Smart Shopping List

TDDE19 - Report

Martin Persson
marpe902@student.liu.se

Sabina Serra
sabse455@student.liu.se

Rolf Sievert
rolsi701@student.liu.se

Erik Svenson
erisv795@student.liu.se

Pontus Svensson
ponsv690@student.liu.se

Jon Vik
jonvi039@student.liu.se

Contents

1 Introduction

During the last couple of years, shopping lists have travelled from paper into our smartphones. This allows the shopping lists to have new features that contribute to a quicker and smoother shopping experience. Two such features that have been examined in this project are item sorting and automatic suggestions based on statistics and the user's shopping history.

1.1 Motivation

Technology often strives for solutions that make people's lives easier. Grocery shopping is a necessary evil that every household has to regularly do. A shopping list that is sorted could be a time-saving touch to peoples' everyday lives, either because it will not have to be done manually or because it will be harder to miss items as opposed to using an unsorted shopping list.

One example of a company that already use this kind of feature is ICA [9]. Their app can only sort items after their own stores. The application developed in this project use a more general approach which will be independent of what store it is used in. Also, it does not require any work from the stores themselves since it is the users that provide the data.

1.2 Aim

As a part of the course TDDE19 *Advanced Project Course - AI and Machine Learning* a smart shopping list was requested. The aimed result is a shopping list that sorts added items into their order of appearance in a specific store, and suggest new items to be added, based on the user's normal shopping behaviour as well as on the items already added to the list. The outcome of this project is supposed to be used in a mobile application.

1.3 Delimitations

This project will mainly focus on algorithms for sorting items and give item suggestions. Thus, implementation of a mobile application and use of different models for different stores will be left as future work. Since limited amount of data for the sorting is available, categorization will also be investigated.

1.4 Limitations

One natural path to take is to create personal models for each user of the application. These models would have a greater chance of finding relevant patterns since they only care about one person. This will however not be possible since the data needed to create these models do not exist.

1.5 Possible Solutions

Solutions to the two major subproblems of the project, sorting and suggestions, are discussed in this section.

1.5.1 Sorting

The goal is a shopping list that sorts added items into their order of appearance in a specific store. To accomplish this, knowledge of the layout of a store, or timestamps of when items are picked up in a store, is necessary. Since this data is limited, different approaches will be investigated. Usually, items that are similar to each other, i.e. are in the same category, will be located in the same section in a store, e.g. dairy products and fruits. Thus, some type of sorting might be accomplished by categorizing the items in the list. Furthermore, since more data with categorized items is available, the focus will

mainly be on categorization. Sorting might also be easier if the items are categorized first.

1.5.2 Suggestions

Many online shopping sites have a section where they suggest items that they think suit the user, so naturally, a smart shopping list should have a suggestion function where it suggests relevant items that it thinks the user should buy.

1.5.2.1 Neural network

One of the first ideas was to use an Artificial Neural Network for the predictions. Such networks have previously been used for different types of prediction tasks to great effect.

The hypothesis was that the accuracy will be rather low, since the dataset used does not seem to contain any clear trend or pattern for the network to learn.

1.5.2.2 Association Rule Learning

It was first decided to implement some association rule algorithm in order to compare it to the neural network. Association rule learning is used to find patterns in large datasets [2], often in the form of strong rules [1] on the form $X_1, X_2, ..., X_n \rightarrow Y_1, Y_2, ..., Y_m$. When the first association rule algorithm was proposed by Agrawal et al. [1], it was used for for finding patterns among customer transactions in stores and has since continued to be used for that purpose. Therefore, it is interesting to see how well association rule learning performs when used for predictions.

The hypothesis is that association rule learning will produce very meaningful predictions, but not many enough to generate the best result.

2 Methods

The different methods that were used in the projects are presented in this section.

2.1 The datasets

During the development of the project, a number of different datasets have been used to train machine learning models and to evaluate performance. All data that was used was in English.

2.1.1 Groceries

In 2006, Hahsler et al. [5] created a dataset gathered from a real-world grocery store during one month. In total, the dataset has 9835 transactions where each item is categorized into one of 169 categories. Furthermore, the items are also divided into more general categories, with 55 and 10 different category labels respectively. Additionally, the dataset is included in the CRAN arules package [4].

2.1.2 Timestamp data

In order to simulate shopping behavior, Cyrille Berger provided a script that generates lists of items along with timestamps stating when the items were added and removed from the list. The order in which the items were removed was based on the order they could appear in a real grocery store, but with some added noise to better simulate the real-world problem.

2.1.3 Web-Scraping

Categories and items were also retrieved by web-scraping the web page of the grocery store Lidl [11]. Around 3700 items categorized into 21 categories were retrieved.

2.2 Sorting and Categorization

One of the main functionalities with the application is that it is supposed to sort the added items in order of appearance in a certain store. This can also be achieved by categorizing the items, since items in the same category usually are located near each other in a grocery store.

2.2.1 Sorting

The sorting was found to be an easily solved problem with the right amount of good data. No machine learning was used to solve the problem, but instead a simple sorting algorithm.

2.2.1.1 Simple sorting algorithm

The sorting problem was solved by using the timestamp data to check at what time during a grocery store visit a specific item was removed from the shopping list. Items with early timestamps was then assumed to appear at the beginning of a store and vice versa. The Timestamp data was used to calculate a mean value for when each individual item was removed from the shopping list.

2.2.2 Categorization

To categorize the shopping list two different approaches were investigated: a neural network solution and a clustering solution.

