Prediction and recommendation of Grocery Products

PROJECT REPORT

Submitted by

JACOB JOHN
(16BCE2205)

ARUNAVA DAS
(16BCE2026)

to
Prof. Geraldine Bessie Amali, SCOPE

in partial fulfilment for the award of the degree of
in
B.Tech, CSE

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
APRIL, 2019

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

TABLE OF CONTENTS

ABSTRACT

The three parts of the project include normal transactions, message notifications and recommender systems. The dataset created for normal transactions purpose includes all grocery details like grocery name, expiry date, MRP, etc. The Dataset used for recommender system included all the patterns in which buyers buy in grocery. In this application first of all the owners will be provided with options using which they can add the entry in the main database or can delete the entry from the main database. They can also update or search for the existing entries. In the same application buyer can select any product of his/her wish and add it to the cart. In addition to this, we will fit our model to predict a probability distribution across all words in the vocabulary. That means that we need to turn the output element from a single integer into a one hot encoding with a 0 for every word in the vocabulary and a 1 for the actual word that the value. This gives the network a ground truth to aim for from which we can calculate error and update the model.

Keywords: Single-value Decomposition, SQLite3, PythonGUI, OpenCV, Clique Percolation Method, Neural Language Model,Prediction algorithm, culturing Algorithms, bag of words

INTRODUCTION

Various departmental stores we have seen their suffering because of complexity and work overload they need to face. In order to decrease their suffering and increase their profit we are designing this smart grocery management system we are providing add, update and capabilities to the system so that the grocery manger will find it easier to handle the stuffs. Moreover, we have added the recommender system here which will help the manager to know beforehand what actually a customer may buy if he buys milk for example and then he will arrange the items in his grocery accordingly. Understanding that grocery store has a highly complex working system, like managing items in a group with similar items and also placing those group of items, etc. So, managing all these things require a high workload which is not suitable for some grocers as they have to hire some people and hence they earn less profit. This project can be used heavily in daily markets also because as this is a complete software-based product cost will also be less and even though will help in decreasing the complexity of grocery management system and increasing their profits. This can be easily applicable in almost all the groceries provided they have any software capabilities.

Introduction to dataset:

The dataset for this task is a social arrangement of records portraying clients'/customer's requests over a period time. The objective of the venture is to foresee which items will be in a client's next request. The dataset is anonymized and contains an example of more than 3 million basic supply orders from in excess of 200,000 Instacart clients. For every client, we give somewhere in the range of 4 and 100 of their requests, with the arrangement of items acquired in each request. It likewise gives the week and hour of day the request was set, and a relative proportion of time between requests.

SOFTWARE REQUIREMENTS-

- Python 3.7.2

Python modules which include-

- pandas and numpy for data manipulation
- turicreate for performing model selection and evaluation
- sklearn for splitting the data into train and test set
- keras

Existing System/approach/method

Seeing all the papers it seems that whatever grocery recommender system is developed till now is either working on hard codes (hard codes here refers to if else statement logic). Various recommendation system used nowadays actually uses prediction algorithms like support vector machines or SVM.

SVM actually works on the support vectors, what this actually means that suppose if we have two classes A and B then SVM mainly works on those A which are similar and those B which are similar to A**.** It works on the concept of hyperplane. When given the inputs SVM proposes a most optimal hyperplane used for classifying new examples. In two-dimensional cases hyperplane will be a simple straight line and thus acts as a decision boundary in this case.

Some recommender systems also make use of clustering algorithm such as K-Means or **hierarchical clustering**. These clustering techniques can be used to clusters the various products on the basis of some parameters. So, whenever it is asked what I should buy if I have already bought this system will just identify the cluster and thus give the answers accordingly.

K-Means is a sort of unsupervised realizing, or, in other words you have unlabelled information (i.e., information without characterized classes or gatherings). The objective of this calculation is to discover bunches in the information, with the quantity of gatherings spoken to by the variable K. The calculation works iteratively to dole out every datum point to one of K bunches dependent on the highlights that are given. Information focuses are grouped dependent on highlight comparability.

Other various systems, for example, Nearest Neighbourhood, Trust and Clustering have been utilized in Recommender Systems. Communitarian Filtering (CF), the best-known kind of Nearest-Neighbourhood, has as basic thought the concession to taste of individuals for anticipating their future enjoying on new things.

clustering has been generally examined in software engineering as an unsupervised learning technique. When all is said in done, separating the huge networks of clients into littler sets (bunches) has appeared to offer preferences, such as adaptability, which therefore enhances the reaction time, due to the littler set of information that calculations work on.

The most common among recommender system is *Latent factor Recommender System*. The main idea of latent factor optimization problem as an optimization problem [1], we think of recommendation as ranking/rating problem and optimization as the method that gives us best

rating prediction. The basic idea is the ***root-mean square error***, the smaller it's the better recommendation system is.

So, we use the idea of SVD(Single Value decomposition) where we will take a matrix say R(which will define the prediction a customer will buy a particular product of a grocery and we take dot product of the individual customer rankings not only to particular products but categorical product attributes matrix, with the grocery product matrix.

Purpose of SVD-Because SVD gives minimum reconstruction error (*Sum of squared errors*)

between true value and predicted value:

$$\min_{U,V,\Sigma} \sum_{ij \in A} \left(A_{ij} - [U\Sigma V^{\mathrm{T}}]_{ij}\right)^2$$

here A ij is actual and $[U\sum V^T]ij$ is the predicted matrix

Now, ***root mean squared error*** and *Sum of squared errors* are monotonically related:

RMSE = (1/c).root(SSE) , c be some constant- number of data points

Hence, SVD is basically minimizing RMSE.

