown products.

Model Steps:

1- **Text mining phase:**
   a. Text tokenization
   b. Remove stop words
   c. Stemming the words
   d. Removing empty words or words whose length is just one character
   e. Removing words that are just digits
   f. Doing count vectorization on the text

2- **LDA phase:** (Latent Dirichlet Allocation)

This is done on the text to get the most k (**k= 9** in our model) common topics in the reviews and get its percent of representation in each product text

3- We used the also viewed and also bought data to detect if there is a relation between products and its type
   a. We considered the also viewed as complementary products
   b. We considered the also bought as supplementary products

4- Preparation step before Logistic regression
   First
   the result of the LDA is a vector of the probability of occurrence of each topic in the text of the product
   For every two related products
   a. We multiply the 2 vectors produced from the LDA
      if they have the same topics the result of the multiplication will not differ much than the original data of every product. So they represent the same topics so they are mostly of the same category
      if they have different topic representation or some topics are represented in one product and not on the other, the result of the multiplication will result in very small values which may be zero. So those products are probably not in the same category
   b. We subtracted the 2 vectors produced from the LDA after adding the price of each in the vectors, also we added a value for the new resulted vector representing if they have the same company or (0.1 and 0.9 respectively)
      this will give us a direction if prod1 will be recommended for prod2 or vice versa

   Second we used some products which has no relations to randomly create some not related products
   Third we marked the also viewed related products as supplementary and the also bought as complementary products and used the multiplication result for them

5- Logistic regression
   we trained logistic regression models to use in future predictions of new products
   we trained 2 models
   a. First one to predict if there is a relation between the 2 products
      to predict the relation (weight and direction)
      i. we trained a logistic regression model on the multiplication result
      ii. we trained a logistic regression model on the subtraction result

    iii. we multiplied the two logistic regression prediction results
    the resulted value will detect the strength of the relation and the sign of the value (+ve or –ve) will detect the direction of the relation

b. Second one is to predict the relation type (complementary or supplementary)

6- **Graph creation phase:**

We created a graph of the related products to use in predictions

The edge:

a. is weighted to detect the strength of the relation between the product
b. carrying the relation type (complementary or supplementary)

NOTE : the graph is directed from one product to the other to detect the direction of recommendation e.g. you may recommend the charger as a complementary to a laptop but can\not do the opposite to detect this direction we used the price and the company similarity we may recommend the cheaper and the one from another company

Model Evaluation

The model evaluation depends on training data mentioned in previous section as following

1- The model studied against training data with number of iteration 100 to predict relation weight
2- Set a threshold on predicted relation weight to map it to Boolean format describes if it is related or not
3- compare the result with the given labels and get training error by counting non matched using following equation
   train-error = count (label != pred)/count(lables)

Deployment

The best way to represent the relations between related products; is to represent them as a graph. Graph is a mathematical structures used to model pairwise relations between objects. A graph in this context is made up of *vertices* (nodes or *points)* which are connected by *edges* (arcs, or *lines).* A graph may be *undirected*, meaning that there is no distinction between the two vertices associated with each edge, or its edges may be *directed* from one vertex to another [5].

1. **Vertex**

   A vertex is the most basic part of a graph and it is also called a node. In the recommendation system model the product is represented as vertex.

2. **Edge**

   An edge is another basic part of a graph, and it connects two vertices. Edges may be one-way or two-way. If the edges in a graph are all one-way, the graph is a directed graph, or a digraph. Edge is considered as a strong relationship between two products.

   Edge will contain two information:

   1. Weight which represent the strength of the relation
   2. Relation type which determine if the two products are supplementary or complementary.

3. **Weight**

   Edges may be weighted to show that there is a cost to go from one vertex to another. For example in a graph of roads that connect one city to another, the weight on the edge might represent the distance between the two cities or traffic status. In our model the edge weight is representing the strength of the relation.

