Product Name Match and Item Relationship Exploration on Transaction Logs

Hanna Hamilton
hhamilton33@gatech.edu

Jingyu Li
alanli@gatech.edu

Pedro Henrique Ribeiro Pinto
phpinto@gatech.edu

Qihang Zhang
qihang.zhang@gatech.edu

Xinze Wang
xwang992@gatech.edu

1 Business Problems and Project Goals

1.1 Background and Problem Definition

This project aims to standardize product names and categories from various merchants and generate insights to improve and automate merchant operations. NCR serves thousands of retailers and restaurants which all have their own product catalogs. There is a lack of consistent product names and categories across product catalogs from different merchants, leading to poor decision making, excess labor expended on manual catalog curation, supply chain errors, and mistakes that decrease customer satisfaction.

By resolving these inconsistencies, NCR can create more data-driven product offerings and provide services which improve and automate merchant operations. If the models for product name matching and item relationship exploration can be built correctly and efficiently, it will benefit NCR, its clients, and its clients' customers.

The main problems in this project are defined as:

Problem 1 (Main Problem)**.** Given a master catalog including the generic name of different products, match the name variations of the identical product from different merchants to the generic name in the master catalog.

Problem 2 (Main Problem)**.** Identify meaningful relationships between products for making product recommendations.

1.2 Summary of Analytics Approach

For product name matching, we managed to automatically classify product name entries across merchants into a consistent set of groupings. First, we applied a novel clustering method to remove duplicates in the USDA grocery product list to generate a valid master catalog. Then, we designed and implemented a similarity-based multi-step model to match the name variations to the generic master catalog. To accomplish this, we broke the product names into substrings by letter level N-grams, used TF-IDF vectorization to generate product name embeddings, and computed cosine similarities.

For item recommendation, we implemented several independent algorithms to explore complementary or substitute relationships between items. We explored complementary relationships via implementation of the Apriori algorithm. We explored substitute relationships via implementation of a transaction co-occurrence similarity approach, as well as a graph convolutional network algorithm.

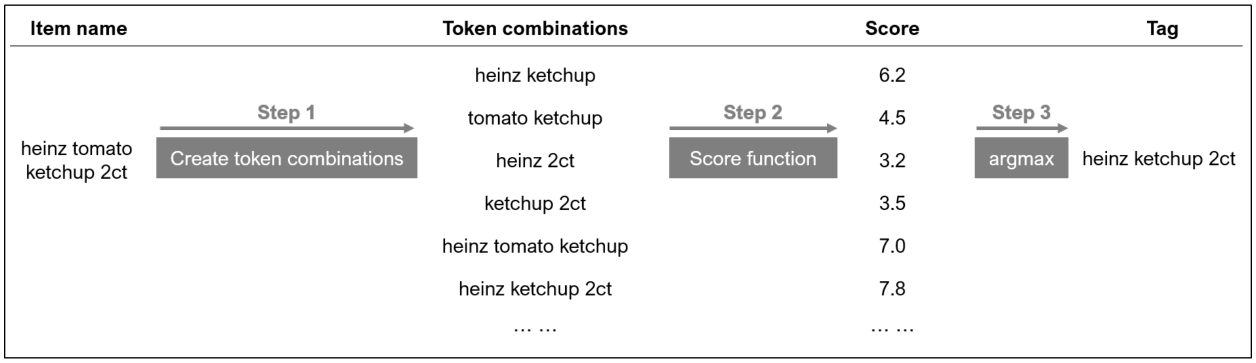


Figure 1: Algorithm Outline of the Clustering Method

2 Product Name Match

The goal of the first problem of this project is to automatically classify identical products with various name entries in transaction records across merchants into a consistent master name. After surveying research papers and experimenting with different modeling approaches, we implemented a two-stage model. To accomplish this, we first generated a master catalog containing all possible grocery products based on the data from USDA. Then, we applied a similarity measure to match the name entries in NCR's transaction logs to the items in the master catalog.

2.1 Master Catalog Generation

We proposed that an ideal and valid master catalog would contain different items as many as possible without any duplicated items in it. According to the business needs, the items in the catalog is defined at the UPC (Universal Product Code) level, which means the same product with different size (or flavor) are counted as different items in the catalog.

However, we didn't find any public well-established grocery product catalog as we want. We found that the USDA FoodData Central Data provide a branded foods list of the market which was close to the expected master catalog. By exploring the dataset, we revealed that there were some flaws in the dataset like duplicated entries and product name typo. We applied a novel clustering method proposed by Akritidis et al. [2] to generate the master catalog based on the USDA dataset, which can also fix the flaws existing in the original dataset.

2.1.1 A Self-verifying Clustering Method on Token Combinations

The key algorithm of this clustering method is firstly generating token combinations for each original product name, then scoring each token combination and choosing the combination with highest score as the generic name tag for the original product name. Thus, product names with the same tag are clustered into the same group (As shown in Figure 1).

We implemented a token lexicon and a combination lexicon to generate and store the token combinations for each product name (See Figure 2). In token lexicon, we assigned the unique id to each token and recorded the frequency and semantic type of the token. We identified three types of semantic type in our project: "class" (tokens represent a generic product class, like milk, tea and ketchup etc.), "size" (tokens represent the size of the product like a number with a unit or some size-related words) and "normal" (all other tokens). In combination lexicon, we recorded the frequency, number of tokens and distance of each combination. The Distance is the index-based Euclidean distance between the combination with its original product name. When computing the distance, we retained the token's relative position from original name and calculated the sum of distance corresponding to all product names that contain the combination. The distance measurement reflects how far away the tokens in the combination to the head of the original name.