The gaps that are found in traditional SVD is: The sum in SVD error term is over all entries (no-ranking is interpreted as zero-ranking). But our grocery prediction matrix R has missing entries. In other words, SVD is not defined when entries are missing.

New things can be done based on this grocery dataset based on the papers [9]

While going through whatisdbms.com it was observed that with some benefits, SQL has one of the disadvantages of its difficult interface, it has some complex interface that can be difficult for many users to use it. So, to make the work of user easy, this database was created in python Tkinter. Which provides programmer to program a Graphical User Interface for user, so that he can handle database easily without any difficulty. Also, to use benefits of SQL, sqlite3 has been used along with python so that advantages of SQL will not be overseen at. In this GUI, user will not have to enter whole code to enter an entry or to remove an entry or to modify any entry. He can do it with just the click of that particular button, instead of memorising the code and typing it again and again, instead of him, python will take care of writing the code in sqlite3, the codes are basic SQL codes. Also, SQLite has its own many benefits, like mentioned in its website.

Gaps identified and observations made from the literature survey: -

- First of all, Soft computing, in contrast to traditional computing, deals only with various approximate models and thus the solutions to multiple complex real-life problems. In last, the role model for soft computing is the human brain neuron. Soft computing is

based on techniques such as fuzzy logic, genetic algorithms, artificial neural networks, machine learning, and expert systems

- It has been observed that in the cases where we need just one single boundary either linear or nonlinear to classify between the classes SVM works well otherwise it gives very ransom results which are of no use.

- Moreover, SVM will work well only for smaller dataset as we increase the dataset length with few thousands it stopped giving good results

- In the k means clustering it is a kind of difficult to find the optimal number of clusters. We generally use elbow method in order to find the optimal number of clusters

- Initialization parameters also affect the final results for this purpose we use k-means++ even then also we are not able to achieve appreciable results.

- Results obtained from k means are also sensitive to scaling and normalization

- Hierarchical clustering is also not applicable for larger dataset, moreover this technique is very much sensitive to outliers.

In traditional recommender system for large dataset Latent factor recommender system and Single value decomposition for utility matrix finding the appropriate features is hard and most importantly over-specialization.

It never recommends items outside user's content profile like if one does buy any product in our case say pasta, he/she may be never recommended sauce of different kind in this case and infact people might have multiple interests and it can't or is unable to exploit quality judgments of other users say like for a store/site the most bought popular grocery is rice, but if utility matrix is proposed based on mapping buying list of users to product, rice can never be recommended to the particular customer.

And one more gap is that cold-start problem for new users, like one customer is new to the site, then he/she might have nothing recommended. So, the popular list might not come up due to the problem of overspecialization.

Proposed Model

First Proposed Algorithm:

Clique Percolation Method-

The underlying idea of this method is the concept of a k-clique community which was defined as the union of all k-cliques (complete subgraphs of size k) that can be reached from each other through a series of adjacent k-cliques (where adjacency means sharing k-1 vertices). The k-clique community can be considered as a usual module (community, cluster or complex) because of its dense internal links and sparse external linkage with other part of the whole network. All of k-cliques of network can be received by iterative recursion, then construct the

overlap matrix of these k-cliques. Finally, a number of k-clique communities are discovered by analysis the overlap matrix.
Our goal here is to break down each list of items in the products column into rows and count the number of products bought by a user.

Two datasets are used in this exercise, which can be found in data folder:

- recommend_1.csv consisting of a list of 1000 customer IDs to recommend as output
- trx_data.csv consisting of user transactions

In [0]:
```
import io

customers = pd.read_csv(io.BytesIO(uploaded['recommend_1.csv']))
transactions = pd.read_csv(io.BytesIO(uploaded['trx_data.csv']))
```

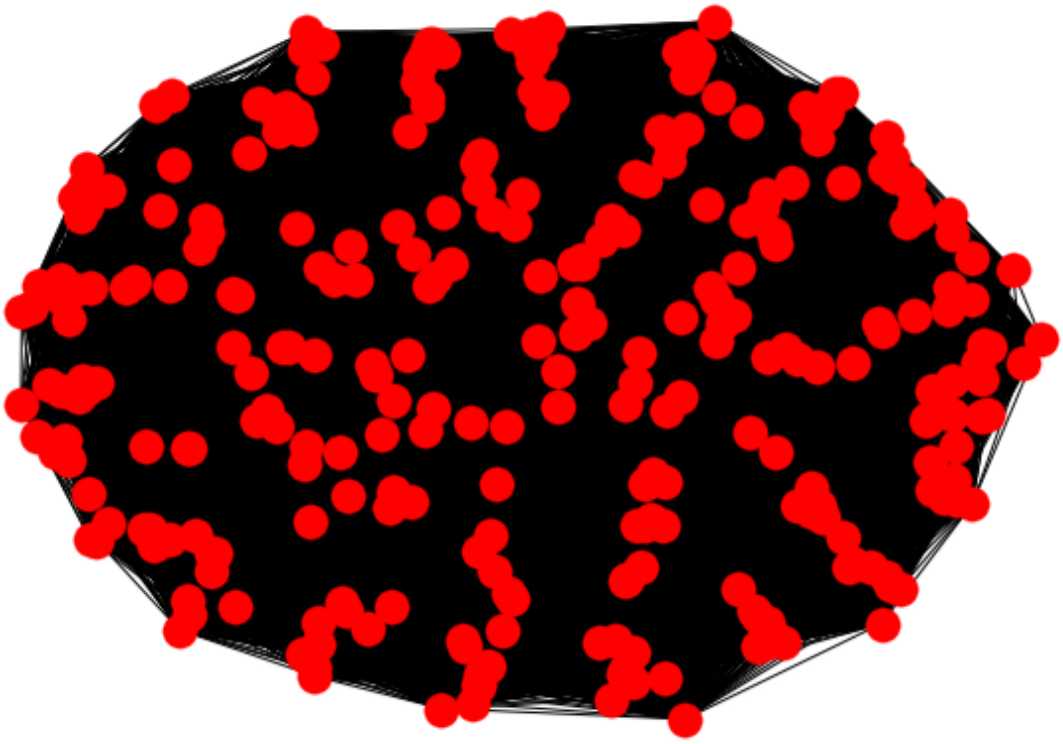
In [5]:
```
print(customers.shape)
customers.head()
```

```
(1000, 1)
```

