UNIVERSITY OF CAPE TOWN

MASTERS THESIS

Exploring the Application of Word2Vec to Basket Transaction Data in the Grocery Retail Industry

Author:
Gideon Jacobus DE SWARDT

Supervisor:
Dr. Juwa NYIRENDA

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Department of Statistical Sciences

October 19, 2021

Declaration of Authorship

I, Gideon Jacobus DE SWARDT, declare that this thesis titled, “Exploring the Application of Word2Vec to Basket Transaction Data in the Grocery Retail Industry” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Signed by candidate

Date:

"You're never too old to dream a new dream or to set another goal"

CS Lewis

UNIVERSITY OF CAPE TOWN

Abstract

Faculty of Science

Department of Statistical Sciences

Master of Science

Exploring the Application of Word2Vec to Basket Transaction Data in the Grocery Retail Industry

by Gideon Jacobus DE SWARDT

In this thesis, we explore the application of Word2vec to basket transaction data provided by a large grocery retailer in South Africa. Word2vec is an algorithm based on representation learning. The objective of the exploration is to establish whether the application of Word2vec to basket transaction data would generate product embeddings that represent a useful relationship between products. Furthermore, we compare Word2vec's outputs and performance to traditional methods for studying product relationships which include Association Rules Mining (ARM) and Recommendation Systems. The results from the experiments showed that indeed product embeddings created by Word2vec on transaction data are meaningful and useful. It was clear that the idea of using transactions in the place of sentences to the neural network, provides analogous results to that of a natural language task. Word2vec clearly demonstrated its ability to cluster products that are homogeneous or fulfill similar needs. Furthermore this sort of product relationship was not provided by any other traditional methods, which was clear when comparing the outputs to that of ARM and Recommendation Systems. We also show that using Word2vec could potentially provide insight on truly complementary products that ARM perhaps fails to do. Word2vec also proved to be incredibly scalable, taking input data of 20 times the size of what traditional methods could handle on a local computer. We end with a description of a potential application of the ideas learnt during the course of this study, with a real business problem, that we believe could lead to an enhanced customer shopping experience and in turn increase revenue and profits for the retailer.

Acknowledgements

I would like to thank my supervisor Dr. Juwa Nyirenda from the University of Cape Town, for his continuous support throughout the years, as well as all other staff that made it possible for me to study this degree. Especially, I would like to thank Dr. Francesca Little, for giving me an opportunity to enrol into this masters, and for not giving up on me when I struggled to manage the study/work balance. I would also like to thank Dr. Ian Durbach, Dr. Sebnem Er and Dr. Etienne Pienaar for inspiring me. Thank you to my career/study friends Stiaan Maree and Zander Wessels, for their unwavering belief in me, even when I didn't believe in myself. Thank you to Dr. Morne Lamont from Stellenbosch University, for the massive role he played in my academic pursuits. Thank you to my parents for always supporting me in everything I do and pushing me to reach greater heights in my life and for making it financially possible for me to complete my studies. Finally, thank you to my fiancee Karmen de Nobrega for her love and putting up with my work, over the past few years. . . .

Contents

List of Figures

List of Tables

Dedicated to my parents Jaco and Antionette de Swardt…

Chapter 1

Introduction

1.1 Background

This study explores the opportunities that technology and big data present to the retail industry with a particular focus on the grocery retail business. It is an undeniable fact that over the past few decades, most grocery retailers have routinely collected and stored massive amounts of transaction data (Micro 2018). This practice is ongoing and increasing by the day. The vast amount of data is collected through what is known as Electronic Point Of Sale (EPOS) systems (Raza 2017) which we will refer to as transaction data in this study. The information collected include among others: product name, store where product was sold, date on which product was sold and information on the customer who bought the product.

The data, if used properly, as demonstrated in other industries (Akter 2016), has potential to provide retailers with the ability to gain a competitive advantage in the share of the market and improve their bottom-line considerably. However, although the grocery retail industry is known to be highly competitive, very little is due to the exploitation of big data and data science. The majority of the businesses continue to employ traditional approaches to make critical decisions and drive operations. Data is used mainly for descriptive analysis, with focus on reporting historical events that have less to do with future decisions. Data is hardly used for predictive or prescriptive analysis from which the company could typically derive value and competitiveness.

The inertia exhibited by major grocery retailers to adopt data science could be due to not understanding data management for analysis. But perhaps the main reason is the fear of upsetting their traditional method of work which has since served them well (Duke 2019). This has instead led to the emergence of consultancy companies such as dunnhumby, IRI, Oracle retail, 84.51 degrees and many other smaller start-up companies specializing in the application of data science to the grocery retail industry. These companies have shown how data science methods can be applied to transaction data and deliver massive value for retailers. Using data driven approaches these companies have the ability to automate and provide insight to retailers on their customers needs and thought processes (Nikolova et al. 2014).

An area involving transaction data that can potentially benefit from using technology and in turn help retailers improve their businesses is concerned with understanding the different types of relationships that exist between products such as item-compliments and item-similarities. These relationships between products are so important as they provide invaluable information used in various marketing strategies aimed at increasing sales and market share such as: advertising campaigns, up-selling, cross-selling, sales promotions, dual marketing and recommendation system design (McAlister 1983).

Item-similarity is defined in terms of user characteristics, where two products are regarded as similar when both have been purchased by customers with similar purchasing history or characteristics. A common concept used to measure similarity between two products is the euclidean distance where dimensions represent the features of products. If such distance is small the similarity between both products is high, while a large distance would indicate a low degree of similarity. The concept of item-similarity forms the basis for Recommendation Systems, typically used in e-commerce, but the idea holds true for physical retail stores. These systems filter content using various similarity measures, to create personalized recommendations or offerings to users. Takealot is an example of an e-commerce platform that makes use of these kinds of systems under sections titled, "You might also like" or at checkout you may see the following: "Have you not forgotten this?". These sort of recommendations play a large role in e-commerce, in order to create a delightful user experience (Koren et al. 2009).

The relationship between products defined by item-complements, on the other hand, measures how often two or more products are associated with each other. In other words, products X and Y are said to be complementary if using more of product X requires the use of more of product Y. For example, printers and cartridges are complementary products in the sense that the demand for printers generates demand for ink cartridges. This sort of concept can be found in literature relating to Association Rules Mining (ARM). ARM gives insight into products that are bought together frequently, referred to as frequent item-sets. The item-complement relationship between products has various applications in the retail business most important of which are: store layout, cross/up selling, catalogue design and target marketing (Shaukat et al. 2015).

These methods predate deep learning. While these traditional methods could work satisfactorily with small data sets, the advent of "Big Data" and its accompanying sparsity representation, has presented a number of challenges. The traditional methods require excessive computational memory and time complexity both of which are expensive (Madadipouya et al. 2017, Sano et al. 2015). In the case of ARM, in addition to not scaling well, the methods generates too many rules most of which do not provide any practical insight (Kotsiantis et al. 2006).

During the last few decades a large amount of interest and effort has been given to another data science field known as Natural Language Processing (NLP) (Manning 1999). Methods within this field attempt to employ machines to solve various linguistic tasks from large amounts of text or voice data. Of main interest is a method called Word2vec which has produced state-of-the-art results on various linguistic tasks. This method attempts to make sense of large amounts of text data by creating vector representations or embeddings of words, based on the context in which they appear in sentences (Ozsoy 2016). Word2vec is also able to capture different degrees of similarity between words. Semantic patterns can be reproduced using vector arithmetic. Thus, we could also use Word2vec to solve word analogy tests. For example, in their experiment, (Mikolov, Sutskever, et al. 2013) queried Word2vec for the analogy of "man is to king as woman is to ......" and Word2vec accurately predicted queen!