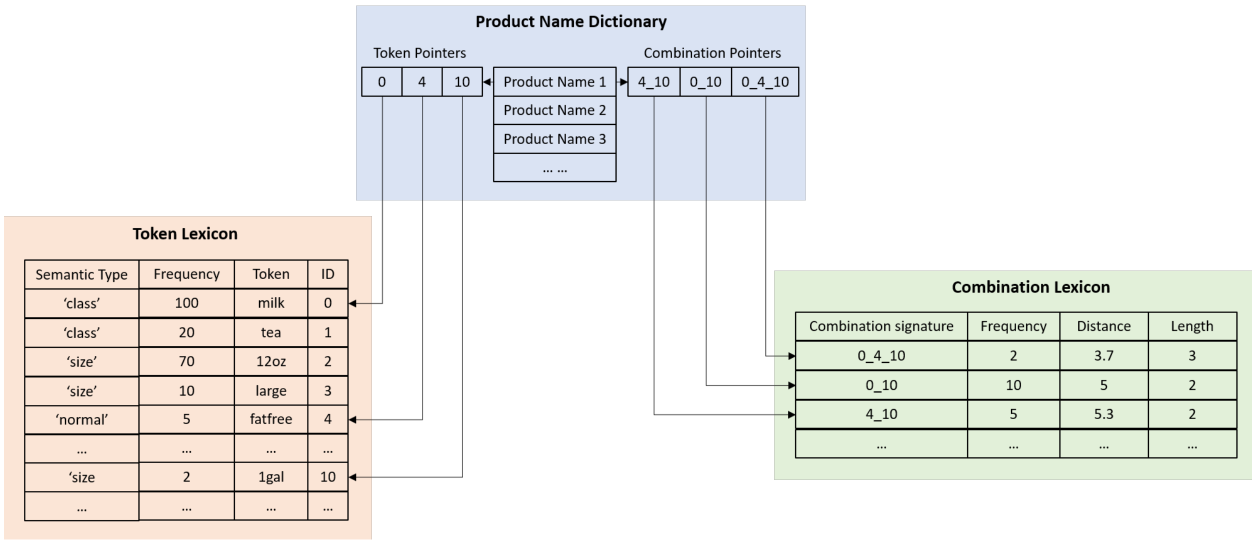


Figure 2: The Data Structure for Clustering Method

We used the same scoring function from the literature [2] as following:

$$I(c) = \frac{kY_c^2}{a + d_c/f_c} \log(f_c), \quad in\ which\ Y_c = \sum_{w \in c} idf(w) \frac{1/Q_{\text{type}}}{1 - b + bk/\bar{l}_c} \tag{1}$$

The parameters in Equation 1 are:
k: the number of tokens in the combination
d_c: the distance of the combination
f_c: the frequency of the combination in the corpus
Y_c: the IR score of the combination
$idf(w)$: the inverse document frequency of the token w in the combination, which equals to $\log \frac{\text{total number of product names}}{\text{token frequency}}$
$\bar{l}_c$: average length of all combinations
Q_{type}: constant weights for tokens with different semantic type
a: a constant number avoiding the denominator being 0
b: a constant number between $[0, 1]$

After getting the score for each combination, we selected the token combination with the highest score as the generic tag for the original product name. Then we clustered the original product name with the same generic tag as a group.

The next step is to verify the product names in each cluster to add them into the master catalog. For the cluster with only one product name, we assumed that it's a unique product at UPC level and added that product into the master catalog. For the cluster with more than one product names, we verified the items cluster by cluster. The way of verification is that we split the product name into "size tokens" and "remaining part". If all products in this cluster have different size, it means they are different product at UPC level and we added them into the master catalog. If some of the products has the same size, we compare the string similarity of the remaining part except the size tokens. If the similarity between two product names is over 0.95, we regarded them as the duplicated name and only added the name with higher token frequency into the master catalog. In such a way, we generated the master catalog for our project.

2.1.2 Dataset, Experiment Settings and Results

The dataset we used to generate master catalog is from the USDA FoodData Central Data [1]. The original number of product names included in the dataset is 238,006. The steps of pre-processing the data are 1) Change all names to lower case, 2) Replace unit token to its abbreviate form (e.g. "ounce" to "oz"), 3) Remove all punctuations in the product name except dot in decimal (e.g. 2.4), % in percentage number (e.g. 100%), and line in fraction (e.g. 1/2), 4) Remove the space between a number and a unit token (e.g. "2 oz" to "2oz").

To ensure the efficiency of the algorithm, we set the max number of tokens in the combination to be 5. And the constant parameters in the scoring function are set as: $a = 0.001, b = 0.5, Q_{class} = 500, Q_{size} = 27000, Q_{normal} = 11000$.

By applying the clustering algorithm, we got 157,083 clusters. In general, the same product with different size or flavor were clustered together. Then we verified the cluster one by one to identify potential duplicates. The final master catalog we generated contained 226,655 items.

2.2 Similarity-based Product Name Match

The key idea behind this approach is to establish a **mathematical similarity measure between strings** and match product names with the highest scores.

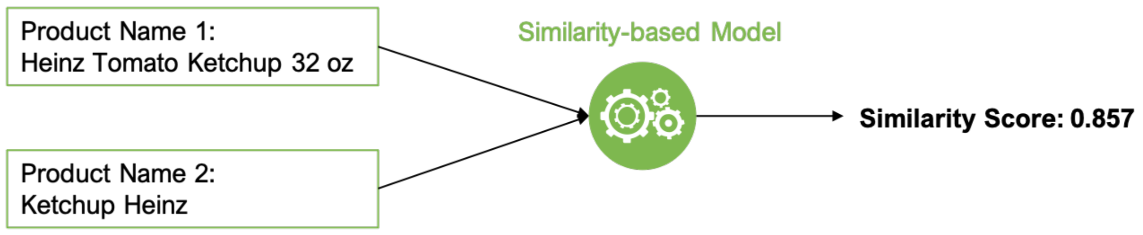


Figure 3: Visual Representation of the Similarity-Based Approach

We started by developing a Naïve model to test the effectiveness of using statistical string similarity as a fuzzy-matching approach. We then analyzed performance of that initial model to build a more robust multi-step version.

2.2.1 Naïve Model

For the Naïve model, each new product name is compared to all product names in the Master Catalog based on the number of substring matches. The new product is then matched to the product in the Master Catalog that generates the highest similarity score.