2.2.2.1 Pre-trained word embeddings

The first step for the categorization was to investigate different pre-trained word embeddings. Different dimensions of the pre-trained word embeddings from GloVe, trained on Wikipedia 2014 and Gigaword 5, were tested. They provided vectors with 50 dimensions, 100 dimensions, 200 dimensions, and 300 dimensions and all of them were tested. The categorized items from the grocery dataset [4], described in 2.1.1 was used to compare the different dimensions with each other. By calculating the cosine distance of an item in a category with the other items in the same category and taking the mean, a similarity measure could be obtained. This was done for one item in every category. The embedding with 200 dimensions gave the best similarity measure and was used in later steps.

2.2.2.2 Neural Network

The Lidl-data was used for the neural network approach. The items in the Lidl-data consisted of several words, so a bag-of-words representation was used. The data was split into 20% test-data and 80% training-data. A vocabulary of over 2000 words was then obtained from the training data. An embedding matrix was created by getting the vectors of every word in the vocabulary, the vectors were obtained from the pre-trained word embedding. The first layer of the model was the embedding layer. The fixed input size was 10 words and the embedding matrix was set as the weights. After the embedding layer, there was a dropout layer with rate 0.5 and a Long Short Term Memory (LSTM) layer with 128 units. After that, there was one more dropout layer with rate 0.5 and a dense layer with the softmax activation function.

The model was compiled with the categorical cross-entropy as loss function and the Adam optimizer. Early stopping with patience 4 and a ratio of 80% training-data and 20% validation-data was used.

2.2.2.3 Clustering

The word embeddings were used by K-Means clustering in order to cluster closely related items together. The implementation is described in Algorithm 1. The algorithm was run with different K:s to find the highest possible similarity. In addition to the word embeddings, the mean timestamp values found with the simple sorting algorithm described in section 2.2.1.1 were tested as features for the clustering.

Algorithm 1: K-Means algorithm

Input: Items, Number of clusters K.
Output: Clusters with Items
1 Initialize K random centroids
2 Repeat step 3, 4 and 5 until convergence
3 Assign each item to its closest centroid
4 Calculate sum of similarity between centroids and their assigned items
5 Update centroids to mean of all items assigned to each cluster

The performance of the clustering was evaluated by the total cosine similarity according to Equation 1, where C_i is a cluster, c_i is the cluster centroid, v is an item belonging to the cluster and $\cdot$ is the dot product.

$$\text{Total similarity} = \sum_{i=1}^{K} \sum_{v \in C_i} \frac{c_i \cdot v}{||c_i||_2 ||v||_2} \tag{1}$$

2.3 Suggestions

The other major functionality of the application is suggesting new items to the user based on the previously selected items.

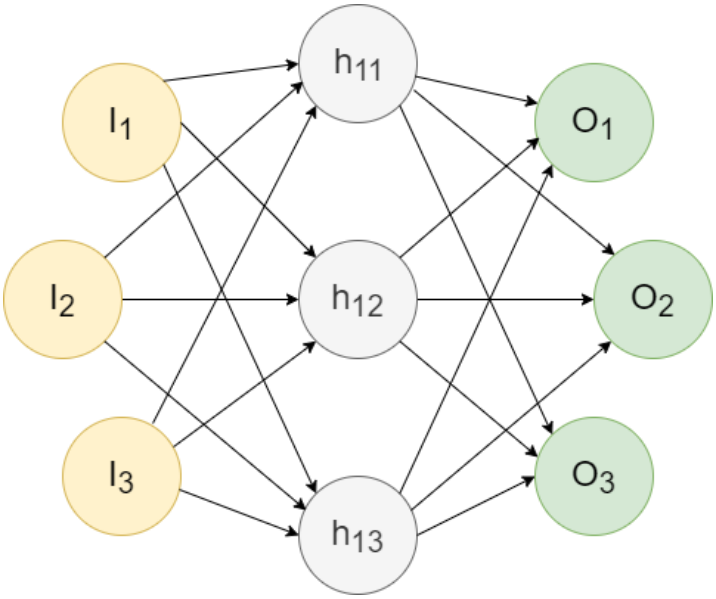

Figure 1: Example of simple DenseNet network with one hidden layer.

2.3.1 Neural Networks

One group decided to implement a neural network to offer suggestions. In the following subsection, their methods will be shown.

2.3.1.1 DenseNet

The network is of DenseNet architecture, which means that all layers are fully connected. These networks require fixed sized input. Initially, a single fully connected layer was used as a baseline. In order for a growing shopping cart to become fixed size, the input was converted to a bag of words representation. The resulting bag of words vector was 169 binary variables long, each corresponding to one of the 169 different categories in the Hahsler shopping list dataset used for this part of the project. Data was generated from the shopping lists in the dataset by splitting the lists into input and correct output. If the network was able to guess one or more of the correct outputs for a given input, it was considered a successful prediction. To make the network generalize to lists of different lengths, data was generated using Algorithm 2.

Algorithm 2: Data generation for Neural Network

Input: List of items, D.
Output: Train and test data
Initialize a data store, S
for *each item* $I \in D$ **do**
    Create copy of D, Dc.
    Split Dc at index of I.
    Insert the pair created by the split into S
end

Some improvements were then made to increase the accuracy of the model. This was done in several steps. Two fully connected layers were added to the model, along with a dropout layer between the first and second fully connected layers. The number of epochs used during training was also increased. To mitigate the overfitting effects from this, early stopping was also implemented. These improvements gave significant increases in accuracy. An example of a DenseNet network can be seen in Figure 1 above.