The method has further shown to be versatile even when the form of the data input to Word2vec is different from text, such as arbitrary product identifiers, which has led to a variant of Word2vec known as Item2vec (Barkan et al. 2016). Item2vec builds a representation of products by using click-through-rates on an e-commerce platform in order to recommend products to users based on their viewed history.

1.2 Objectives

In this study we investigate the application of Word2vec on actual basket transaction data for a physical grocery retailer. Here we assume that products in a transaction could be cast into the same light as click-through-rates on a website in e-commerce, or as words in sentences in an NLP task. Specifically, we would like to address the following questions:

1. Given the success of Word2vec in NLP tasks, can it be implemented in a grocery retail business using basket transaction data and create embeddings that represent a useful underlying relationship between products? By useful we mean, relationships that can be used to improve operations and profitability in the grocery retail business?

2. By using analogical reasoning tests, we have seen that Word2vec has the power to preserve semantic relationships between words and categorize characteristics of words that are not provided as input to the model. What similar hidden features can be inferred from analogical reasoning tests when performed on product embeddings created by Word2vec?

3. We have seen that traditional methods such as ARM and Recommendation Systems can create item-complements and item-similarities respectively using basket transaction data. How does the underlying relationship between product embeddings, created by Word2vec on transaction data, compare to the relationship represented by item-complements and item-similarities as created by traditional methods?

1.3 Motivation

The motivation to apply Word2vec to address our questions above stems from the fact that text data shares a lot of characteristics with transaction data. For example, in text data one is often interested in similar words or words that occur together in a corpus.

Similarly, in transaction data one is interested in similar products or products that are bought together frequently. Given the striking similarities between the two data types and the fact that both data types are high dimensional and sparse, it seems logical that the application of Word2vec to transaction data could likely result in data representations or embeddings from which item similarity or complements or any other interesting concepts applicable to the retail industry could be derived.

As discussed earlier, knowledge of item-complements or similarities, have the power to assist retailers in solving various retail marketing tasks such as product assortments, placement within shelves, general store flow, coupon development and other marketing functions. Kimberley Busdieker the CCO of Kroger, a large retailer in the US, had this to say: "For Kroger to deliver to its shareholders, it delivers to its customers first, by placing the customers needs as the main objective and doing so through the use of transaction data" (Cooper 2019).

The key take away point is that retailers who make use of data science and insight derived from transaction databases are in a better position to understand the needs of their customers quicker and more efficiently. This understanding could provide the retailer with a justifiable basis for initiating interventions or strategies that result in not only the enhancement of customers shopping experience but also likely to increase revenue.

Chapter 2

Theory & literature review

This chapter is split into two sections. The first section is a review of methods and models that may be considered as traditional for understanding various relationships and interactions that occur between products in transaction data. The second section is a review of Statistical NLP methods that led to the discovery of Word2vec, the model we will use to learn product relationships in a transaction database.

2.1 Traditional methods used to understand various relationships that exist between products

In this section we discuss two data science methods, that attempt to understand and quantify various relationships that exist between products using transaction data as input. We will discuss these methods against the backdrop of the grocery retail industry.

2.1.1 Association Rules Mining (ARM)

ARM widely referred to as Market Basket Analysis is a method for analysing large transaction databases for patterns or co-occurrence. The task of Market Basket Analysis is to discover actionable knowledge in transaction databases. It identifies frequent item sets - an item set is made up of at least one item or product - from which "frequent if-then associations", known as association rules are derived. Its application became popular in the last decade due to improvements in computing power, algorithms and storage technologies to the extent that retailers could collect and house data at an item-transaction level, where previously retailers could only have sight on aggregated daily or weekly sales.

Every rule is made up of two parts: an antecedent (if) and a consequent (then). An antecedent is an item set X found within the data. A consequent is an item set Y found in combination with the antecedent. A rule is defined as an implication of the form: $X \Rightarrow Y$ where X, Y are item sets.

Association rules are generated by searching for "frequent if-then patterns" in a transaction database through the application of two criteria: support and confidence to identify the most important relationships between item sets. Support is defined as the frequency with which the item set appears in the transaction database. Confidence on the other hand is a measure of the conditional probability that Y will be in the basket given that X, is in the basket or the number of times the "if-then" statements are found to be true. A third popular metric used with ARM is known as the lift. The lift is defined as the ratio of the observed support to that expected if X and Y were independent. A lift of 1, would imply that the probability of occurrence of X and Y are independent of each other. A value of lift greater than 1, is a measure of the degree to which the occurrence of X and Y are dependent on each other, and is potentially useful for predicting the consequent in future data sets.

In the early 1990's (Agrawal et al. 1993) proposed an efficient algorithm known as the Apriori algorithm for generating association rules between all items in a transaction database. Although many variations of this method now exists, in this study we will restrict ourselves to the original Apriori algorithm when conducting market basket related computations. The Apriori algorithm generates association rules in two steps. In the first step the algorithm generates all frequent item sets whose support is greater than the specified minimum support and in the second step it generates high confidence rules from each frequent item set, where each rule is a binary partitioning of a frequent item set. The algorithm employs the Apriori property to avoid complete enumeration of all item sets. The Apriori property states that: any subset of a frequent item set is also a frequent item set (Agrawal et al. 1993).

In the retail business the main application of ARM is in discovering products that are bought together often in single transactions. Such products are considered proxies for complementary products which are used to design a number of marketing strategies aimed at increasing revenue as discussed in Chapter 1, pp 2. For example, a retailer might want to know which products frequently appear in transactions that contain a fresh milk product. Intuitively we would expect the answer to be along the lines of bread or eggs. This intuition is not unfounded and according to (Agrawal et al. 1993) 90% of the transactions containing a milk product also contained bread and butter products. This is incredibly useful information and a retailer could use this type of insight and variations of it, to implement combo deals for products that are highly complementary and ultimately drive sales and volume (Shaukat et al. 2015).

A major disadvantage of mining association rules with the Apriori algorithm is that even with a relatively small sized database of transactions and despite the algorithm's Apriori property, which is supposed to help with the pruning of weak rules, the algorithm still produces a very large number of rules most of which provide no useful practical insight (Kotsiantis et al. 2006, Valle et al. 2018).

In an effort to mitigate the need to generate and search through a large number of rules, (Raeder et al. 2011) modelled the transaction data as a product network. Through the network model they reported discovering expressive communities or clusters of products in the data and by using an appropriate "interestingness" measure were able to isolate useful and actionable communities. They also found that the approach of combining both traditional and network-based techniques is capable of generating insights that are difficult to achieve with traditional analysis methods. However, despite some positive results as reported, the Apriori remains the most practical. Perhaps, the most promising method for dealing with ARM is the Frequent Pattern (FP) Growth algorithm developed by (Han et al. 2000).

The FP Growth algorithm uses a divide and conquer approach to ARM and is efficient both in terms of computer memory and time as it conducts only two scans of the database compared to multiple scans required by the Apriori algorithm (Kavitha et al. 2016). Unfortunately, the drawback with the FP Growth algorithm is that it generates a huge number of conditional FP Growth trees which renders it impractical in the face big data.

In summary, we have seen that the Apriori algorithm provides relevant rules but when the database size increases, its performance deteriorates. This is because it has to scan the entire database each time it scans a transaction. The Apriori algorithm uses a breadth first search technique to get the large item sets. The consequence of this is that the algorithm cannot be applied directly on large databases. The other well-known competing algorithm to the Apriori algorithm is the FP Growth Algorithm which employs the divide-and-conquer technique to form trees of frequent patterns and computes the frequent item sets. When the two are compared, FP Growth outperforms the Apriori in terms of efficiency but generates a large number of conditional FP Growth trees which makes it impractical in the face of large databases.