In our experiments, we used the *Grocery.com* [4] dataset to represent the new product names and the *USDA* [1] dataset to represent the Master Catalog. We also performed some string pre-processing techniques to improve the quality of the match results.

- **Data Sources:**
  - **Master Catalog:** food.csv file from the *USDA* [1] dataset
  - **New Product Names:** Grocery_UPC_Database.xlsx file from the *Grocery.com* [4] dataset

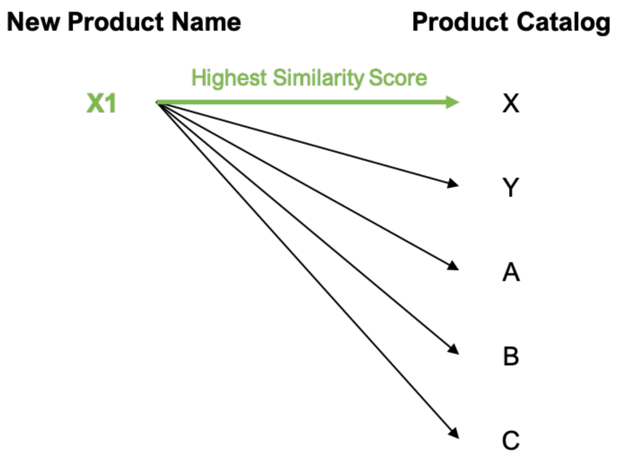


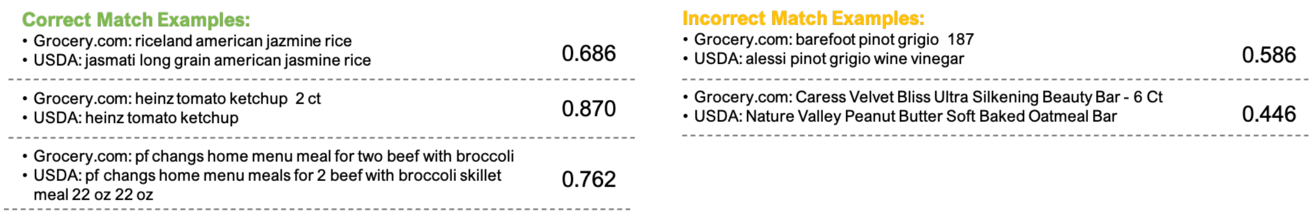
Figure 4: Visual Representation of the Similarity-Based Naïve Model

- **String Pre-Processing:**
  - Letter lower-casing and Punctuation removal
- **Similarity Method:**
  - **Implementation:** *SequenceMatcher* class from the difflib Python library.
  - **Formula:**

$$score = \frac{M}{T}$$

  where M = of substring matches, T = total of elements

The Naïve model performed reasonably well, proving that a similarity-based approach was feasible for matching product names. Since the datasets on this experiment did not contain labels to be used as ground truth, we had to analyze the quality of the results based on our own knowledge of grocery product names. Here are some examples of matches produced by this model:



Correct Match Examples:

- Grocery.com: riceland american jazmine rice
- USDA: jasmati long grain american jasmine rice

0.686

- Grocery.com: heinz tomato ketchup 2 ct
- USDA: heinz tomato ketchup

0.870

- Grocery.com: pf changs home menu meal for two beef with broccoli
- USDA: pf changs home menu meals for 2 beef with broccoli skillet meal 22 oz 22 oz

0.762

Incorrect Match Examples:

- Grocery.com: barefoot pinot grigio 187
- USDA: alessi pinot grigio wine vinegar

0.586

- Grocery.com: Caress Velvet Bliss Ultra Silkening Beauty Bar - 6 Ct
- USDA: Nature Valley Peanut Butter Soft Baked Oatmeal Bar

0.446

Figure 5: Examples of product name matches from the Similarity-Based Naïve Model

Based on these results, we empirically established some match quality thresholds. Scores over 0.75 were considered high-probability matches while scores under 0.6 were considered non-matches. We named he range of scores between 0.6 and 0.75 as the *"Danger Zone"*, because, while the confidence level for that range was lower, correct matches were still observed fairly often.

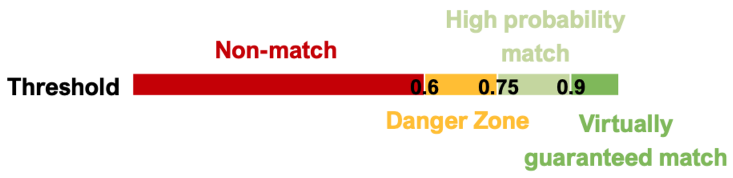


Figure 6: Match thresholds established for the Similarity-Based Naïve Model

This model produced satisfactory preliminary results and had the advantage of being simple and highly interpretable. Additionally, matches outputted by the model could potentially be used as labeled training data for more advanced models.

However, the Naïve Model also possessed some significant limitations: it was extremely slow, since each new product is compared to every item in the master catalog, and the similarity measure was overly simplistic. Consequently, we developed a more robust multi-step approach focused on addressing these issues.

2.2.2 Multi-Step Model

As aforementioned, the objective was to improve on the Naïve approach by using a more sophisticated string similarity measure while reducing the overall computation time. This new model applies the cosine similarity on vectorized versions of each product name and improves performance by taking advantage of fast matrix operations performed by numerical libraries like NumPy and SciPy. These are the descriptions of each of the three steps:

- **Step 1: N-gram Tokenization**
  - Product names are broken up into substring tokens of predetermined lengths called N-grams. The value of N is a hyperparameter.
  - Example: “Heinz Ketchup” 5-gram [‘Heinz’, ‘einz ’, ‘inz K‘, etc.]
- **Step 2: TF-IDF Vectorization**
  - Term Frequency - Inverse Document Frequency assigns higher value for terms (n-grams) that appear on very few product names and lower value for common terms. The resulting matrix has one row per product name and one column per unique n-gram.
  - TF-IDF Formula:
  $$tf-idf_{t,d} = (1 + \log tf_{t,d}) \cdot \log \frac{N}{df_t}$$
- **Step3: Optimized Cosine Similarity**
  - To match different product name vectors, we used an optimized implementation of the cosine similarity developed by the Dutch bank ING [3] that takes advantage of the sparsity of the TF-IDF matrix. We can also input a minimum matching score threshold and a maximum number of top matches per product.
  - Formula:
  $$\cos(\mathbf{t}, \mathbf{e}) = \frac{\mathbf{te}}{\|\mathbf{t}\|\|\mathbf{e}\|} = \frac{\sum_{i=1}^{n} \mathbf{t}_i \mathbf{e}_i}{\sqrt{\sum_{i=1}^{n} (\mathbf{t}_i)^2} \sqrt{\sum_{i=1}^{n} (\mathbf{e}_i)^2}}$$