Out[5]:

	customerId
0	1553
1	20400
2	19750
3	6334
4	27773

Next, Our goal here is to break down each list of items in the products column into rows and count the number of products bought by a user and create a networkX graph.



Here, we apply Clique Percolation method on NetworkX graph to generate max cliques. However, with this approach we can conclude that Clique percolation method would not work since Graph has too many nodes.

We use associate market basket analysis to generate the rules for grocery prediction. We see that the top associations are not surprising, with one flavor of an item being purchased with another flavor from the same item family (eg: Strawberry Chia Cottage Cheese with Blueberry Acai Cottage Cheese, Chicken Cat Food with Turkey Cat Food, etc). As mentioned, one common application of association rules mining is in the domain of recommender systems. Once item pairs have been identified as having positive relationship, recommendations can be made to customers in order to increase sales. And hopefully, along the way, also introduce customers to items they never would have tried before or even imagined existed!

Second Proposed Algorithm:

Word-Based Neural Language Model in Python with Keras

In this method, The first step is to encode the text as integers. Each lowercase word in the source text is assigned a unique integer and we can convert the sequences of words to sequences of integers.
Keras provides the Tokenizer class that can be used to perform this encoding. First, the Tokenizer is fit on the source text to develop the mapping from words to unique integers. Then sequences of text can be converted to sequences of integers by calling the texts_to_sequences() function.

```
In [0]: # source text
        data2 = """ Jack and Jill went up the hill\n
                    To fetch a pail of water\n
                    Jack fell down and broke his crown\n
                    And Jill came tumbling after\n """

        # integer encode text
        tokenizer3 = Tokenizer()
        tokenizer3.fit_on_texts([data2])
        encoded3 = tokenizer3.texts_to_sequences([data2])[0]
```

We will need to know the size of the vocabulary later for both defining the word embedding layer in the model, and for encoding output words using a one hot encoding. The size of the vocabulary can be retrieved from the trained Tokenizer by accessing the word_index attribute.

```
In [58]: # determine the vocabulary size
         vocab_size3 = len(tokenizer3.word_index) + 1
         print('Vocabulary Size: %d' % vocab_size3)
```

Running this example, we can see that the size of the vocabulary is 21 words. We add one, because we will need to specify the integer for the largest encoded word as an array index, e.g. words encoded 1 to 21 with array indices 0 to 21 or 22 positions. Next, we need to create sequences of words to fit the model with one word as input and one word as output.

In [59]:
```
# create word -> word sequences
sequences3 = list()
for i in range(1, len(encoded3)):
        sequence3 = encoded3[i-1:i+1]
        sequences3.append(sequence3)
print('Total Sequences: %d' % len(sequences3))
```

We can then split the sequences into input (X) and output elements (y). This is straightforward as we only have two columns in the data.
We will fit our model to predict a probability distribution across all words in the vocabulary. That means that we need to turn the output element from a single integer into a one hot encoding with a 0 for every word in the vocabulary and a 1 for the actual word that the value. This gives the network a ground truth to aim for from which we can calculate error and update the model. Keras provides the to_categorical() function that we can use to convert the integer to a one hot encoding while specifying the number of classes as the vocabulary size.

In [0]:
```
# one hot encode outputs
y3 = to_categorical(y3, num_classes=vocab_size3)
```

We are now ready to define the neural network model.

The model uses a learned word embedding in the input layer. This has one real-valued vector for each word in the vocabulary, where each word vector has a specified length. In this case we will use a 10-dimensional projection. The input sequence contains a single word, therefore the input_length=1.

The model has a single hidden LSTM layer with 50 units. This is far more than is needed. The output layer is comprised of one neuron for each word in the vocabulary and uses a softmax activation function to ensure the output is normalized to look like a probability.

In [62]:
```
# define model
model3 = Sequential()
model3.add(Embedding(vocab_size3, 10, input_length=1))
model3.add(LSTM(50))
model3.add(Dense(vocab_size3, activation='softmax'))
print(model3.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_6 (Embedding)      (None, 1, 10)             220
_________________________________________________________________
lstm_6 (LSTM)                (None, 50)                12200
_________________________________________________________________
dense_6 (Dense)              (None, 22)                1122
=================================================================
Total params: 13,542
Trainable params: 13,542
Non-trainable params: 0
_________________________________________________________________
None
```

We will use this same general network structure for each example in this tutorial, with minor changes to the learned embedding layer.

Next, we can compile and fit the network on the encoded text data. Technically, we are modeling a multi-class classification problem (predict the word in the vocabulary), therefore using the categorical cross entropy loss function. We use the efficient Adam implementation

of gradient descent and track accuracy at the end of each epoch. The model is fit for 500 training epochs, again, perhaps more than is needed.

The network configuration was not tuned for this and later experiments; an over-prescribed configuration was chosen to ensure that we could focus on the framing of the language model.