2.1.2 Recommendation Systems

Recommendation Systems are machine learning systems used mainly by online retailers and digital entertainment platforms in order to drive incremental revenue or increase usage of the platforms. The main goal of Recommendation Systems is to help users or customers discover relevant items such as movies to watch, books to read or products to buy, so as to create a delightful user experience by filtering relevant content (Madadipouya et al. 2017).

When you search online for the keywords: "Recommendation Systems", you are likely to see the now famous, "Netflix prize" pop up. In 2006 Netflix made a data set available with one million movie reviews and offered the team that could improve Netflix's own Recommendation System by 10% a cash prize of $1,000,000. Several teams from around the world entered the competition and eventually one team was awarded the prize by narrowly beating the next best team on prediction accuracy. The point is that, Recommendation Systems are playing a pivotal role in improving revenue and market share for many companies such as: Amazon, Google and Apple to mention a few.

e-Commerce platforms were some of the earliest adopters of Recommendation Systems. These platforms like Takealot or Amazon, recommend products based on users clicking patterns on the website. That is, when a user opens the e-Commerce website, each subsequent product a user views, is stored in a database. Using this information and comparing it to similar users, the platform provides recommendations on relevant products that the user might be interested in. These types of real time recommendations play an incredibly large role in e-commerce and according to (Koren et al. 2009) 35% of Amazon sales are based off of them.

The entertainment industry is another that widely uses Recommendation Systems on their online platforms to recommend relevant content. Examples of this is Youtube which creates a playlist (a recommendation) of videos to follow in play after the video the user is currently watching based on the users profile. Here profile refers to the users viewed history and patterns. By collaborating different users' preferences these platforms make inferences and try to predict what a user might like to view next. Yet again attempting to create a delightful user experience and at the same time trying to keep the user online as long as possible, which, in turn, drives incremental revenue.

Recommendation Systems have come a long way and use different types of models and algorithms. The two prominent methods which are also relevant to our study are: Collaborative Filtering and Matrix Factorization methods. In the next sections we shall discuss Collaborative Filtering and thereafter Matrix Factorization.

Collaborative Filtering

Collaborative Filtering is described by (Koren et al. 2009) as a neighbourhoods method and was introduced by the creators of Tapestry (Goldberg et al. 1992) the first Recommendation System. In Collaborative Filtering systems the items recommended to a user are those preferred by similar users. The "collaborative" part in the word "collaborative filtering" involves identifying other users with similar attributes to a target user. This step helps to determine the preferences of the target user. The "filtering" part involves sub-setting recommendations to remove items already consumed by the user, as to expand the users consumption by suggesting new items.

The basis for most Recommendation Systems is the relationship between users and items. The relationship is expressed through a user-item matrix, as illustrated in Figure 2.1 below. The values in these matrices can either be ratings given to items by users or one-hot encoded values indicating which items have been consumed by which users.

		Items						
		1	2	3	4	5	..	p
Users	1	0	1	0	0	0	..	0
	2	0	0	1	1	1	..	1
	3	1	1	0	1	0	..	0
	4	0	0	0	1	1	..	0
	5	1	0	0	0	1	..	0
	:	:	:	:	:	:	:	:
	n	0	1	0	0	0	..	1

FIGURE 2.1: User-item matrix for Collaborative Filtering

In Figure 2.1, each user is represented as a vector of all items, with components as one-hot encoded values representing items consumed by users.

Two popular approaches of implementing Collaborative Filtering methods are item-based and user-based Collaborative Filtering. Both these methods use the user-item matrix displayed in Figure 2.1, although, in the case of item-based Collaborative Filtering, the user-item matrix is transposed.

The first step of the Collaborative Filtering approach is to calculate the cosine similarity between the different vector pairs of the user-item matrix. Cosine similarity is defined as the cosine of the angle between two vectors and is found by taking the dot product of two vectors and dividing the result by the product of the magnitudes of the two vectors, as shown in Equation 2.1 below:

$$Simlarity(p,q) = cos\theta = \frac{p.q}{||p||||q||} = \frac{\sum_{i=1}^{n} p_i q_i}{\sqrt{\sum_{i=1}^{n} p_i^2}\sqrt{\sum_{i=1}^{n} q_i^2}} \tag{2.1}$$

In user-based Collaborative Filtering the outcome of Equation 2.1 on all pairs of vectors is a user-user similarity matrix. On the other hand, in item-based Collaborative Filtering where the transpose of the user-item matrix is used, the outcome of Equation 2.1 on all pairs of vectors results in an item-item similarity matrix.

The process of recommending products to users based on these two approaches is straight forward. For a target user, the system recommends the items liked by users most similar to the target user (user-user) or the items most similar to those liked by the target user (item-item).

In this study, we are particularly interested in item-based filtering which produces the item-item similarity matrix, from which similarity lists can be inferred for any selected target product.

Unfortunately, Collaborative Filtering methods have very serious computational drawbacks due to data sparsity that exists in the user-item matrix (Singh 2020). Sparsity is a condition in which most of the elements in a matrix are zeros.

In the case of the grocery retail setting such as the one considered in this study, the problem of sparsity is exacerbated by the sheer size of the number of different products in the data. The database is made up of millions of transactions and tens of thousands of unique products, with customers only purchasing a hand full of goods at a time, or at most a full trolley. Given this scenario, many zeros exist in the user-item matrix making it difficult for strong similarity between users or items to occur.

In order to overcome the problem of sparsity, latent factor models were later introduced by (Koren et al. 2009). Simon Funk was one of the first people to explore this idea in the Netflix prize competition. The result of his exploration was a method that became known as Matrix Factorization, the subject of the next section.

Matrix Factorization

The motivation behind the development of Matrix Factorization was to overcome the problem of sparsity associated with the user-item matrix (Bokde et al. 2015). Matrix Factorization overcomes the problem of sparsity by making use of two dense matrices U and V as displayed in Figure 2.2 below.

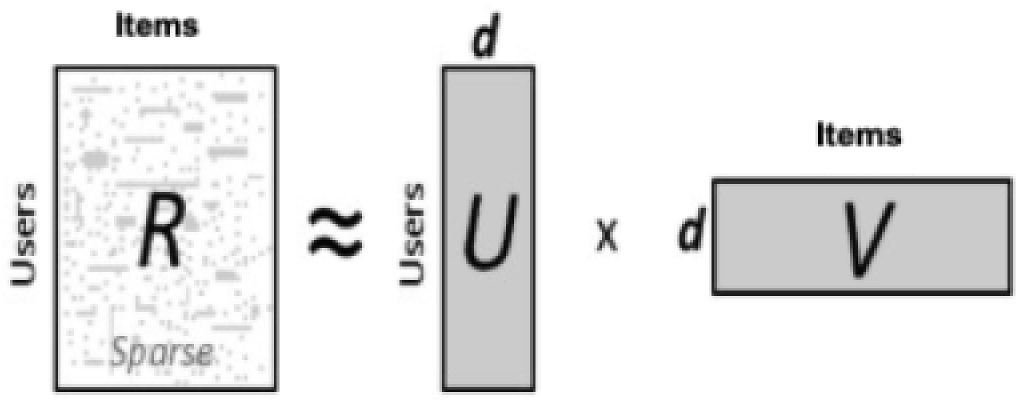

FIGURE 2.2: Illustration of Matrix Factorization

The first matrix, U has latent factors of size d and length equal to the number of users and the second matrix, V which is orthogonal to U, has latent factors of size d and length equal to the number of items.

The idea of representing a sparse user-item matrix, R with two dense latent matrices can also be extended to one-hot encoded user-item vectors.

In order to learn the elements of U and V, first, the two matrices are initialised with random values and then their product is calculated and compared with R using a cost function such as the Root Mean Square Error (RMSE). The values of U and V are then repeatedly adjusted in order to minimize the selected cost function.