The initial input is a column vector containing all the product names. We then split all the names into arrays of N-grams and input the modified vector to the *TfidfVectorizer* function from the *sklearn* Python Machine Learning library. This results in a sparse matrix whose rows represent each product name and columns represent the different N-grams observed across all product names.

In the following step, this newly constructed TF-IDF matrix is multiplied by its own inverse and the resulting matrix is inputted to the Optimized Cosine Similarity function. This generates a square sparse matrix where both the rows and columns represent each product name. Each index contains the similarity score between the product names in the index's row and column. However, only the top n highest similarity scores above the pre-established threshold are saved for each product name (where n is the chosen maximum number of matches during the cosine similarity calculation). Finally, we save the mathces from that matrix into a Python dictionary.

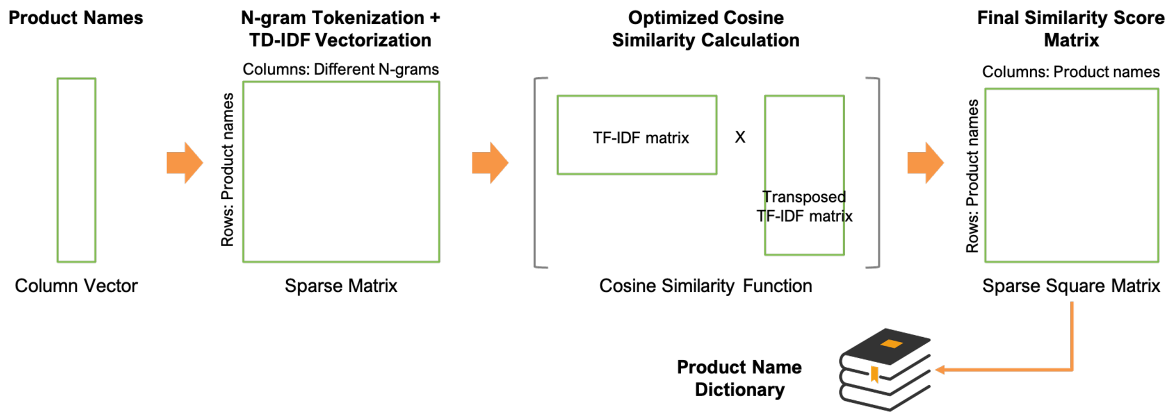


Figure 7: Visual representation of the matrix operations in the Multi-Step Model

After combining the product names provided by NCR with the new Master Catalog generated form the Self-verifying Clustering Method, the data set used to train the Multi-Step Model contained a total of 275,206 names. During our experiments, we noticed that, while the time it takes to compute the TF-IDF matrix is not significantly affected by the size of the dataset, the optimized cosine similarity calculation could be considered a bottleneck. However, this calculation could be performed in batch as a background procedure ahead of time.

We also devised an approach for quickly obtaining matches for new product names once the Product Name Dictionary has been generated. If the new name is one of the keys in the dictionary, it returns the existing matches for that name. If not, it quickly re-computes the TF-IDF Matrix and calculates the cosine similarity exclusively for the new product name. It then returns the newly found matches and updates the Product Name Dictionary.

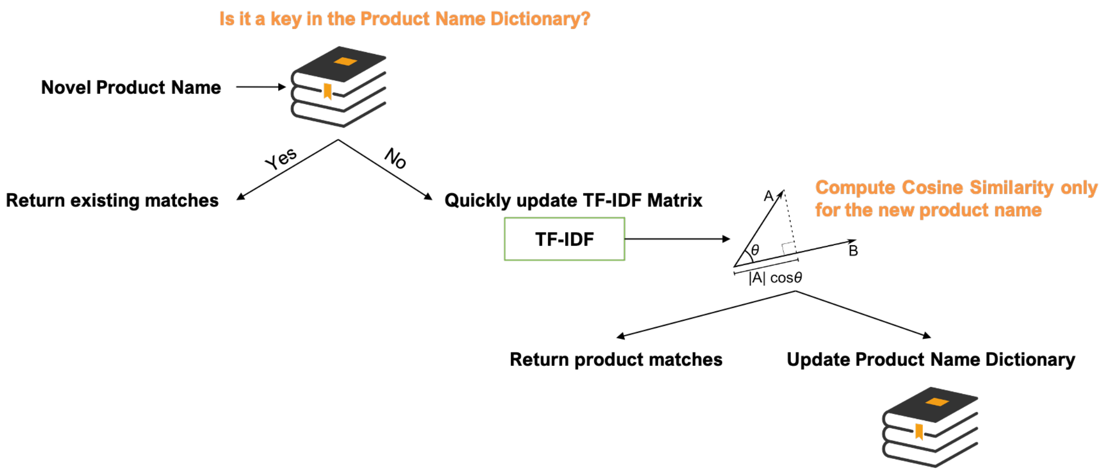
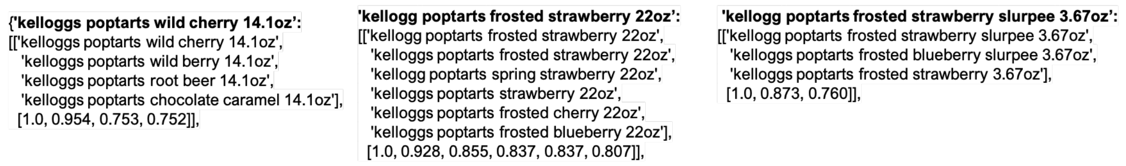

Is it a key in the Product Name Dictionary?
Novel Product Name
Yes
No
Return existing matches
Quickly update TF-IDF Matrix
TF-IDF
Compute Cosine Similarity only for the new product name
A
B
θ
|A| cosθ
Return product matches
Update Product Name Dictionary

Figure 8: Visual representation of the approach for obtaining matches for new product name

The Similarity-based Multi-Step Model was a highly effective approach and displayed a significant improvement form the initial naïve method both in terms of performance and quality of matches. Another great advantage of this approach is that it is very flexible, where changes in thresholds values, top number matches and size of n-grams allow it to better adapt to different datasets without compromising on computation time.