After the model is fit, we test it by passing it a given word from the vocabulary and having the model predict the next word. Here we pass in 'Jack' by encoding it and calling model.predict_classes() to get the integer output for the predicted word. This is then looked up in the vocabulary mapping to give the associated word.

In [70]:
```
# evaluate
print(generate_seq2(model3, tokenizer3, 'Jack', 6))
```

```
Jack and jill went up the hill
```

Third Approach

Line-by-Line Sequence

Another approach is to split up the source text line-by-line, then break each line down into a series of words that build up. This approach may allow the model to use the context of each line to help the model in those cases where a simple one-word-in-and-out model creates ambiguity. In this case, this comes at the cost of predicting words across lines, which might be fine for now if we are only interested in modeling and generating lines of text.

Note that in this representation, we will require a padding of sequences to ensure they meet a fixed length input. This is a requirement when using Keras. First, we can create the sequences of integers, line-by-line by using the Tokenizer already fit on the source text.

In [0]:
```
data = ""
indexset = set()

for i in orders.index:
  if len(indexset) < 2000:
    for itemid in orders[i].data:
      data += item_name.loc[itemid].item_name + " "
      indexset.add(itemid)
    data += "\n"

vocab_size = len(indexset)
```

In [75]:
```
# prepare the tokenizer on the source text
tokenizer = Tokenizer()
tokenizer.fit_on_texts([data])

# determine the vocabulary size
vocab_size = len(tokenizer.word_index) + 1
print('Vocabulary Size: %d' % vocab_size)

# create line-based sequences
sequences = list()
for line in data.split('\n'):
        encoded = tokenizer.texts_to_sequences([line])[0]
        for i in range(len(encoded)-2, len(encoded)):
                sequence = encoded[:i+1]
                sequences.append(sequence)
print('Total Sequences: %d' % len(sequences))
```

Vocabulary Size: 2450
Total Sequences: 6090

Next, we can pad the prepared sequences. We can do this using the pad_sequences() function provided in Keras. This first involves finding the longest sequence, then using that as the length by which to pad-out all other sequences.

The model can then be defined as before, except the input sequences are now longer than a single word. Specifically, they are max_length-1 in length, -1 because when we calculated the maximum length of sequences, they included the input and output elements. We can use the model to generate new sequences as before. The generate_seq() function can be updated to build up an input sequence by adding predictions to the list of input words each iteration.

In [81]:
```
# evaluate model
print(generate_seq(model, tokenizer, max_length-1, 'Strawberry', 4))
print(generate_seq(model, tokenizer, max_length-1, 'sandwiches', 4))
```

```
Strawberry drink bags carrots potatoes
sandwiches ranchera filberts soup santa
```

RESULTS AND DISCUSSION

For clique percolation method, a network graph is created using the NetworkX library and associated rule function is defined.

3.2 Association rules function

In [0]:
```
def association_rules(order_item, min_support):

    print("Starting order_item: {:22d}".format(len(order_item)))


    # Calculate item frequency and support
    item_stats             = freq(order_item).to_frame("freq")
    item_stats['support']  = item_stats['freq'] / order_count(order_item) * 100


    # Filter from order_item items below min support
    qualifying_items       = item_stats[item_stats['support'] >= min_support].index
    order_item             = order_item[order_item.isin(qualifying_items)]

    print("Items with support >= {}: {:15d}".format(min_support, len(qualifying_items)))
    print("Remaining order_item: {:21d}".format(len(order_item)))


    # Filter from order_item orders with less than 2 items
    order_size             = freq(order_item.index)
    qualifying_orders      = order_size[order_size >= 2].index
    order_item             = order_item[order_item.index.isin(qualifying_orders)]

    print("Remaining orders with 2+ items: {:11d}".format(len(qualifying_orders)))
    print("Remaining order_item: {:21d}".format(len(order_item)))
```

In [13]:
```
# Replace item ID with item name and display association rules
item_name   = pd.read_csv('products.csv')
item_name   = item_name.rename(columns={'product_id':'item_id', 'product_name':'item_name'})
rules_final = merge_item_name(rules, item_name).sort_values('lift', ascending=False)
display(rules_final)
```

	itemA	itemB	freqAB	supportAB	freqA	supportA	freqB	supportB	confidenceAtoB
0	Organic Strawberry Chia Lowfat 2% Cottage Cheese	Organic Cottage Cheese Blueberry Acai Chia	306	0.010155	1163	0.038595	839	0.027843	0.263113
1	Grain Free Chicken Formula Cat Food	Grain Free Turkey Formula Cat Food	318	0.010553	1809	0.060033	879	0.029170	0.175788
3	Organic Fruit Yogurt Smoothie Mixed Berry	Apple Blueberry Fruit Yogurt Smoothie	349	0.011582	1518	0.050376	1249	0.041449	0.229908
9	Nonfat Strawberry With Fruit On The Bottom Gre...	0% Greek, Blueberry on the Bottom Yogurt	409	0.013573	1666	0.055288	1391	0.046162	0.245498
10	Organic Grapefruit Ginger Sparkling Yerba Mate	Cranberry Pomegranate Sparkling Yerba Mate	351	0.011648	1731	0.057445	1149	0.038131	0.202773