This task is achieved by solving the following convex optimisation problem using least squares or gradient descent:

$$minJ = k\frac{1}{2}(\sum_{(i,j)R}(r_{i,j}\, u_i v_j^T)^2 + \lambda({v_i}^2 + {v_j}^2)) \tag{2.2}$$

Here λ is a regularization parameter.

In order to predict if a user will purchase an item, the dot product between user i and item j vectors will result in a single number given by:

$$\hat{r_{ij}} = u_i v_j^T = \sum_{k=1}^{K} u_{ik} v_{kj}. \tag{2.3}$$

The resultant matrix from Figure 2.1 after the optimization process could look as illustrated in Figure 2.3 below after inferring the missing values in the matrix through the optimised product of latent factors.

The result of Matrix Factorization is a dense user-item matrix.

		Items						
		1	**2**	**3**	**4**	**5**	**..**	**p**
Users	**1**	0.8	1	0.4	0.4	0.3	..	0.6
	2	0.6	0.2	1	1	1	..	1
	3	1	1	0.9	1	0.5	..	0.2
	4	0.2	0.5	0.9	1	1	..	0.1
	5	1	0.6	0.1	0.7	1	..	0.8
	:	:	:	:	:	:	:	:
	n	0.8	1	0.3	0.9	0.4	..	1

FIGURE 2.3: Completed user-item matrix (R)

To evaluate the quality of the solution, we can treat some known elements in R as missing during the training process and use them as a test set once the elements of U and V have been learnt. Testing involves finding the dot product of U and V which gives the prediction and then comparing the prediction with the corresponding test value in terms of the cost function.

The new completed matrix now forms the basis from which cosine similarity calculations can be done in order to create the user-user or item-item similarity matrix in order to recommend products. Or in the case of a rating user-item matrix one can just recommend the items that have the highest ratings, after removing the items the user has already consumed.

Although it might seem that this method has overcome the sparsity problem, the optimization process used to infer missing values, suffers when scaling to larger data sets (such as our grocery retail data set) and is highly inefficient (M et al. 2016). The wideness of the user-item matrix also poses difficulty as grocery retailers generally have thousands of different products (Sano et al. 2015).

2.2 Statistical Natural Language Processing (NLP)

Statistical NLP was defined by (Manning 1999) as: "ideas that include automated language processing, probabilistic modelling, information theory and linear algebra". They also argue that the line between this field and what is commonly known as NLP, gets somewhat fuzzy at times and that the latter is more concerned with speech recognition, machine learning and cognitive science. In this section, we shall confine ourselves to Statistical NLP.

Within Statistical NLP we will be concerned with techniques that are categorised as text mining methods. According to (Salloum et al. 2018) text mining approaches are essential in processing text due to the unstructured nature of text data. Structure is needed before the data can be explored for insight and understanding.

In the following subsections we review a number of methods that are used for structuring and pre-processing text data. The models discussed include: N-Grams language models, Bag-Of-Words models and Latent Semantic Analysis (LSA) and can be broadly categorized as Statistical NLP.

2.2.1 N-Grams language model

The N-Gram model is the simplest statistical language model. It is a type of model that assigns probabilities to the sequences of words and is used to predict the next N words in a sentence. Building a N-Gram language model begins with what is known as tokenization which refers to the splitting of sentences into sequences consisting of 1, 2, 3 or N number of words. When the sentence is split into single words, the single word sequences are referred to as unigrams; and when the sentence is split into sequences of two-words, the two-word sequences are referred to as bigrams and so forth. Tokenization can be likened to sliding a window of size k (where $k = 1, 2, ..., N$) from left to right over a sentence so that a window of size $k = 1$ generates unigrams; a window of size $k = 2$ generates bigrams and so on as illustrated in Figure 2.4 below.

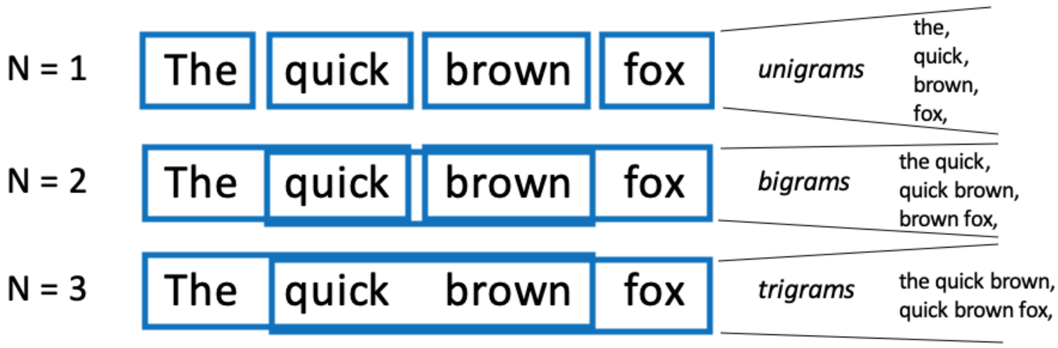

FIGURE 2.4: N-Gram language models illustration

Consider the N-Gram language model where $N = k$. We can predict the probability of seeing the word, w_k, given a history of previous words, $(w_1...w_{k-1})$ using the conditional probability distribution:

$$P(w_k|w_1...w_{k-1}) \tag{2.4}$$

This says that the probability that the next word in a sentence will be w_k depends on the sequence of all preceeding words in the sentence. Computation of this function is extremely difficult as one would have to account for a vast number of combination of words that occurred at any point prior. To make the probability function more practical, we add the assumption that the probability of the next word in a sentence being w_k only depends on the previous word w_{k-1}. This is known as the Markov property. Given this assumption, the new probability function becomes the following:

$$P(w_k|w_1...w_{k-1}) \approx P(w_k|w_{k-1}) \tag{2.5}$$

This is a simpler form of the original function and approximates the history (the context) of the word w_k by considering only the last word of the context. The context used here is of length 1; which corresponds to a bigram model; in general we could use larger fixed-sized histories. However, these lead to increased computation.

To compute a bigram probability of a word w_k given a previous word w_{k-1}, one needs to count the total number of the bigrams (w_{k-1}, w_k) in the text and then normalize it by the sum of all the bigrams that share the first-word w_{k-1}.

The model performs better as N increases but leads to increased computation requirements. The other drawback of the model is that it is a sparse representation of language and as a result words that are not in the training corpus are assigned a probability of zero.

2.2.2 Bag-Of-Words model

The Bag-Of-Words model allows arbitrary text which cannot be understood by a computer to be mapped to a fixed-length numerical vector. It is a model for extracting features from text data for use in machine learning algorithms. The Bag-Of-Words model thus gives structure to unstructured text data.

In this approach, the basic unit of text data is a document and usually represents different levels at which words can be found namely; sentence, paragraph or even chapter level. Each word in the document is regarded as an observation whose count is sought. The total number of unique words in all of the documents or corpus is termed the vocabulary. Bag-Of-Words model is an algorithm that counts how many times a word appears in a specific document. The Bag-Of-Words model results in a matrix where each column in the matrix represents a different word and each row represents a different document of the corpus as illustrated in Figure 2.5.

	the	quick	brown	fox	jumps	over	the	lazy	dog
Document 1	2	1	1	1	1	0	0	0	0
Document 2	1	1	0	0	0	1	1	0	0
Document 3	0	0	0	1	0	0	0	1	1

FIGURE 2.5: Document-term matrix

Word counts allow us to compare documents and gauge their similarities. The use of counts to compare documents finds applications in document search, document classification and topic modelling. The other applications of the Bag-Of-Words model is to prepare text for input to deep-learning neural networks.