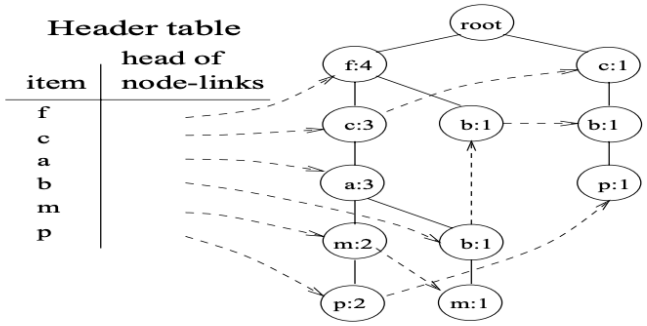

Figure 2: Example of header table [7]. Dotted lines represent a header link.

2.3.2 Association Rules

There are multiple association rule algorithms and one had to be chosen because of time constraints. Two popular algorithms are Apriori and FP-growth. In this project, FP-growth was implemented because it is more efficient than Apriori [7, 8]. A method based on the algorithm Breadth First was also implemented, which would theoretically give more general suggestions than FP-growth.

The FP-growth algorithm was proposed in 2000 by Han et al. [7]. FP-growth has the advantage of not having to generate so called candidate sets, in contrast to the previously mentioned Apriori algorithm. The candidate sets are possible combinations of items in the database which increase exponentially in size with the database size. The algorithm comes in two parts: Tree construction and mining of the tree for frequent patterns.

2.3.2.1 Tree Construction

The tree is used in order to get a compressed representation of the transactions that facilitates later frequent pattern mining. Each node in the tree has an item name and the number of times it occurs in the path leading to that node. This path can be seen as all the transactions containing the node and all the nodes before it in the path. Furthermore, each node has a header link that points to the next node in the tree that also has the same item name. These connections between nodes with the same item names create the header table. The header table defines the mining order of items with the same name. An example of a header table can be found in Figure 2. The algorithm for constructing the tree can be found in Algorithm 3 and Algorithm 4. The algorithms use the measure support. Support is simply the amount of times an item appears in the transactions [1].

Algorithm 3: FP tree construction [7, 10]

Input: Transaction database D, minimum support *minsup*.
Output: The FP tree for D and *minsup*

```
1 Count the support for each item in D
2 Remove the infrequent items from the transactions in D
3 Sort the items in each transaction in D in support descending order
4 Create an FP tree with a single node T with T.name = NULL
5 for each transaction I ∈ D do
6     insert-tree(I, T)
7 end
```

Algorithm 4: insert-tree($I_1, ..., I_m, T$) [7, 10]

Input: Itemset $I_1, ..., I_m, T$ and a node T in the FP tree.
Output: Modified FP tree.

```
if T has child N such that N.name = I_1.name then
    N.count++
else
    Create a new child N of T with N.name = I_1.name and N.count = 1
    Add N to the end of the header table
end
if m > 1 then
    insert-tree(I_2, ..., I_m)
end
```

2.3.2.2 Mining Frequent Patterns

In order to find items that are often bought together, i.e. frequent patterns [6], the tree has to be mined. By making use of the header table, each node serves as a starting point, or suffix, in each iteration of the tree mining. Next, a conditional pattern base is created for that node, meaning all paths that end with the aforementioned suffix. The conditional pattern base is then used to construct a conditional sub-tree with Algorithm 3. This sub-tree is then recursively used as input to the mining algorithm. Suffixes larger than one item is achieved by concatenating suffixes generated from the conditional sub-tree. As a result, all frequent patterns of varying length will be generated [7]. The process is described in Algorithm 5.

Algorithm 5: FP-growth($Tree, \alpha$) [7]

Input: Tree constructed as described in Algorithm 3 and minimum support ξ.
Output: The complete set of frequent patterns.

```
if Tree contains a single path P then
    for each combination (denoted as β) of the nodes in the path P do
        generate pattern β ∪ α with support = minimum support of nodes in β
    end
else
    for each a_i in the header table of Tree do
        generate pattern β = a_i ∪ α with support = a_i.support construct β's
          conditional pattern base and then β's conditional FP tree Tree_β
        if Tree_β ≠ ∅ then
            FP-growth(Tree_β, β)
        end
    end
end
```

2.3.2.3 Generating Strong Rules

Lastly, strong rules are needed to make the predictions. These rules are generated from the frequent patterns and exist on the form
$X_1, X_2, ..., X_m \rightarrow Y_1, ..., Y_n$. A rule expresses that if a transaction contains the items to the left of the arrow, the antecedent, then we also expect to see the items to the right of the arrow, the consequent [3]. How certain a rule is can be expressed in different measures. One popular measure is confidence and is defined as [1]:

$$Conf(X \rightarrow Y) = Supp(X \cup Y)/Supp(X)$$

The rules are generated by taking all combinations of the antecedent together with the original consequent and removing the items that appear in the antecedent. This

is done for each frequent pattern and after this, the confidence is calculated for each rule. The rules that do not reach the minimum confidence specified by the programmer are discarded [6]. Thus, all possible strong rules that meet the minimum confidence requirement are generated.

2.3.2.4 Breadth First

The Breadth First implementation used the generated Tree, but in the opposite way of FP-growth. Instead of searching patterns beginning from the leaves, probable item suggestions was based on patterns beginning in the root of the tree. See the following pseudo code.