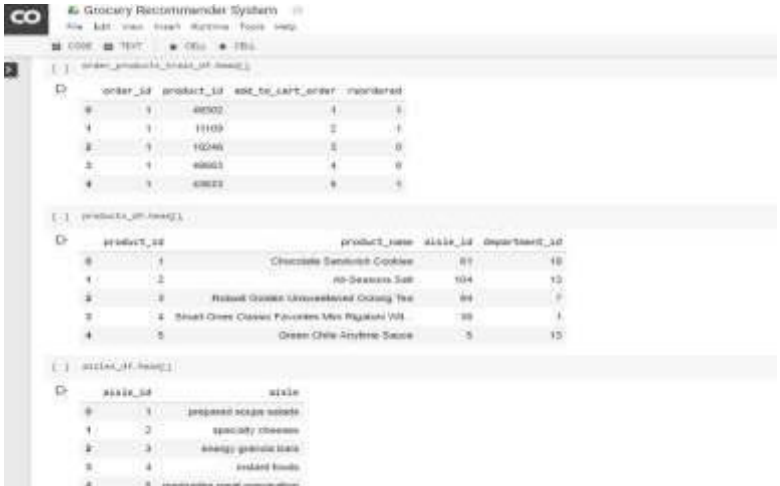
In [73]:
```
item_name = pd.read_csv('products.csv')
item_name = item_name.rename(columns={'product_id':'item_id', 'product_name':'item_name'})
item_name.tail()
```

Out[73]:

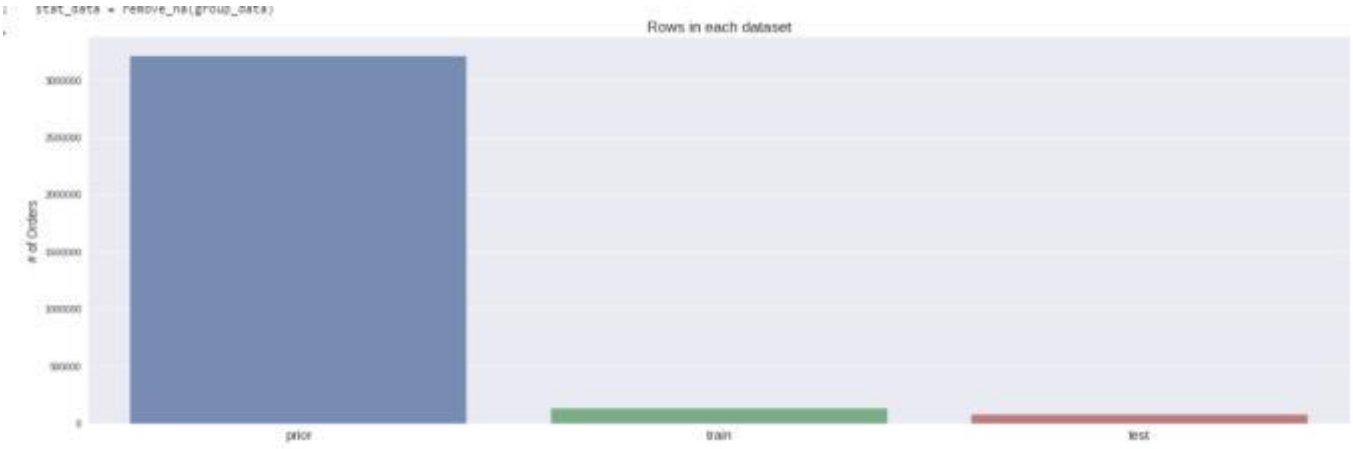
	item_id	item_name	aisle_id	department_id
49683	49684	Vodka, Triple Distilled, Twist of Vanilla	124	5
49684	49685	En Croute Roast Hazelnut Cranberry	42	1
49685	49686	Artisan Baguette	112	3
49686	49687	Smartblend Healthy Metabolism Dry Cat Food	41	8
49687	49688	Fresh Foaming Cleanser	73	11

Observations made from the data visualization: -
Each entity (customer, product, order, aisle, etc.) has an associated unique id. Most of the files and variable names should be self-explanatory.

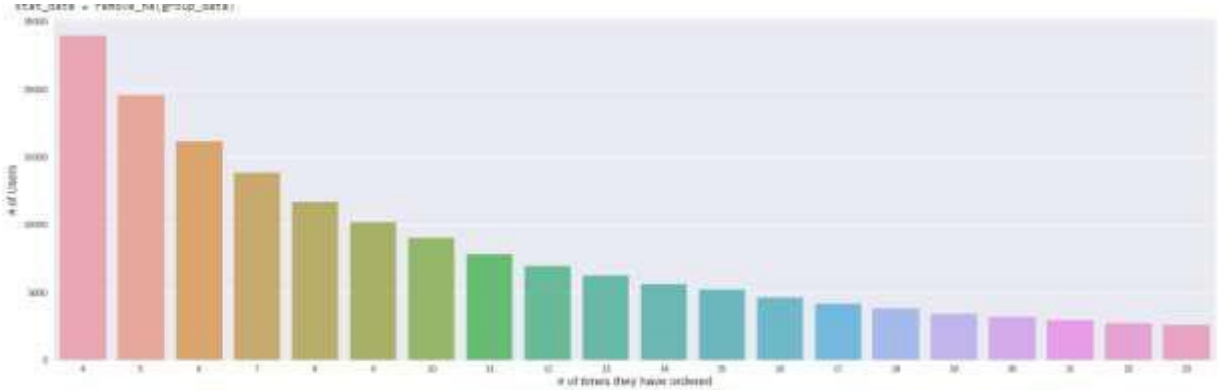
The data tables:



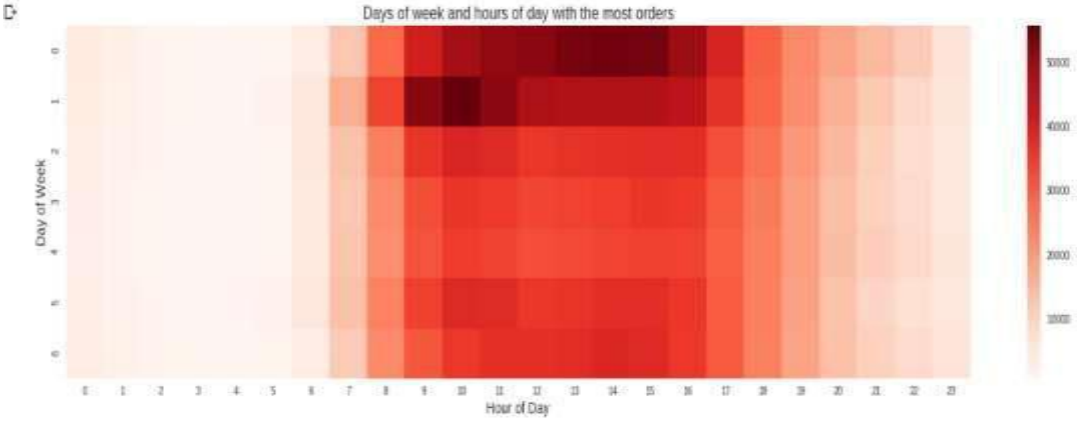
Rows in each dataset tables:



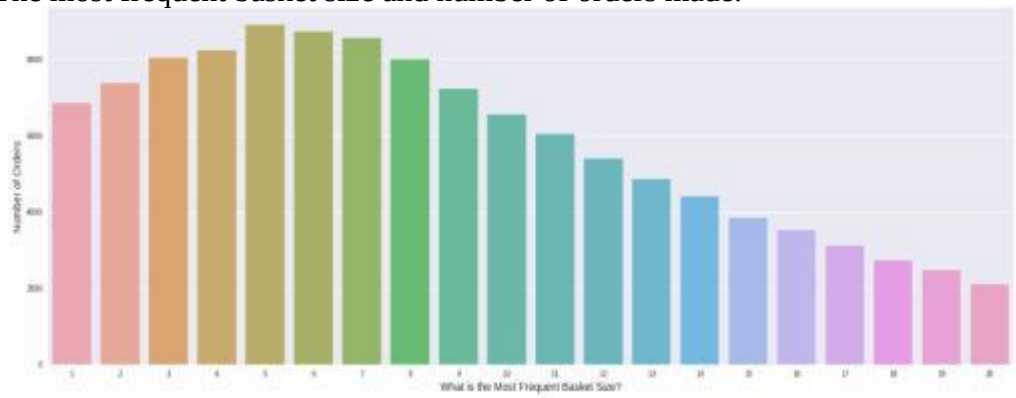
The visualization of number of users have bought a product of how many times.



The following heatmap shows the days and hours of week for which the customer buys a product most, this helped in maintaining the distribution of data while manipulating for our purpose:



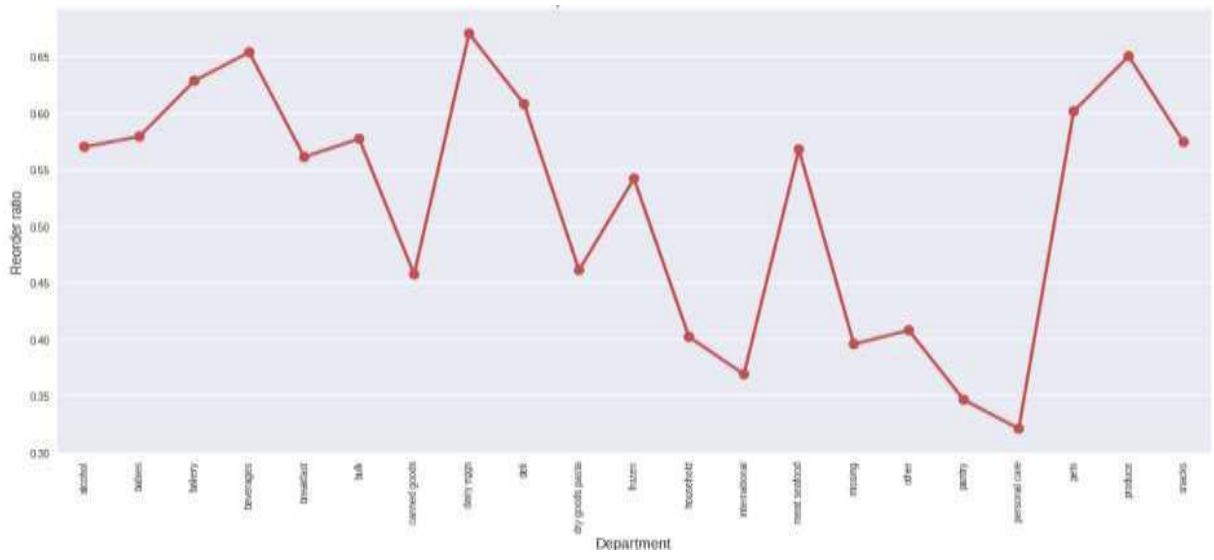
The most frequent basket size and number of orders made:



And we took up the items which is bought together frequently viz.:

	product_name	frequency_count
0	Banana	472565
1	Bag of Organic Bananas	379450
2	Organic Strawberries	264683
3	Organic Baby Spinach	241921
4	Organic Hass Avocado	213584
5	Organic Avocado	176815
6	Large Lemon	152657
7	Strawberries	142951
8	Limes	140627
9	Organic Whole Milk	137905
10	Organic Raspberries	137057
11	Organic Yellow Onion	113426
12	Organic Garlic	109778
13	Organic Zucchini	104823
14	Organic Blueberries	100060
15	Cucumber Kirby	97315
16	Organic Fuji Apple	89632
17	Organic Lemon	87746
18	Apple Honeycrisp Organic	85020
19	Organic Grape Tomatoes	84255

And finally the products which are bought most frequently in re-order ration on scale 0-1 plotted as:



CONCLUSION

So, we have designed our final end application which is ready to be used by owner of the grocery as well as by the buyer frequent to that grocery shop. Owner can perform daily functions like adding/deleting an entry or updating an existing an entry etc. In this same application smart recommender system is also there for buyer. From the three proposed models we have concluded that Word based neural language model works for large dataset provided the proper statistical distribution is maintained but not much in case of zero product bought because of the case of Singular value decomposition which is basically inversion of matrix with zero eigen values.