The Term Frequency Inverse Document Frequency (TF-IDF) is another approach for comparing documents. The approach was introduced by (Salton et al. 1988) for use in information retrieval with large corpora. TF-IDF works by determining the relative frequency of a word in a specific document compared to the inverse proportion of the same word over the entire document corpus. In other words, TF-IDF measures relevance rather than frequency.

The TF-IDF value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpora that contain the word, which helps to adjust for words that are common to all documents. Currently, TF-IDF is one of the most popular term-weighting schemes, with usage of about 83% in text-based recommendation systems in digital libraries (Beel et al. 2013).

Bag-Of-Words models and TF-IDF have similar drawbacks to those of N-Gram language models; namely, extreme sparsity, high dimensionality and inability to capture high-level semantic meanings behind text data (Zhao et al. 2017). These are all points to which Word2vec has proven to be superior.

2.2.3 Latent Semantic Analysis (LSA)

LSA is both a theory and a method for extracting and representing the contextual-usage meaning of words by statistical computations applied to a large corpus of text (Landauer et al. 1998). This method borrows ideas from TF-IDF and Bag-Of-Words models. Here "latent" means "hidden" and "semantic" the meaning of words in natural language.

The aim of the technique is to summarize the document-term matrix into smaller matrices through the application of Singular Value Decomposition (SVD) which is similar to the idea of Matrix Factorization as explained above. Some have even treated LSA as a dimension reduction technique, which attempts to overcome the problem of sparsity of the document-term matrix. The technique assumes that words with similar meaning occur in similar pieces of text and therefore the decomposition of the document-term matrix still allow for the semantics or meaning of words to remain intact.

The technique has proven to be highly accurate at classifying words from the decomposed matrix (Landauer et al. 1998, Reidy 2009). LSA have been found capable of simulating a variety of human cognitive phenomena, such as: mimicking human word sorting and category judgement, accurately estimating passage coherence and learnability of passages by individual students (Landauer et al. 1998). What does this mean for us? It means that we can derive meaning from natural language by understanding similarity of words, in terms of the context that words appear in.

However, (Manning 1999) notes that the technique still struggles with vocabulary. He argues that the inevitability of sparsity and the loss of word order with this structure, one loses a lot of insight in terms of context of words which in turn affects the meaning of words. For example, if words are arranged differently: ("this is interesting" vs "is this interesting"), or when it comes to synonyms: ("old bike" vs "used bike") these methods tend to struggle.

Furthermore it is not uncommon for text data to be large in size. Implementing SVD on these large corpora of text data is computationally costly, time consuming and sometimes practically infeasible (Bogaardt et al. 2019). The same is true with the Matrix Factorization.

In this chapter we have discussed a number of early methods based on Statistical NLP for modelling text data for use in computational tasks. We have seen that while the methods worked well with small data sets, the methods do not generally scale well computationally. In the next chapter we will introduce newer methods based on neural networks for addressing the problem faced with traditional methods. In particular we are going to focus on a method known as Word2vec which efficiently transforms sparse word vectors into dense vectors that can be used for various NLP tasks.

Chapter 3

Method

In the previous chapter we reviewed a number of methods that are generally regarded as Statistical NLP and that formed the basis on which newer more sophisticated machine learning NLP work is built on. We noted however that these methods are inefficient due to extreme sparsity in high dimensionality and their inability to capture high-level semantic meanings behind text data. In this chapter, we introduce and discuss Word2vec a method that addresses both these problems.

3.1 Word2vec

Word2vec is a method that was developed by a team at Google led by Tomas Mikolov (Mikolov, Sutskever, et al. 2013). It is a Vector Space Model (VSM) which maps words, typically represented by high dimensional sparse vectors - in a NLP task - to low dimension continuous dense vectors. Hence, the name Word2vec, which literally means "words-to-vectors". VSM's can be broadly categorized into two groups: count-based and predictive techniques.

Count-based techniques are the more traditional approaches to solving NLP tasks such as the Bag-Of-Words model and other methods described in Chapter 2. Predictive techniques, on the other hand, involve neural probabilistic language models. These models try to predict a word from its neighbours by using small dense embedding vectors. Prominent among these is Word2vec, GloVe and Fast Text.

The effectiveness of these methods stems from their ability to create representations where semantically similar words are mapped close to each other (Mikolov, Sutskever, et al. 2013). Semantics refers to the meaning of words and where humans interpret this meaning through words, signs and sentence structure. Semantics largely determines our reading comprehension, how we understand others, and even what decisions we make as a result of our interpretations. Neural probabilistic language models are able to capture both the syntactic information and semantics of words and sentences by giving weight to the context in which words appear. Traditional language models could not capture these features, which are core in being able to interpret natural language (Mikolov, Chen, et al. 2013).

Negation is an example where semantics of sentences and words are crucial when interpreting natural language. For example, the word "not bad" might be classified as negative in a sentiment analysis task because the words "not" and "bad" are classified as having a negative meaning, and yet, when put together, the words mean the complete opposite. Hence, the meaning of sentences also depend on the context in which words appear or the neighbouring words that a specific word is associated with.

Consequently, neural probabilistic models significantly outperform count-based language models (Bengio et al. 2003; Schwenk 2007). Word2vec is described as "3 silver bullets" for solving various problems in NLP that have historically been solved by traditional methods (Ma 2018).

3.2 Word2vec model

The Word2vec model is based on a shallow neural network architecture. It accepts the text corpus as input and its output is a set of low dimension dense vectors of real numbers each of which represents a word in that corpus. Although Word2vec is not a deep neural network, its output - the embeddings - can be used as input to deep neural networks. The application of Word2vec is not limited to merely parsing sentences but can be extended to include any type of data in which patterns can be observed or discerned such as genes, documents, etc. It was argued by (Mikolov, Chen, et al. 2013) that simple neural probabilistic language models perform better than complex models on large amounts of data. Word2vec is based on the idea that we can understand the meaning of a word by understanding the meaning (context) of nearby words.

Thus, learning a vector representation of a word, say, w, in the vocabulary, involves learning the context in which the word, w, appears, and the expectation is that words found in similar contexts will have a high degree of similarity. Such a vector representation for the word w is known as a distributed representation, which implies that the meaning of the word w is distributed across all the dimensions.

Word2vec can be implemented in two main ways: either using context words to predict a target word (a method known as Continuous-Bag-Of-Words (CBOW) (Mikolov, Chen, et al. 2013)), or using a target word to predict context words (a method known as the Skip-Gram (Mikolov, Sutskever, et al. 2013)). Figure 3.1 illustrates the two approaches. Context words are words that surround a target word in a string of text and are defined in terms of a window size. For example a window size of 2 would mean, two words before and two words after the target word. The window size is therefore a hyper-parameter that must be set before implementing Word2vec.

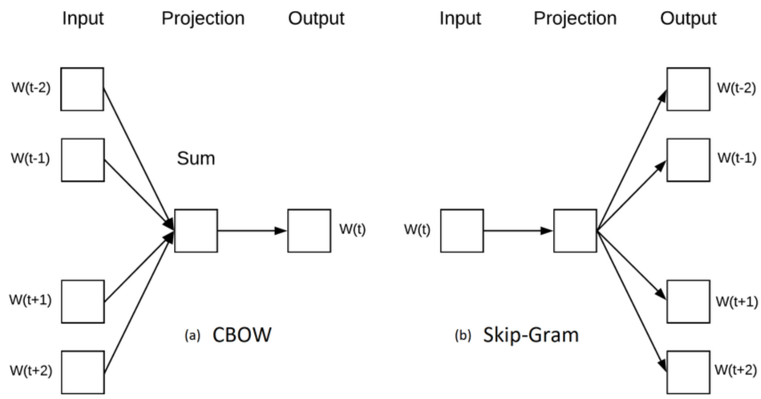

FIGURE 3.1: CBOW and Skip-Gram architecture
https://www.researchgate.net/figure/The-CBOW-and-Skip-gram-architectures-from-15$_{fig}4_3$24014399

Next we discuss the architecture of the Skip-Gram model and how the model works and trained. We have chosen to discuss the Skip-Gram model in detail because it is the most widely used model. In appendix B we present the mathematical foundation of both the Skip-Gram and CBOW model.