Here are some examples of matches produced by the Multi-Step Model:



```
{'kelloggs poptarts wild cherry 14.1oz':
[['kelloggs poptarts wild cherry 14.1oz',
  'kelloggs poptarts wild berry 14.1oz',
  'kelloggs poptarts root beer 14.1oz',
  'kelloggs poptarts chocolate caramel 14.1oz'],
 [1.0, 0.954, 0.753, 0.752]],
```

```
'kellogg poptarts frosted strawberry 22oz':
[['kellogg poptarts frosted strawberry 22oz',
  'kelloggs poptarts frosted strawberry 22oz',
  'kellogg poptarts spring strawberry 22oz',
  'kelloggs poptarts strawberry 22oz',
  'kelloggs poptarts frosted cherry 22oz',
  'kelloggs poptarts frosted blueberry 22oz'],
 [1.0, 0.928, 0.855, 0.837, 0.837, 0.807]],
```

```
'kellogg poptarts frosted strawberry slurpee 3.67oz':
[['kellogg poptarts frosted strawberry slurpee 3.67oz',
  'kelloggs poptarts frosted blueberry slurpee 3.67oz',
  'kelloggs poptarts frosted strawberry 3.67oz'],
 [1.0, 0.873, 0.760]],
```

Figure 9: Visual representation of the approach for obtaining matches for new product name

To achieve same level of granularity, the threshold was set in the range of 0.80 - 0.83.

3 Item Relationship Exploration and Item Recommendation

The goal of the second problem of this project is to identify meaningful relationships between products according to transaction logs, which forms the basis of product recommendation. There are two types of relationships between products: complementary and substitute. A complementary relationship exists between products which are commonly bought together. One example of a complementary relationship is peanut butter and jelly. A substitute relationship exists between products which are similar, and thus serve the same purpose. One example of a substitute relationship is Coke and Pepsi.

We implemented several independent methods to explore the complementary or substitute relationships between items, which are described in the following sections. Since there was no opportunity for us to evaluate the results based on ground truth or through online experiments, we mainly focused on implementing different algorithms, which form a basis for NCR to further explore at the production level.

3.1 Apriori Algorithm to Explore Complementary Relationships

The Apriori algorithm is a common and intuitive algorithm for identifying complementary relationships. We optimized our implementation of this algorithm to ensure that it can be used with large datasets.

3.1.1 Introduction of the Algorithm

The first step in this algorithm is to select all frequent items. An item is considered to be frequent if the number of transactions in which it appears is greater than or equal to a threshold. This threshold is a hyperparameter and is also referred to as the minimum support count. Likewise, an item's frequency is also referred to as its support count. All possible combinations of 2 are then created with the frequent items. The itemsets of 2 which are frequent in the transactional data are selected. An itemset is considered to be frequent if the number of transactions in which it appears is greater than or equal to the minimum support count. As the size of itemsets repeatedly increases by 1, this process repeats until an iteration is reached where there are no frequent itemsets. Alternatively, a stopping criterion can be used. Then, the lifts of all possible complementary relationships from all frequent itemsets are calculated. Lift is a measure of importance of a complementary relationship. A lift well above 1 indicates a strong complementary relationship. The lift calculation is as follows:

$$Lift(X->Y) = Lift(Y->X) = \frac{P(X,Y)}{P(X)P(Y)} \tag{2}$$

3.1.2 Dataset, Experiment Settings and Results

Given transactional data from NCR (*items_transactions.csv*), we first performed some preprocessing to filter out non food items and appropriately structure the data for the Apriori Algorithm. Then we chose our threshold by taking the smallest support count of the top ten percent support counts of individual items in the transactional data. After passing the data through our implementation of the Apriori Algorithm, we had a dataframe containing all frequent itemsets. Then, we calculated the lift values for all possible complementary relationships for itemsets of 2, 3, 4, and 5 and sorted them from high to low in terms of lift. The final output is these complementary relationships and their corresponding lifts.

To explore a simpler implementation, we repeated this process with only small transactions. We defined a small transaction to be a transaction with at most 10 items. The idea behind this approach is that small transactions may be more likely to carry more meaning. Therefore, it may not be necessary to use all transactions, and especially when given large transactional data, this small transaction approach may be preferred. Since we do not have a way of comparing the accuracy of each implementation, we decided to provide both. They can be used separately or together. For example, when it makes sense to recommend two items, one item could come from the all transactions implementation and one item could come from the small transactions implementation. We found 14,176 complementary relationships with lifts greater than or equal to 1.5 from the all transactions implementation and 460 complementary relationships with lifts greater than or equal to 1.5 from the small transactions implementation. The top five complementary relationships from each implementation are shown in Table 1.

3.2 Identify Substitute Items based on Transaction Co-occurrence Similarity

The first approach we used to identify the substitute relationships between items is a semantic-like approach. We regarded the other items appearing in the same transaction as the context of the target item. As for the substitute relationship, it satisfies that 1) the two items appear in similar context, which means they appear with a similar basket of other items, and 2) the two items rarely appear in the same transaction.