However, our proposed model though not work well for large dataset, however it can be taken for no rating which is in case for no item bought here. It doesn't specifically requires any distribution of dataset to be in and it is found to compute score fastest which in comparison to Eclat which takes lot amount of time. Further improvement idea is a Language Processing Model: From the idea taken from [3] in Ngrams feature for word we have basically can be thought of these ways:

1) Find the bigrams using pyspark.ml Library's-> N-gram module.
2) Count frequency.
3) Bigrams are stored in nested dictionary.
4) first layer key is the first word in a bigram.
5) Second layer key is the second word in a bigram.
6) The second layer value is the frequency, eg:{'organic_mint_bunch': {'organic_navel_orange':2, 'c':2}}
7) Sort the bigram frequencies in descending order, then return the corresponding product names in the same order.
8) For recommending products →
8.1 Recommend products□empty_list initialize
8.2 Run a for loop in sorted_data
8.3 Check if value of k, which is some set of confidence limit set agains the total no of pairs of products bought together with (let's say no of recommend variable)
8.4 If yes, then append k → products to Recommend products, decrease the no of products commodity forms bigram with(i.e. no of recommend)
8.5 Else recommend products→ random_sample(k,no of recommend)
8.6 Return recommend_product.
9) END

One more functionality that we can add id an ads optimization capability. The main idea behind this is that whenever a company launches a set of ads it doesn't which will benefit them the most. So we need to do the exploration as well as exploitation of the dataset at the same time. In order to achieve this we can reinforcement learning techniques like upper confidence bound etc.

REFERENCE

[1] Eirinaki, Gao, Varlamis, Tserpes (2018) Recommender Systems for Large-Scale Social Networks: A review of challenges and solutions, ELESVIER, Future Generation Computer Systems 78 (2018) 413–418

[2] *Shaikh, Rathi, Janrao (2017):* Recommendation system in E-commerce websites: AGraph Based Approached, 2017 IEEE 7th International Advance Computing Conference

[3] Zhang, Liu, Zeng (2017): Timeliness in recommender systems, ELSEVIER, Expert Systems With Applications 85 (2017) 270–278

[4] Sidorov, Velasquez, Stamatatos, Gelbukh, Chanona-Hernández (2014): Syntactic N-grams as machine learning features for natural language processing, ELSEVIER Expert Systems with Applications 41 (2014) 853–860.

[5] Title: Comparing Dataset Characteristics that Favor the Apriori, Eclat or FP-Growth Frequent Itemset Mining Algorithms Authors: Jeff Heaton College of Engineering and Computing Published: Nova Southeastern University, 2017

[6] Title: A novel hybrid based recommendation system based on clustering, association mining Authors: S. Pandya ; J. Shah ; N. Joshi ; H. Ghayvat ; S. C. Mukhopadhyay ; M. H. Yap Published: 2016 10th International Conference on Sensing Technology (ICST)

[7] Title: Implementation of a Recommendation System using Association Rules and Collaborative Filtering Authors: JinHyun Jooa, SangWon Bangb, GeunDuk Parka Published: Information Technology and Quantitative Management (ITQM 2016)

[8] Title: A recommendation engine by using association rules Authors: Ozgur Cakira 1, Murat Efe Aras b Published: WCBEM 2012

[9] Title: Advanced eclat algorithm for frequent itemsets generation Authors: M. Kaur, Urvashi Garg, S. Kaur Published: January 2015 International Journal of Applied Engineering Research

[10] Title: Grocery Stop database management tool Author: Tapan Desai www.slideshare.net/tapandesai2992/grocery-station

[11] www.sqlite.org/aff_short.html

[12]Jun Yang, Zhonghua Li, Wei Xiang School of Computer Science, Leshan Normal University, Leshan, Sichuan 614000, China Lstc_yy@163.com, Luxin Xiao The Dean's Office, Leshan Normal University, Leshan, Sichuan 614000, China , An Improved Apriori Algorithm Based on Features,2016, 10.1109/CIS.2013.33/https://ieeexplore.ieee.org/document/6746369/

[13] Yang Xi ,Qi Yuan Intelligent Recommendation Scheme of Scenic Spots Based on Association Rule Mining Algorithm , 2017 International Conference on Robots & Intelligent System (ICRIS), 10.1109/ICRIS.2017.53/https://ieeexplore.ieee.org/document/8101376/

[14] C Valliyammai, MIT Campus, Anna University, Chennai, India, R PrasannaVenkatesh MIT Campus, Anna University, Chennai, India, C Vennila, MIT Campus, Anna University, Chennai, India, S Gopi Krishnan MIT Campus, Anna University, Chennai, India /An intelligent personalized recommendation for travel group planning based on reviews/2016 Eighth International Conference on Advanced (ICoAC)/ 10.1109/ICoAC.2017.7951747/ https://ieeexplore.ieee.org/document/7951747/

APPENDIX:

Prediction groceries using a Word-Based Neural Language Model in Python with Keras