3.2.1 Skip-Gram model

As shown in Figure 3.1 the Skip-Gram model is a neural network made up of three fully connected layers: the input, hidden and output layers. The hidden layer has no activation function, the neurons in the hidden layer simply copy the weighted sum of inputs to the next layer.

The dimensions of the input vector is $1 \times V$, where V is the number of unique words in the vocabulary and each word in the vocabulary is represented as a one-hot encoded vector. Thus given that the vocabulary of the corpus is V words, and the dimension of the word vectors is d, the input to the hidden layer connections can be represented by a matrix W_{input} of size $V \times d$ in which each row is a vector representation of a vocabulary word. Similarly, the connections from the hidden layer to the output layer can be represented by a matrix W_{output} of size $d \times V$ where, in this case, each column of the matrix represents a word from the given vocabulary. In other words, the matrix, W_{input}, maps the input $1 \times V$ to the hidden layer with ($V \times d$ dimensional matrix) and matrix, W_{output}, is the weight matrix that maps the hidden layer outputs to the final output layer and is of size $d \times V$. The output from the hidden layer is fed into the softmax function. The dimensions of the output layer is $1 \times V$, where each value in the vector records the probability that the randomly chosen word in the vocabulary is a context word for the given input word.

Learning the Skip-Gram model

To learn the Skip-Gram model, first, the matrices W_{input} and W_{output} are initialized to small random values and pairs of training samples made up of a target word and context words generated from the corpus of text. The number of pairs of training samples generated for a given target word depends on the chosen window size. For example, suppose that the words in the sentence,**"The quick brown fox jumps over the lazy dog"**, make up the vocabulary V of our corpus of text and that a window size of two is selected. This means that $V = 8$, for there are eight unique words.

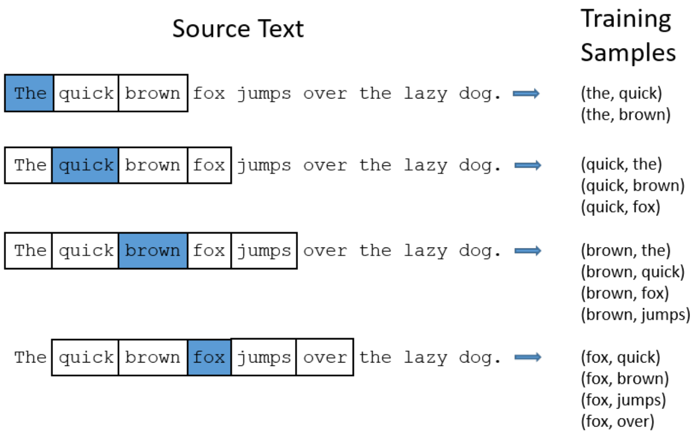

FIGURE 3.2: Word2vec illustration
http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

In Figure 3.2, initially, the target word highlighted in blue is **"The"** while the context words are **"quick"** and **"brown"** resulting in two training samples: namely, **[the, quick]** and **[the, brown]**. After sliding the window by one word to the right as shown in second line of Figure 3.2, **"quick"** becomes the target word while **"The", "brown"** and **"fox"** become the context words resulting in three training samples: **[quick, the]**, **[quick, brown]** and **[quick, fox]**. Continuing in this way, all pairs of training samples corresponding to all the words in the vocabulary can be generated. The network is trained by looping through the training samples and applying stochastic gradient descent and back propagation.

Suppose the training sample under consideration is the first sample, **[the, quick]**. Application of the Skip-Gram model to the training sample, takes the target word, **"the"** (in form of a one hot vector), that is $X_1 = [1,0,0,0,0,0,0,0]^t$ as input. This means that X_1 is multiplied by the matrix W_{input} to yield in the hidden layer a $1 \times d$ embedding vector h_1 of the word **"the"**. Initially, the embedding vector is a poor representation of the word but this improves with more training iterations. Through forward propagation, the product between h_1 and W_{output} yields the output scores for each word in the vocabulary. By passing the scores through a softmax function, the scores are converted into probabilities. The probabilities are compared to the one hot vector, $Y_2 = [0,1,0,0,0,0,0,0]^t$, for the context word **"quick"**. The training objective is to minimize the prediction error between the output and the one hot vector of the context word **"quick"** in the output layer. In our example, the input is the word **“the”**, and the context word we hope to see is: **“quick”** at the output layer. The error that is computed at the end of the output layer is propagated back from the output layer to the input layer through the hidden layer by applying the chain rule. Gradient descent is used to update the weights between these two layers. By repeatedly looping through the training samples and adjusting the weights of the matrix W_{input}, the vector representation of the target words are learned.

The CBOW model is the opposite of the Skip-Gram model. It is concerned with predicting a target word given that the input is a set of context words. Apart from slight differences in the implementation of the method the mechanics are essentially very much similar to the Skip-Gram model. In Appendix B we present the mathematical basis for the two methods.

3.3 Analogical reasoning tests

Word2vec does not only model similarities between two words but also concepts such as analogies between pairs of words as shown by (Mikolov, Sutskever, et al. 2013). Surprisingly, this is learned implicitly, while the training process does not necessarily provide any supervised information regarding what words represent or mean. Therefore, it is possible to use word embeddings to answer questions of the following type: "Man is to woman as king is to what?", (Mikolov, Sutskever, et al. 2013) report that the analogy "man is to woman as king is to queen" is approximately described by a parallelogram, in which the resultant vector: Vking + Vwoman - Vman is closer to the vector Vqueen than to any other vector as shown in Figure 3.3.

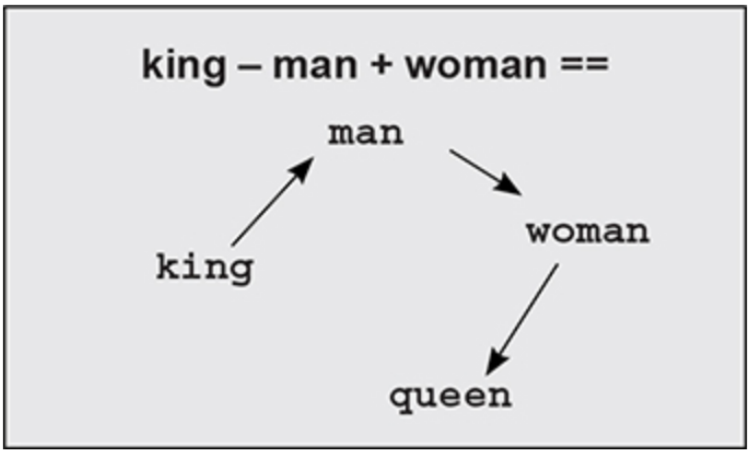

FIGURE 3.3: Analogical reasoning tests, king woman man illustration https://livebook.manning.com/book/exploring-deep-learning-for-language/chapter-4/1

Figure 3.4 shows additional Word2vec results of analogical reasoning tests on various news articles, that amongst other text, contained text referring to country and capital city names. Word2vec successfully preserved the direction of vectors from countries to capital cities. Meaning that the embeddings created by Word2vec accurately modelled whether a word represented a capital city vs a country.