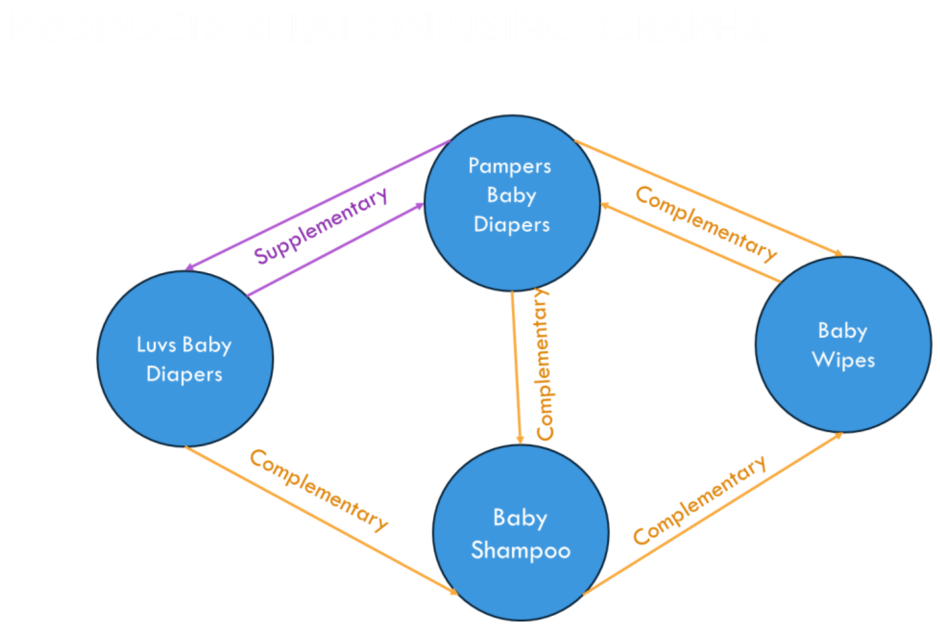
From that graph we get the products related to another product and sort those related product according to the strength of the relation. Those related products can be viewed in different sections supplementary and complementary sections.

Why Graphx?

GraphX is Apache Spark's API for graphs and graph-parallel computation. Seamlessly work with both graphs and collections. GraphX unifies ETL, exploratory analysis, and iterative graph computation within a single system. You can view the same data as both graphs and collections, transform and join graphs with RDDs efficiently. Comparable performance to the fastest specialized graph processing systems. GraphX competes on performance with the fastest graph systems while retaining Spark's flexibility, fault tolerance, and ease of use. GraphX is developed as part of the Apache Spark project. It thus gets tested and updated with each Spark release [6].

The relation graph is built for product with weight above specific threshold only. For every related products the edge contains the relation type supplementary and complementary.

When customer browsing any product, system will check the product relation graph and return all related product(s), then for every related product, system will check the relation type supplementary or complementary.



Limitations of the Proposed Solution

The implemented solution is already applied the same as the paper solution, that is why we didn't face a limitations.

All limitations we have faced are technical implementation as the below:

- In the data preparation we try to implement the count factorization using Pyspark but it wasn't working well; so we convert it to Java Spark instead.
- In the Graph phase; we faced many tires to implement graph using Python, NetworkX package, GraphForm Python/Scala and all of them didn't work well, so we convert it to Scala GraphX.

Conclusion

Strong recommender system should match user expectations as possible and this has already been achieved in our implementation for Inferring networks methodology for implementing this recommender system to help user who browse a particular product by recommending complementary products (products that can be purchased in addition) and supplementary products (products that can be purchased instead of) sorted by some criteria, and this happened by applying idea of topic categorization for each product and find a relationship between products' topics. Our goal has been to learn these concepts from product features (topic categorization using LDA), especially from the text of their reviews and predicting strong relation between products (using Logistic Regression) and finally store and explore the result in form of graph. We have applied this to the problem of identifying substitutable and complementary products on subset of a large collection of Amazon data (Baby Section), including a lot of reviews and ground-truth relationships based on browsing and co-purchasing logs.

References

1. https://www.snet.tu-berlin.de/fileadmin/fg220/courses/SS11/snet-project/recommender-systems_asanov.pdf
2. J McAuley, R Pandey, and J Leskovec Inferring Networks of Substitutable and Complementary Products. Sydney, NSW, Australia — August 10 - 13, 2015
3. http://www.sciencedirect.com/science/article/pii/S1110866515000341
4. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.39.2552&rep=rep1&type=pdf
5. https://en.wikipedia.org/wiki/Graph_(discrete_mathematics)
6. https://spark.apache.org/graphx/
7. http://interactivepython.org/courselib/static/pythonds/SortSearch/searching.htmlhttp://interactivepython.org/courselib/static/pythonds/Graphs/VocabularyandDefinitions.html
8. https://www.ibm.com/developerworks/library/os-recommender1/
9. http://recommender-systems.org/