Algorithm 6: Breadth-First($Tree$, $Cart$)

Input: $Tree$ constructed as described in Algorithm 3 and current shopping list $Cart$
Output: List of items sorted on probability in ascending order.

```
1  Create an empty map pointing from items to probabilities probabilities
2  Store each item and it's probability in res
3  Initialize the frontier with only the Tree root contained
4  while frontier is not empty do
5      Pop frontier and store it in node
6      for child in node.children do
7          Calculate probability by dividing child.count with node.count
8          if name in Cart then
9              frontier.append(child)
10         end
11         else
12             if name not in probabilities then
13                 res.append(child)
14                 probabilities[child] = probability
15             end
16             else if probability > probabilities[child] then
17                 Update the item with name of child in res with the new
                    probability probability
18                 probabilities[child] = probability
19             end
20         end
21     end
22 end
23 Sort res on probability in ascending order
24 returnres
```

In short;

- Search from the root node
- Add all items not already in the shopping list query, and calculate it's probability: *child.count / parent.count*
- Add all items already in the shopping list query to the frontier
- Continue with the same process but start searching from the elements added to the frontier

2.3.2.5 Evaluation

Accuracy: To test the accuracy of the rules, the dataset was first divided into 75% training data and 25% test data. Because the result of the FP-growth algorithm depends on the user defined parameters *minimum support* and *minimum confidence*, the parameters that yields the highest accuracy had to be found.

For minimum support between 2 and 20 transactions and minimum confidence between 20% to 100%, the FP tree was constructed and mined for frequent patterns. The patterns were used to generate the strong rules.

The training data was firstly used to create an FP tree, secondly the tree was mined for frequent patterns and lastly strong rules were generated according to previous subsections. Then, all combinations of items of size 1 to 4 for each transaction in the test data was used for predictions. The combination of items was used as input to check if it matches the antecedent of a rule. If it does, the rule consequent would be the prediction. Next, if the prediction contains any of the items left in the transaction that was not a part of the antecedent, then it is a correct prediction. If there is no matching rule to the item combination or no item in the consequent matches the rest of the items in the transactions it is then an incorrect transaction.

A maximum combination size of 4 was chosen because larger values took exponentially longer time to generate with seemingly not better results.

Precision: Precision describes how many items were correctly predicted when the algorithm provides a prediction. More formally, precision is described as:

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

In this case, the true positives are correctly predicted items and false positives are wrongly predicted items.

Recall: Similarly to accuracy, recall describes how many of the items are labeled correctly. Recall is formally defined as:

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

In this case, the true positives are correctly predicted items and false negatives are when the algorithm gives no prediction.

f1 score: Precision favors very accurate predictions, even if the algorithm seldom makes a prediction. In contrast, the recall measure will favor algorithms that often make predictions. In order to get a balance between the two measures, the f1 score was constructed. It is defined as:

$$f1\ score = 2 * \frac{Precision\ *\ Recall}{Precision\ +\ Recall}$$

Comparison Between FP-Growth and Breadth First: Some sort of comparison was needed for these two methods, and it seemed logical to test them based on two critical factors: number of known items in shopping list, and number of unknown items. The number of known items controls the possibility of narrower and more specific suggestions, while the number of unknown items broadens the number of possible correct answers. The first measurement would hypothetically be more beneficial to a specific-giving suggestion method, and the second by a model giving general suggestions. The training and test data was created by a split of 80- and 20% of the original dataset. All samples was then modified to only contain 1 to (N-1) items of the original sample, where N is the length of that sample. The rest of the items that had been cut out was used as possible correct answers to each suggestion query. A correct suggestion was defined as one of the top three suggestions being in the cut-off items of the sample shopping list.

3 Results

In this section the results from the different experiments are presented.

3.1 Sorting and Categorization

Below are all results gathered from the sorting and categorization methods.

3.1.1 Simple sorting algorithm

The simple sorting algorithm performed well to the naked eye. The method is very straight-forward and should give reasonable results as long as the data follows a certain pattern without too many outliers. This was the case with the dataset that was used in this project. The sorting result is presented in Appendix A.

3.1.2 Categorization

With the neural network solution a test accuracy of 0.80 was obtained. In Figure 3 the results for each class in the test data can be seen. In Figure 4 the loss curve and the accuracy curve can be seen.

	precision	recall	f1-score	support
baby	0.65	0.58	0.61	19
bakery	0.70	0.67	0.68	21
baking & cooking ingredients	0.62	0.52	0.57	25
beverages	0.79	0.92	0.85	48
candy	0.89	0.76	0.82	33
dairy	0.89	0.96	0.92	70
desserts & cookies	0.74	0.70	0.72	20
fresh fruits	0.89	0.70	0.78	23
fresh vegetables	0.85	0.79	0.81	28
frozen foods	0.94	0.97	0.96	68
health & beauty	0.78	0.89	0.84	57
herbs & spices	0.92	0.92	0.92	13
household essentials	0.96	0.81	0.88	32
meat & poultry	0.77	0.93	0.85	44
organic selection	0.43	0.60	0.50	25
pantry	0.78	0.72	0.75	101
pasta, rice & grains	0.59	0.83	0.69	12
pet shop	0.94	0.94	0.94	17
refrigerated	0.77	0.57	0.65	30
sauces & marinades	0.91	0.71	0.80	14
snacks	0.83	0.78	0.80	49
accuracy			0.80	749
macro avg	0.79	0.77	0.78	749
weighted avg	0.81	0.80	0.80	749

Figure 3: Precision, recall, f1-score and support for each class in the test-data