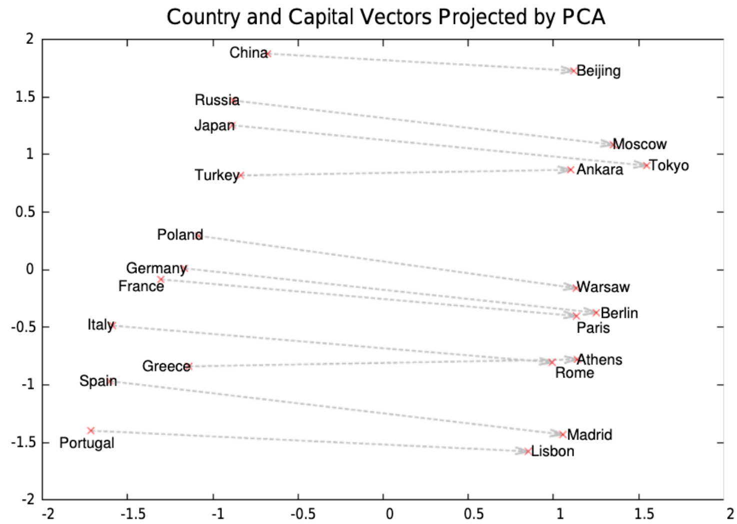

FIGURE 3.4: Analogical reasoning tests, countries and cities illustration (Mikolov, Sutskever, et al. 2013)

The accuracy of analogical reasoning tests is unanimously used to assess the quality of word embeddings in a natural language task (Levy et al. 2015). We hypothesise that implementing similar tests on product embeddings, created on transaction data, will not only help determine the validity of the embeddings but likely result in successful analogical reasoning tests.

Chapter 4

Data Collection and Pre-processing

In this chapter we discuss all matters relating to data. We introduce the origin of the data used in this study as well as the processes we undertook to extract and transform the required data into a format that was suitable as an input object to the models. We also conduct some exploratory data analysis.

4.1 Transaction databases

For the purposes of this study, we were granted access to extract data from a transaction database of a large retailer in South Africa. The transaction database represents a complete history of information collected through EPOS. The information include: identity of the store, store size, location of store, product, brand, customer details and much more. The database can be dissected and combined in many different ways. Figure 4.1 shows an Entity-Relationship (ER) diagram as an illustration of the database architecture.

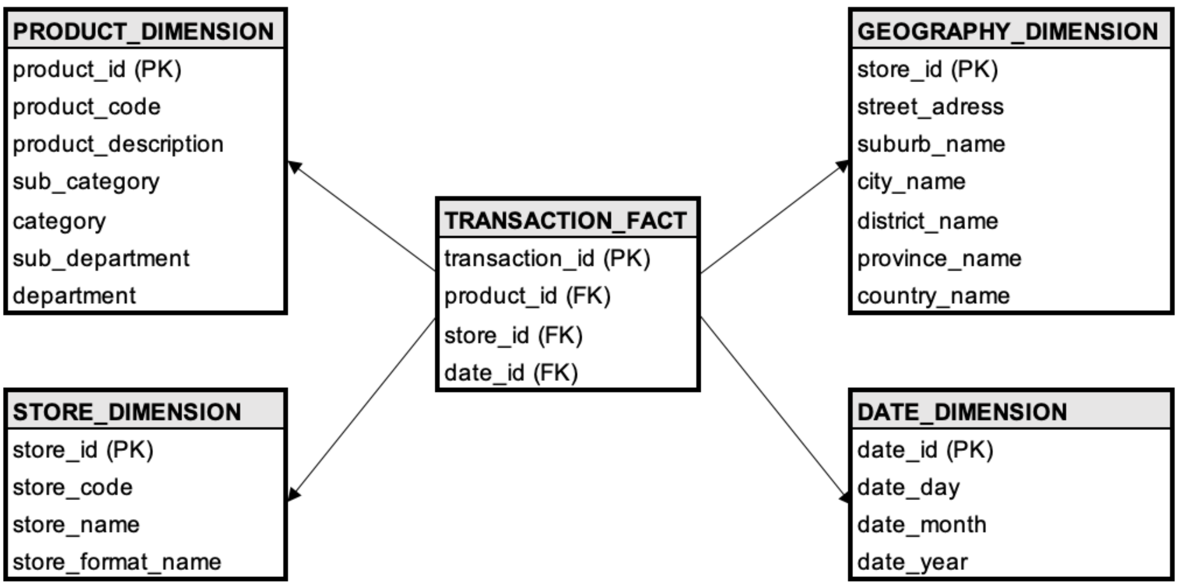

FIGURE 4.1: Database illustration (ER diagram)

Retailer transaction databases like the one in this study, usually consist of two types of data tables. The first type is called the "FACT" table. Fact tables usually contain all the transaction id's as a primary key, and all other ids, that act as links to other tables as foreign keys. The second type is called the "DIMENSION" tables. Dimension tables contain additional information that relate to each transaction. Furthermore, dimension tables primarily have one primary key that has a one-to-many relationship with the corresponding foreign key in the fact table. This relationship allows for the tables to be joined.

The idea behind this sort of architecture is that data can be stored at different levels of granularity, so that the amount of data that needs to be stored is reduced as dimension tables only contain unique records. Secondly, if all records were stored in one table, it would become tedious to dissect through hundreds of columns with duplicated information across rows for every transaction. This duplicated information can simply be stored in a dimension table and then joined to the fact table using an id.

Using these tables and the ER diagram we were enabled to make use of Structured Query Language (SQL) to distil from the transaction database the data required. Furthermore we made use of the dimensions; product, store, geography and date to extract the required data.

4.2 Data selection criteria

In this section we describe the criteria we considered in order to delineate the transaction data into use-able and relevant data samples for the experiments.

Date ranges: At the time of data collection, the database had more than five years of transaction data. We created two data sets from the database, each covering a six month period. The first data set was from 1-January-2018 to 30-June-2018 while the second data set was from 1-July-2018 to 31-December-2018. The first data set is used to train the models, and is referred to as the training data. The second data set is used to validate the results of our models and is referred to as the validation data.

Store format: The database consisted of transaction data from different store formats within South Africa. Store formats refer to the size and range of products in the store. Formats range from a small convenience store to a large bulk buy store and with various sizes in between. We selected the biggest format store, which tends to attract a customer that typically buys a large and diverse basket of goods. This is in contrast to the convenience store format, where transactions typically consist of 1 or 2 products.

Geography: The Western Cape was selected for the study as it had eight of the biggest stores at the time of the study. Secondly it had a large and diverse market for a wide range of products coupled with a well developed distribution system for local suppliers which meant the same range of products were found in all the stores selected.

Product: Products are distributed differently based on geographical constraints and store formats. Each store format and each region has a slightly different range of products from the next. Products are therefore included based on the region and store format we have selected.

Category/Department: All product categories and departments were included. We therefore did not remove a department such as "NON-FOODS" that occurs less regularly in transactions as this might provide powerful information in terms of the context in which products appear.

After the data had been selected according to the criteria explained above, we were left with the following number of records for the training and validation sets.

	Unique Products	Unique Transactions
Training data	48 138	4 895 636
Validation data	49 731	5 082 136

TABLE 4.1: Records per data set

From the table above it is completely understandable that the validation set contained slightly more products and transactions as compared to the training set. This is due to the festive season being included in the validation set.

4.3 Data pre-processing

From each of the two data sets, we created two distinct data frames referred to as the product and transaction data frames.

Table 4.2 shows a snippet of the product data frame for the training data. It contains product codes for the various products in the training data as well as additional information relating to those product codes.