Table 1: Top 5 Complementary Relationships from Both Implementations

All Transactions		**Small Transactions**	
Complementary Relationship	**Lift**	**Complementary Relationship**	**Lift**
yellow bell pepper -> (green bell pepper, red bell pepper)	69.5	red bell pepper -> yellow bell pepper	180.6
espinazo fresco de puerco -> patitas frescas de puerco	69.0	green bell pepper -> red bell pepper	82.9
red bell pepper -> (green bell pepper, yellow bell pepper)	68.7	costillas de puerco -> trocitos frescos de puerco	48.8
red bell pepper -> yellow bell pepper	57.2	chayote squash -> mexican squash	44.1
chamorrow de res -> espinazo de res fresco	43.5	carrots -> chayote squash	34.6

3.2.1 Introduction of the Algorithm

As the core idea of this algorithm, We used co-occurrence matrix as a way to generate the context feature embedding of the target item. We firstly counted the pairwise co-occurrence among all items based on historical transaction logs. We proposed that a transaction with less number of items was more informative compared with larger-size transaction. So we assigned different weights on the co-occurrence according to the size of the transaction (As in Table 2).

Table 2: Co-occurrence Weights Based on Transaction Size

Number of items in transactions	**Number of transactions**	**Co-occurrence weights**
(0,5]	52,972	1.0
(5,10]	29,036	0.9
(10,20]	25,500	0.8
(20, ∞)	16,703	0.7

Then we defined the function to measure the substitute relationship of two targeted items i and j as

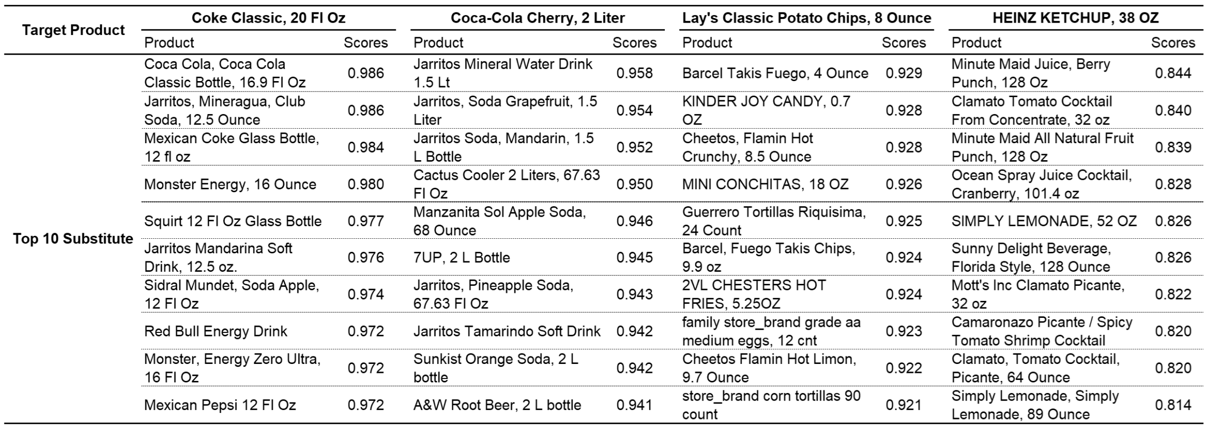
$$score = \frac{\boldsymbol{X_i} \cdot \boldsymbol{X_j}}{\|\boldsymbol{X_i}\|\|\boldsymbol{X_j}\|} \, e^{-\frac{C_{ij}}{\min(C_{ii}, C_{jj})}} \tag{3}$$

In Equation 3, We included the cosine similarity between the item embeddings and then added a penalty on the co-occurrence of the two targeted items. In detail, C_{ij} represents the co-occurrence between item i and item j ($i \neq j$). C_{ii} represents the frequency of item i appearing in all the transactions. $\boldsymbol{X_i}$ and $\boldsymbol{X_j}$ are the context feature embeddings of item i and item j, which contain all the co-occurrence values with other items except item i and item j. If the substitute score between two items is high, they are more likely to be substitute products with each other.

3.2.2 Dataset, Experiment Settings and Results

We experimented with this approach upon the transaction logs provided by NCR (*items_transactions.csv*). The total number of transactions in this dataset is 124,211 and the distribution of transaction size is shown in Table 2. There are 10,121 items appearing in the transaction logs. Thus we generated a co-occurrence matrix with shape of 10,121 by 10,121.

By applying the scoring function (Equation 3), we computed the pairwise substitute score among all items. Figure 10 illustrates the top 10 substitute products of some target products as examples. Intuitively,



Target Product	Coke Classic, 20 Fl Oz		Coca-Cola Cherry, 2 Liter		Lay's Classic Potato Chips, 8 Ounce		HEINZ KETCHUP, 38 OZ	
	Product	Scores	Product	Scores	Product	Scores	Product	Scores
Top 10 Substitute	Coca Cola, Coca Cola Classic Bottle, 16.9 Fl Oz	0.986	Jarritos Mineral Water Drink 1.5 Lt	0.958	Barcel Takis Fuego, 4 Ounce	0.929	Minute Maid Juice, Berry Punch, 128 Oz	0.844
	Jarritos, Mineragua, Club Soda, 12.5 Ounce	0.986	Jarritos, Soda Grapefruit, 1.5 Liter	0.954	KINDER JOY CANDY, 0.7 OZ	0.928	Clamato Tomato Cocktail From Concentrate, 32 oz	0.840
	Mexican Coke Glass Bottle, 12 fl oz	0.984	Jarritos Soda, Mandarin, 1.5 L Bottle	0.952	Cheetos, Flamin Hot Crunchy, 8.5 Ounce	0.928	Minute Maid All Natural Fruit Punch, 128 Oz	0.839
	Monster Energy, 16 Ounce	0.980	Cactus Cooler 2 Liters, 67.63 Fl Oz	0.950	MINI CONCHITAS, 18 OZ	0.926	Ocean Spray Juice Cocktail, Cranberry, 101.4 oz	0.828
	Squirt 12 Fl Oz Glass Bottle	0.977	Manzanita Sol Apple Soda, 68 Ounce	0.946	Guerrero Tortillas Riquisima, 24 Count	0.925	SIMPLY LEMONADE, 52 OZ	0.826
	Jarritos Mandarina Soft Drink, 12.5 oz.	0.976	7UP, 2 L Bottle	0.945	Barcel, Fuego Takis Chips, 9.9 oz	0.924	Sunny Delight Beverage, Florida Style, 128 Ounce	0.826
	Sidral Mundet, Soda Apple, 12 Fl Oz	0.974	Jarritos, Pineapple Soda, 67.63 Fl Oz	0.943	2VL CHESTERS HOT FRIES, 5.25OZ	0.924	Mott's Inc Clamato Picante, 32 oz	0.822
	Red Bull Energy Drink	0.972	Jarritos Tamarindo Soft Drink	0.942	family store_brand grade aa medium eggs, 12 cnt	0.923	Camaronazo Picante / Spicy Tomato Shrimp Cocktail	0.820
	Monster, Energy Zero Ultra, 16 Fl Oz	0.972	Sunkist Orange Soda, 2 L bottle	0.942	Cheetos Flamin Hot Limon, 9.7 Ounce	0.922	Clamato, Tomato Cocktail, Picante, 64 Ounce	0.820
	Mexican Pepsi 12 Fl Oz	0.972	A&W Root Beer, 2 L bottle	0.941	store_brand corn tortillas 90 count	0.921	Simply Lemonade, Simply Lemonade, 89 Ounce	0.814