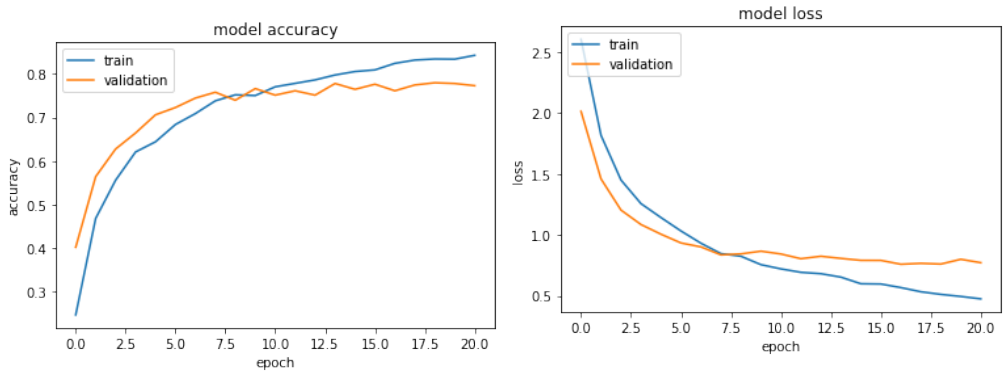

Figure 4: Accuracy and loss curve during training

3.1.3 Clustering

The K-Means clustering was run with one through ten clusters as seen in Figure 5. The additional time feature did not give a noticeable difference in cluster similarity. According to the Elbow method that is used to find an appropriate K, two or three clusters are enough for the dataset, although there are more categories in the data. An example clustering with K=7 and word embeddings as features can be found in Appendix B.

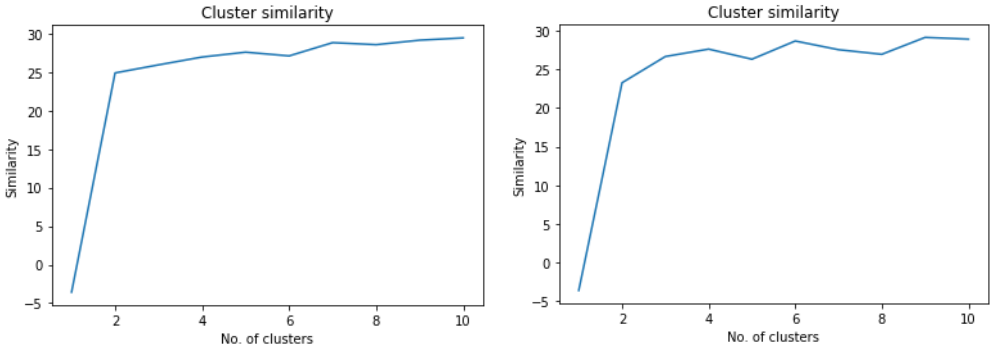

Figure 5: Total cluster similarity for different K:s with word embeddings (left) and with both word embeddings and time (right) as features.

3.2 Suggestions

Below are all results gathered from the different suggestion methods.

3.2.1 Neural networks

The initial neural network structure using only one hidden layer achieved an accuracy score of 51%, while the final neural network using 3 hidden layers and a dropout layer between the first and second hidden layers in addition to using model selection to find the best model achieved an accuracy score of 56%.

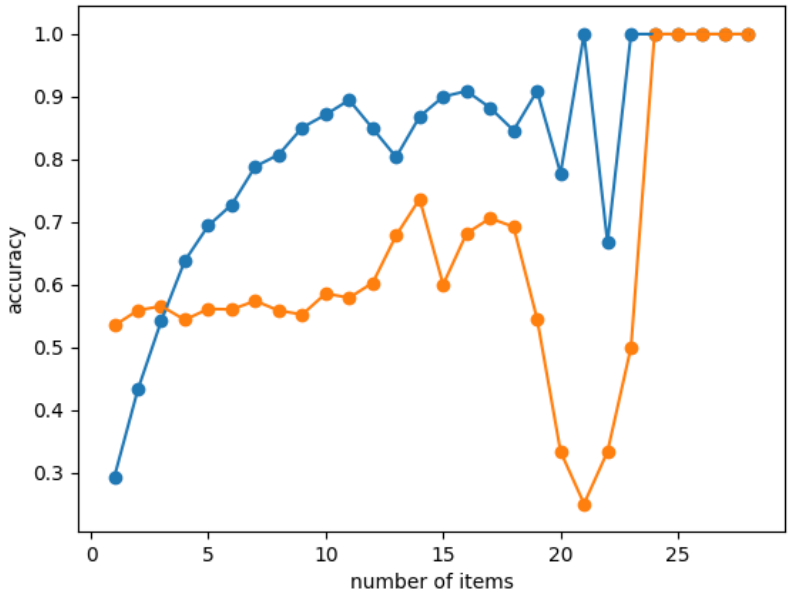

Figure 6: Neural network accuracy based on number of known items and number of unknown items.

The blue line represents the accuracy of the model in correlation to how many items that gives a correct prediction. The orange line represents the accuracy of the model in correlation to how many items the model got as input.

The model is more accurate when having many items as input, and when there is a lot of items that gives a correct answer.

3.3 Association Rule Learning

Below are all results gathered from the Association Rule based methods.

3.3.1 FP-growth

The plot showing the accuracy, precision, recall and f1 score can be found in Figure 7, 8, 9 and 10 respectively. The best results and the parameters that yielded them can be found in Table 1. Note that the precision result is excluding the 100% precision outlier, since it only provided 4 predictions.