CODE

```
]:
from numpy import array
from keras.preprocessing.text import Tokenizer
from keras.utils import to_categorical
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Embedding

# generate a sequence from the model
def generate_seq2(model, tokenizer, seed_text, n_words):
     in_text, result = seed_text, seed_text
     # generate a fixed number of words
     for _ in range(n_words):
          # encode the text as integer
          encoded = tokenizer.texts_to_sequences([in_text])[0]
          encoded = array(encoded)
          # predict a word in the vocabulary
          yhat = model.predict_classes(encoded, verbose=0)
          # map predicted word index to word
          out_word = ''
          for word, index in tokenizer.word_index.items():
               if index == yhat:
                    out_word = word
                    break
          # append to input
          in_text, result = out_word, result + ' ' + out_word
     return result

# generate a sequence from a language model
def generate_seq(model, tokenizer, max_length, seed_text,
n_words):
```

```
	in_text = seed_text
	# generate a fixed number of words
	for _ in range(n_words):
		# encode the text as integer
		encoded = tokenizer.texts_to_sequences([in_text])[0]
		# pre-pad sequences to a fixed length
		encoded = pad_sequences([encoded],
maxlen=max_length, padding='pre')
		# predict probabilities for each word
		yhat = model.predict_classes(encoded, verbose=0)
		# map predicted word index to word
		out_word = ''
		for word, index in tokenizer.word_index.items():
			if index == yhat:
				out_word = word
				break
		# append to input
		in_text += ' ' + out_word
	return in_text

# source text
data2 = """ Jack and Jill went up the hill\n
		To fetch a pail of water\n
		Jack fell down and broke his crown\n
		And Jill came tumbling after\n """

# integer encode text
tokenizer3 = Tokenizer()
tokenizer3.fit_on_texts([data2])
encoded3 = tokenizer3.texts_to_sequences([data2])[0]

# determine the vocabulary size
vocab_size3 = len(tokenizer3.word_index) + 1
print('Vocabulary Size: %d' % vocab_size3)

# create word -> word sequences
sequences3 = list()
for i in range(1, len(encoded3)):
	sequence3 = encoded3[i-1:i+1]
	sequences3.append(sequence3)
print('Total Sequences: %d' % len(sequences3))


# split into X and y elements
sequences3 = array(sequences3)
X3, y3 = sequences3[:,0],sequences3[:,1]

# one hot encode outputs
y3 = to_categorical(y3, num_classes=vocab_size3)

# define model
```

```
model3 = Sequential()
model3.add(Embedding(vocab_size3, 10, input_length=1))
model3.add(LSTM(50))
model3.add(Dense(vocab_size3, activation='softmax'))
print(model3.summary())

# compile network
model3.compile(loss='categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])

# fit network
model3.fit(X3, y3, epochs=500, verbose=2)

# evaluate
print(generate_seq2(model3, tokenizer3, 'Jack', 6))
```

Grocery prediction using clique percolation method

```
import pandas as pd
import numpy as np
import time
import turicreate as tc
from sklearn.model_selection import train_test_split
from google.colab import files
uploaded = files.upload()
import io

customers =
pd.read_csv(io.BytesIO(uploaded['recommend_1.csv']))
transactions =
pd.read_csv(io.BytesIO(uploaded['trx_data.csv']))
print(customers.shape)
customers.head()
print(transactions.shape)
transactions.head()

# example 2: organize a given table into a dataframe with
customerId, single productId, and purchase count
pd.melt(transactions.head(2).set_index('customerId')['products
'].apply(pd.Series).reset_index(),
             id_vars=['customerId'],
             value_name='products') \
    .dropna().drop(['variable'], axis=1) \
    .groupby(['customerId', 'products']) \
    .agg({'products': 'count'}) \
    .rename(columns={'products': 'purchase_count'}) \
    .reset_index() \
    .rename(columns={'products': 'productId'})
```

```
s=time.time()

data =
pd.melt(transactions.set_index('customerId')['products'].apply
(pd.Series).reset_index(),
             id_vars=['customerId'],
             value_name='products') \
    .dropna().drop(['variable'], axis=1) \
    .groupby(['customerId', 'products']) \
    .agg({'products': 'count'}) \
    .rename(columns={'products': 'purchase_count'}) \
    .reset_index() \
    .rename(columns={'products': 'productId'})
data['productId'] = data['productId'].astype(np.int64)

print("Execution time:", round((time.time()-s)/60,2),
"minutes")

import networkx as nx

G=nx.Graph()

print(G.nodes())
print(G.edges())

print(type(G.nodes()))
print(type(G.edges()))

from networkx.algorithms.approximation import clique

c1 = clique.max_clique(G)
print(c1)

# Returns frequency counts for items and item pairs
def freq(iterable):
    if type(iterable) == pd.core.series.Series:
        return iterable.value_counts().rename("freq")
    else:
        return pd.Series(Counter(iterable)).rename("freq")


# Returns number of unique orders
def order_count(order_item):
    return len(set(order_item.index))


# Returns generator that yields item pairs, one at a time
def get_item_pairs(order_item):
    order_item = order_item.reset_index().as_matrix()
    for order_id, order_object in groupby(order_item, lambda
x: x[0]):
```

```
        item_list = [item[1] for item in order_object]

        for item_pair in combinations(item_list, 2):
            yield item_pair


# Returns frequency and support associated with item
def merge_item_stats(item_pairs, item_stats):
    return (item_pairs
                .merge(item_stats.rename(columns={'freq':
'freqA', 'support': 'supportA'}), left_on='item_A',
right_index=True)
                .merge(item_stats.rename(columns={'freq':
'freqB', 'support': 'supportB'}), left_on='item_B',
right_index=True))
```