Figure 10: Example of Substitute Relationships Identified based on Transaction Context

the results are plausible, especially in the categories where large number of similar products are competing and consumer's need and preference are not stable, like beverages and snacks.

Since we were lack of ways to evaluate the results systematically, we only checked the results generated by this approach manually according to the naive living knowledge and experience. However, we came up with this method in an intuitive way, which is logically reasonable. Further, we also explored a graph-based model beyond this naive approach to find the substitute relationships, which was proved feasible and valid in research [5, 6].

3.3 Graph-based Clustering

The second approach we used to identify the substitute relationships between items is a graph convolutional network (GCN) algorithm developed by reseachers at Target and known as GraphSWAG (Sample Weight and AGgregate). This algorithm uses efficient random walks and graph convolutions on weighted graphs to produce node embeddings [5].

3.3.1 Introduction of the Algorithm

This algorithm attempts to generate item recommendations by converting a transactional dataset into a graph and using a Graph Convolutional Network (GCN) to learn the embeddings for each nodes in the graph. In order to improve the quality of the recommendations it implements a new function for calculating the edge weights and include features from product images and textual descriptions on top of the graph structure. Since we did not have access to the data used by Target, we had to make modifications to the algorithm and try to implement it on our own.

Here are the steps of our implementation of the GraphSWAG Algorithm:

- **Generating the Graph:**
  - **Nodes:** Individual Products
    - Non-food items such as plastic bags were removed
    - Items without names were removed
  - **Edges:** Weighted Pairwise Co-Occurrence

        * Weights are calculated by adding co-occurrent transactions adjusted by time-decay and transforming them to a [0,1] range using the arctan function
    - **Word Embeddings:**
        * Item names were standardized by lower casing and removing special characters
        * Used Gensim's implementation of Word2Vec to convert item names to 200 dimensional vectors

$$w(i,j) = arctan\left(\sum_{t=1}^{t=N} \frac{1}{Rec(t)}\right)$$

$w(i,j)$: weight between products i and j
t: t_{th} co-occurrent transaction
$Rec(t)$:Recency of the t_{th} transaction in $days$

Figure 11: Modified edge weighting formula

- **Graph Convolutional Network:**
    - **Sampling:**
        * K neighbors of each node are sampled before each layer.
        * A beta hyperparameter assign a higher or lower importance to the edge weights when generating sampling probabilities.
    - **Aggregation:**
        * The feature vectors of all sampled neighbors of a node a combined with some aggregation function.
        * We used the Mean aggregator, similarly to the Target paper.
    - **Concatenation:**
        * Each aggregated feature vector is concatenated to the output of the previous layer before being input to the next layer.
    - **Non-Linear Activation Function:**
        * All hidden layers used the ReLu activation funciton
        * The final layer used the Softmax activation funciton
    - **Final Output:** 200-long feature vectors embedding each node of the graph.

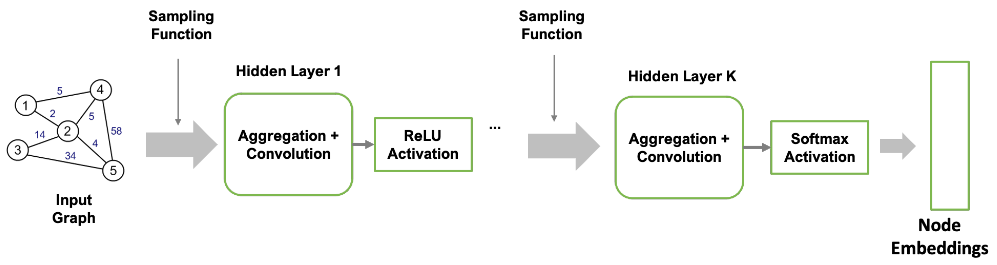
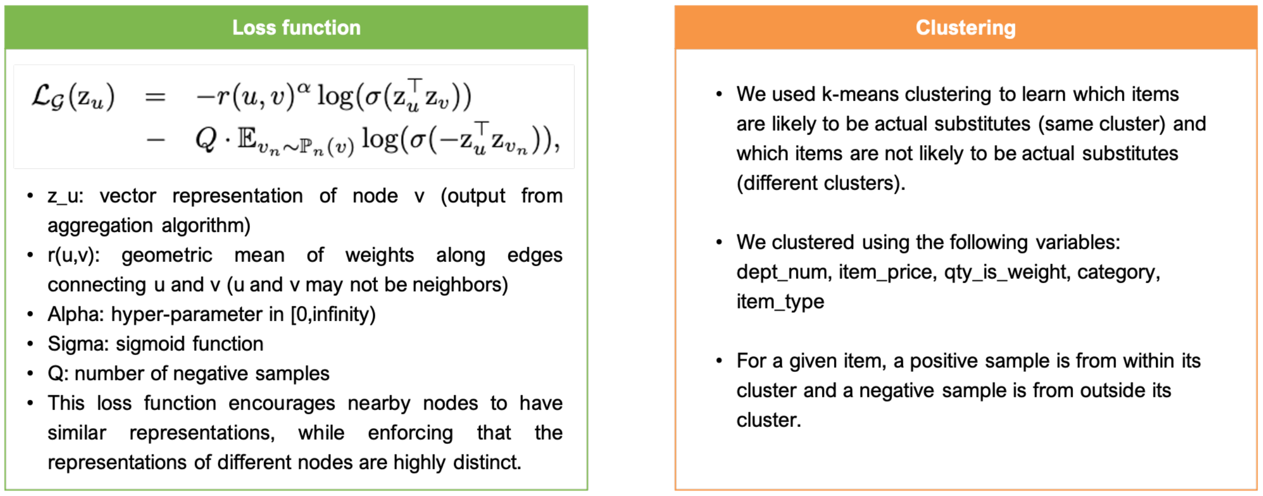