	Best Result	Min Support	Min Confidence
Accuracy	34%	16	0.2
Precision	63%	19	0.8
Recall	61%	20	0.2
f1 score	51%	16	0.2

Table 1: Best results of different measures and their parameters.

The elapsed time of the FP-growth algorithm is presented in Figure 11. By looking at the graph, it becomes evident that the FP-growth runtime increases exponentially as the minimum support is decreased.

Using the highest score parameters, the accuracy based on number of known and number of unknown items is shown in Figure 12. The average accuracy is 20%.

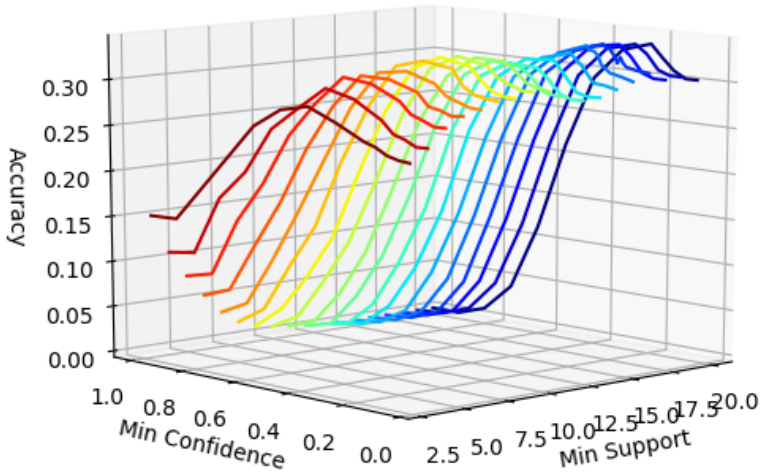

Figure 7: FP-growth accuracy plotted against minimum support and minimum confidence.

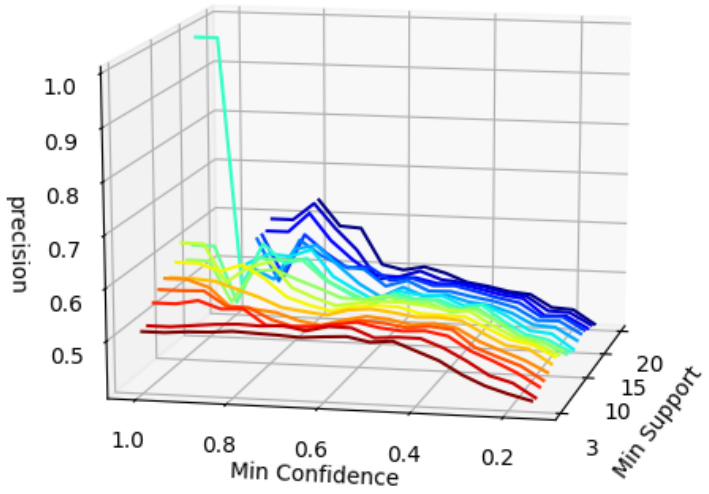

Figure 8: FP-growth precision plotted against minimum support and minimum confidence.

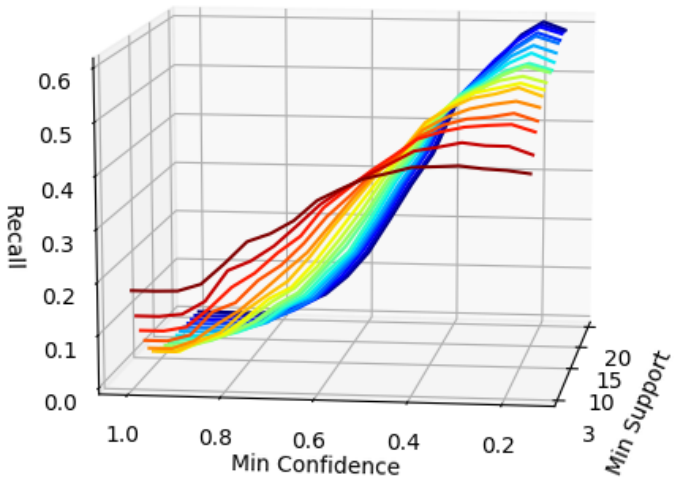

Figure 9: FP-growth recall plotted against minimum support and minimum confidence.

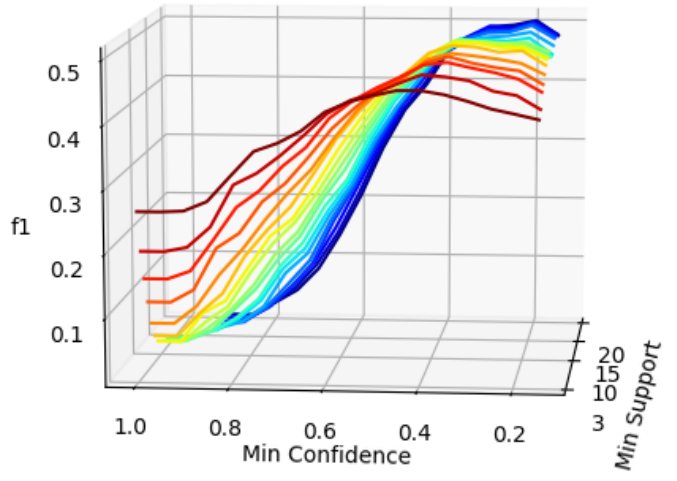

Figure 10: FP-growth f1 score plotted against minimum support and minimum confidence.