Figure 12: Architecture of the Graph Convolutional Network

- **Finding Item Recommendations:** The outputed features vectors are used to build a **Nearest Neighbors** models with a neighborhood of size n. The n neighbors of each product will represent its substitute recommendations.

3.3.2 Implementation and Preliminary Results

We successfully implemented functions to convert the transactional dataset into a graph containing edges weighted based on the function shown earlier. This graph was saved as an edge list and as a sparse adjecency matrix. We also implemented an initial version of the graph convolutional network where the graph gets sampled and aggregated before every layer and outputs the node embeddings at the end. However, we did not fully implement the training method for the network. Here is the current work being done on implementing the loss function:



Loss function

$$\mathcal{L}_{\mathcal{G}}(\mathrm{z}_u) = -r(u,v)^{\alpha} \log(\sigma(\mathrm{z}_u^{\top}\mathrm{z}_v)) - Q \cdot \mathbb{E}_{v_n \sim \mathbb{P}_n(v)} \log(\sigma(-\mathrm{z}_u^{\top}\mathrm{z}_{v_n})),$$

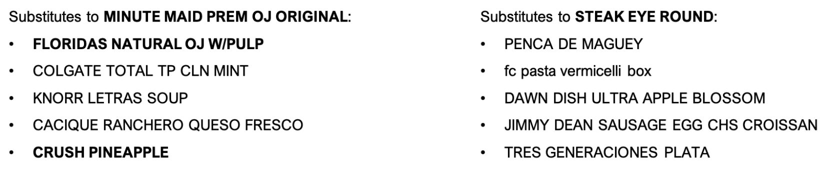
- z_u: vector representation of node v (output from aggregation algorithm)
- r(u,v): geometric mean of weights along edges connecting u and v (u and v may not be neighbors)
- Alpha: hyper-parameter in [0,infinity)
- Sigma: sigmoid function
- Q: number of negative samples
- This loss function encourages nearby nodes to have similar representations, while enforcing that the representations of different nodes are highly distinct.

Clustering

- We used k-means clustering to learn which items are likely to be actual substitutes (same cluster) and which items are not likely to be actual substitutes (different clusters).
- We clustered using the following variables: dept_num, item_price, qty_is_weight, category, item_type
- For a given item, a positive sample is from within its cluster and a negative sample is from outside its cluster.

Figure 13: Loss Function for GCN Training

We were able to conduct a trial run of the GraphSWAG algorithm with randomly generated weights for the GCN. While the quality of the recommendations was poor, we hope to improve it once the training method is fully implemented.



Substitutes to **MINUTE MAID PREM OJ ORIGINAL**:

- **FLORIDAS NATURAL OJ W/PULP**
- COLGATE TOTAL TP CLN MINT
- KNORR LETRAS SOUP
- CACIQUE RANCHERO QUESO FRESCO
- **CRUSH PINEAPPLE**

Substitutes to **STEAK EYE ROUND**:

- PENCA DE MAGUEY
- fc pasta vermicelli box
- DAWN DISH ULTRA APPLE BLOSSOM
- JIMMY DEAN SAUSAGE EGG CHS CROISSAN
- TRES GENERACIONES PLATA

Figure 14: Initial recommendations of the GraphSWAG algorithm with randomly generated weights for the GCN

3.3.3 Future Work

Some avenues for future work on our implementation are training Word2Vec, using more data, and tuning hyperparameters. We used Gensim's implementation of Word2Vec, which is pre-trained on the Google News dataset. As far as using more data, item descriptions and images could be used. The researchers at Target used these features in their implementation. We only used item names because we did not have item descriptions for most items, and we did not have any item images. If this data becomes available in the future, it can be incorporated into the model easily. Lastly, the following hyperparameters can be tuned: length of item vectors in Word2Vec, length of random walks, number of random walks, number of neighbors in the sampling algorithm, beta in the sampling algorithm, gamma in the aggregation algorithm, alpha in the loss function, number of layers in the network. Perhaps even better substitutes can be obtained by working in some or all of these areas.

References

[1] U. D. of Agriculture. “USDA Fooddata Central”. In: *https://fdc.nal.usda.gov/download-datasets.html* (2020).

[2] L. Akritidis et al. “A self-verifying clustering approach to unsupervised matching of product titles”. In: *Artificial Intelligence Review* (2020), pp. 1–44.

[3] I. Bank. “Optimized Cosine Similarity Library”. In: *https://github.com/ing-bank/sparse*$_d$*ot*$_t$*opn* (2020).

[4] Grocery.com. “Open Grocery Database Project”. In: *https://www.grocery.com/open-grocery-database-project/* (2020).

[5] A. Pande, K. Ni, and V. Kini. “SWAG: Item Recommendations using Convolutions on Weighted Graphs”. In: *2019 IEEE International Conference on Big Data (Big Data)*. IEEE. 2019, pp. 2903–2912.

[6] A. Pande et al. “Substitution Techniques for Grocery Fulfillment and Assortment Optimization Using Product Graphs”. In: *http://www.mlgworkshop.org/2020/papers/MLG2020*$_p$*aper*$_7$*.pdf* (2020).