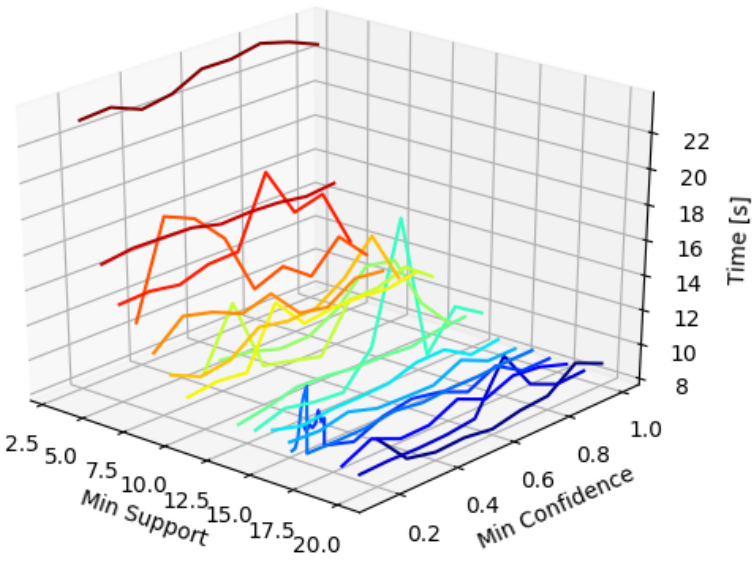

Figure 11: FP-growth execution time plotted against minimum support and minimum confidence.

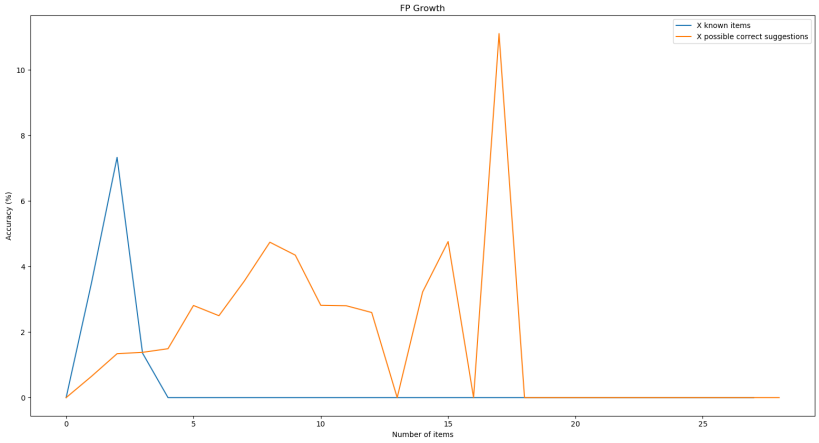

Figure 12: FP-growth accuracy based on number of known items and number of unknown items.

3.3.2 Breadth First

Using the Breadth First algorithm, the accuracy based on number of known and number of unknown items is shown in Figure 13. The average accuracy is 38%.

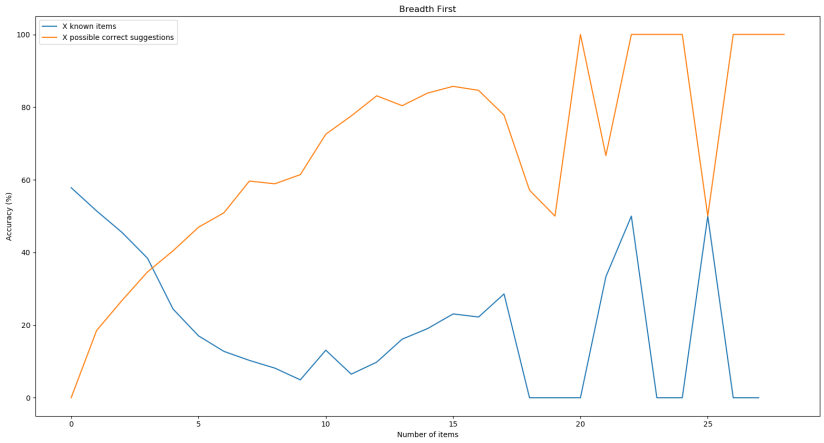

Figure 13: Breadth First accuracy based on number of known items and number of unknown items.

4 Discussion

In this section the results for the different solutions are discussed.

4.1 Sorting and Categorization

Since timestamp data is used, the solution to the sorting problem is heavily dependent on that the user removes each item from the list once it is collected in the store, instead of doing it in bunches or when at home. An alternative method that is already used by e.g. ICA [9] today, is to have a sorted list of the items already in the app. This will always give perfectly sorted shopping lists for the supported stores, but cannot be generalized to work with any store.

The main advantage of using categorization instead of sorting is the data. There are more datasets with categorized items than with items and timestamps. It is also easier to collect categorized items, e.g. with web scraping.

The use of pre-trained word embeddings worked pretty well for the categorization even though they are not trained for this particular case. The optimal solution would probably be to train word embeddings in a way so that for example all vegetables have a high similarity while being different to dairy products.

Even though a simple model architecture was used in the neural network solution, the model had quite high accuracy which is very promising and shows that there is potential for this type of solution. However, if this solution is consistent with reality, the layout of an actual store, is not clear. A study should be conducted to investigate which categories that are representative of the sections of grocery stores, and if it is possible to generalize the categories so that they fit several different grocery stores. An advantage of the neural network solution compared to the clustering solution is that it can handle an item that is described with several words. Support for this should be added to the clustering solution as well.

Future work in this task would be to combine the sorting and the categorization. It would be desirable to sort the categories and if GPS coordinates where added to the items, this would allow different sorting for different stores.

4.2 Suggestions

In this section the results for the different suggestion methods are discussed.

4.2.1 Neural nets

The results from the neural network were better than expected. Since the dataset used contained receipts from many different customers, the prospects of discovering meaningful patterns were considered bleak. However, the resulting 56% final accuracy score is significantly higher than the 45% score achieved when suggesting only the two most frequent items in the dataset. The method of calculating accuracy is also somewhat arbitrary, since suggestions is just that, suggestions. When using the application in real life, even though predicting only eggs, milk and bread might have given a high accuracy, the application would be completely useless. Therefore, there might be other evaluation methods that are more relevant than accuracy for pre-made receipts. This issue however, was beyond the scope of this project, and is something that should be considered if this should turn to a real product.

4.2.2 Association Rule Learning

The results from making predictions using association rule learning do not look promising. In fact, just suggesting the two most frequent items, 'whole milk' and 'other vegetables', will perform better than predictions made by the FP-growth algorithm with 45% accuracy to the 34% of FP-growth. While suggesting the same two items every time has a higher accuracy, is it fair to say it is *better* than FP-growth? Probably not, since users would find the same suggestions rather pointless. In contrast, users may accept that they do not always get item suggestions on every input, but sometimes very useful

suggestions. Therefore, accuracy may not be the best measure to define predictions performance in the case predicting items in a shopping list. However, the f1 score that is a balance between precision and recall, reported the same parameters as accuracy did. Perhaps this is a coincidence, or accuracy may be a good indicator of how good an algorithm is after all.

The Breadth First algorithm seemed to perform better than FP-growth, but, as previously noted, accuracy does not give a fair comparison of these methods. When testing these manually, and generating shopping lists from scratch, FP-growth rarely gave any suggestions at all, but when it did, it gave some specific and useful suggestions. The Breadth First algorithm gave more general results, and a lot of them. The two algorithms functionality can be described with: If we have a shopping list with bread and butter, FP-growth would say something like "ham", or "sliced cheese", while Breadth First would give something related to the breakfast genre, e.g. "yoghurt".

We concluded that the best use of association rules is not to use one of these methods solely, but complement it with a method giving more general/specific suggestions as needed.

References

[1] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining Association Rules Between Sets of Items in Large Databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, SIGMOD '93, pages 207–216, New York, NY, USA, 1993. ACM. event-place: Washington, D.C., USA.

[2] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. The KDD process for extracting useful knowledge from volumes of data. *Communications of the ACM*, 39(11):27–34, November 1996.

[3] William J. Frawley, Gregory Piatetsky-Shapiro, and Christopher J. Matheus. Knowledge Discovery in Databases: An Overview. *AI Magazine*, 13(3):57–57, September 1992.

[4] Michael Hahsler, Christian Buchta, Bettina Gruen, Kurt Hornik, Ian Johnson, and Christian Borgelt. arules: Mining Association Rules and Frequent Itemsets, August 2019.

[5] Michael Hahsler, Kurt Hornik, and Thomas Reutterer. Implications of Probabilistic Data Modeling for Mining Association Rules. In Myra Spiliopoulou, Rudolf Kruse, Christian Borgelt, Andreas Nürnberger, and Wolfgang Gaul, editors, *From Data and Information Analysis to Knowledge Engineering*, pages 598–605. Springer-Verlag, Berlin/Heidelberg, 2006.

[6] Jiawei Han, Micheline Kamber, and Jian Pei. *Data mining concepts and techniques, third edition*. Morgan Kaufmann Publishers, Waltham, Mass., 2012.

[7] Jiawei Han, Jian Pei, and Yiwen Yin. Mining Frequent Patterns Without Candidate Generation. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, pages 1–12, New York, NY, USA, 2000. ACM. event-place: Dallas, Texas, USA.

[8] Daniel Hunyadi. Performance comparison of apriori and FP-growth algorithms in generating association rules. 2011.

[9] ICA. Appen ica. `https://www.ica.se/appar-och-tjanster/appen-ica/`. Last visited: 2019-11-16.

[10] Jose M Peña Ida and Linköping University. 732a61/TDDD41 Data Mining - Clustering and Association Analysis - Lecture 7: FP Grow Algorithm. page 17.

[11] Lidl. Lidl. `https://www.lidl.com/`. Last visited: 2020-01-04.

Appendices

A Simple sorting algorithm result

Item
strawberries
pear
rhubarb
oranges
grapes
mango
banana
nectarines
apricots
apple
peach
pineapple
grapefruit
avocado
salad
lemons
carrots
potatoes
garlic
onion
cucumber
tomatoes
mushroom
broccoli
chicken
sausage
entrecote
cream
eggs
yoghurt
milk
yeast
sugar
ice cream
flour
cleaning product
frozen vegetables
frozen diner
detergent
toilet paper
mop
toothpaste
shampoo
soap

B Clustering result

Cluster	Item
1	strawberries
	oranges
	grapes
	nectarines
	apricots
	grapefruit
	lemons
2	pear
	mango
	banana
	apple
	pineapple
	avocado
	carrots
	potatoes
	garlic
	onion
	tomatoes
	mushroom
	chicken
	eggs
3	rhubarb
	peach
	broccoli
4	salad
	cucumber
	sausage
5	cream
	milk
	yeast
	sugar
	flour
	mop
6	yoghurt
	detergent
	toothpaste
	shampoo
	soap
7	ice cream