Product Code	Product Description	Sub Category	Category	Sub Department	Department
A90321Z9	MAGAZINE BRAND R WEEKLY	GOSSIP	GOSSIP	MAGAZINES	NON-FOODS
A179899Z4	FRZ VEG MIXED B/NUT&SWT POT BRAND S 1KG PACK	FROZENVEGMIXES	FROZENVEG	FROZENVEGETABLES	PERISHABLES
A782751Z4	BUTTERNUT SPAGHETTI 300G PACK	VEGFRESHCUT	PREPAREDVEG	VEGETABLES	FRUIT & VEGETABLES
A553399Z1	CHIPS ALL GOLD TOM BRAND K 125G	POTATOCHIPSLARGE	CHIPS/NUTS/SNACKS	DRYGROCERIES	GROCERIES
A147920Z7	RUSKS MUESLI BRAND T 500G PACK	RUSKS0	BISCUITS	DRYGROCERIES	GROCERIES
A686236Z7	BOTTLE BRUSH SPONGE TPR	SCRUBBERS	BRUSHES&SCRUBBERS	BRUSHWARE	NON-FOODS
A514832Z5	DRIED MANGO JUST FRUIT BRAND U 32G	DRIEDFRUITSMALL	GLACEDANDDRIEDFRUIT	DRYGROCERIES	GROCERIES
A534929Z9	SWEETS MINI WINE GUMS BRAND V 50G PACK	PREPACKSSMALL(50G-500G)	BOXED/BAGGEDSWEETS&CHOCS	DRYGROCERIES	GROCERIES
A808885Z5	RDY MEAL BBQ CHSY WEDGE STREET FOOD 500G	FRESHREADYMEALS	FRESHHEATNEAT	FRESHCONVENIENCE	PERISHABLES
A741281Z1	GARDEN HOSE EXTEND X3 BRAND W 25FT	VERIMARK	DIRECTMARKETING	DIRECTMARKETING	NON-FOODS

TABLE 4.2: Product data frame

This data frame acts as a dimension table in a database. It is essentially a lookup table, where product codes are used to access information on relevant product descriptions, category, department and so on. The lookup table is necessary as no information can be understood from product codes alone, as these are unique and arbitrary numbers.

In the case where we wish to do subsequent analysis on a subset of products, or a specific category or department, the product data frame is useful as it provides information that makes further sub-setting possible. Later we utilise the product data frame to annotate the product embeddings created by the Word2vec model to facilitate our understanding of how products are clustered in vector space in terms of categories or departments.

Transaction Id	Product Code
234229700362998	A100954Z6
234229700362998	A163935Z2
234229700362999	A99733Z9
234229700362999	A337892Z7
234229700363000	A566821Z2
234229700363001	A422628Z9
234229700363003	A406862Z2
234229700363003	A411224Z2
234229700363003	A787679Z2
234229700363004	A142831Z4

TABLE 4.3: Transaction data frame

Table 4.3, is a snippet of the transaction data frame for the training data. The table shows each transaction id in the first column, duplicated for each corresponding product code, occurring in a transaction, in the second column. Each transaction or "basket" contains a varying amount of product codes. The transaction data frame will be used to create the input object required by the Word2vec model.

No customer information other than the transaction information is needed to carry out our experiments. This differs from Recommendation Systems which require customer identifiers to create a user-item matrix. Similarly, brand information that can be found in the product description has been hidden through the use of aliases such as Brand A and Brand B.

4.4 Word2vec input object from transaction data frame

The similarity between text data in documents and products in transactions is so strong that the two can be analysed using the same approaches. In particular, NLP techniques that have been applied to text data to study relationships between words in documents can be similarly used to study relationships between products in transactions. As discussed earlier we have seen that Word2vec is an approach that has shown to be very successful at capturing word relationships in documents.

In the application of Word2vec to a corpus of text, input data consists of a list of sentences, with each sentence being a list of words. Thus, in terms of the transaction data, transactions are analogous to sentences and product codes are analogous to words. A similar input structure to Word2vec for transaction data can be created directly from the transaction data frame.

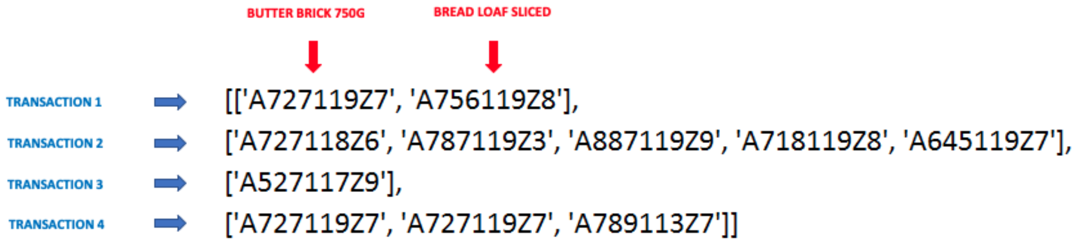

FIGURE 4.2: Input vector (Nested list)

Figure 4.2 illustrates this idea well, using a subset of the transaction data frame as an example. From the figure it should be clear that the input object is a nested list or a list of lists. The outer list represents all transactions or baskets and the inner list the product codes that represent the products in the transactions. It should be clear from this illustration that the model does not receive transaction ids, but only sequences of products codes whose positions represent unique transactions.

4.5 Exploratory data analysis

The nature of transaction data in terms of sheer size, does not lend itself well to the usual simple descriptive statistical analysis. Nonetheless, most market basket analysis studies begin by considering the following two questions: (1) what are the most popular products and (2) what is the count of products per transaction? The answers to these questions are usually given in form of simple graphs.

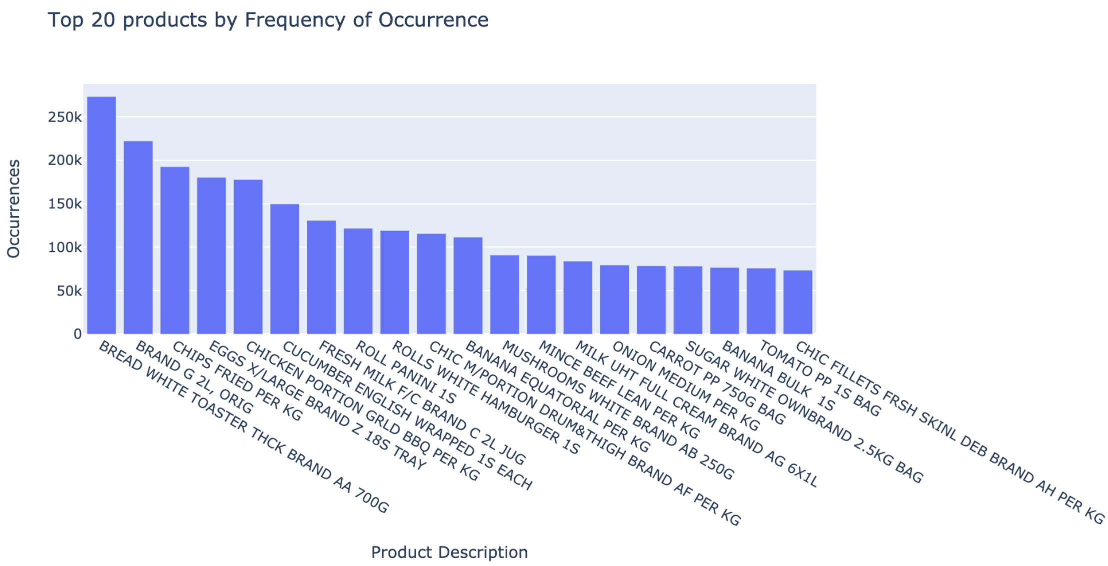

FIGURE 4.3: Highest frequency products

Figure 4.3 shows the top 20 most popular products. Here we see that "BREAD WHITE TOASTER THICK BRAND AA 700G" is the most popular, appearing in approximately

250 000 transactions. This amounts to roughly 5% of all transactions in our training data.

Since the training data has approximately 50,000 unique products it is clear that this graph will have a considerably long tail with some products only appearing a handful of times as compared to the highest frequency products. This information is crucial as the long tail phenomenon leads to generalisation and computational problems due to sparsity.

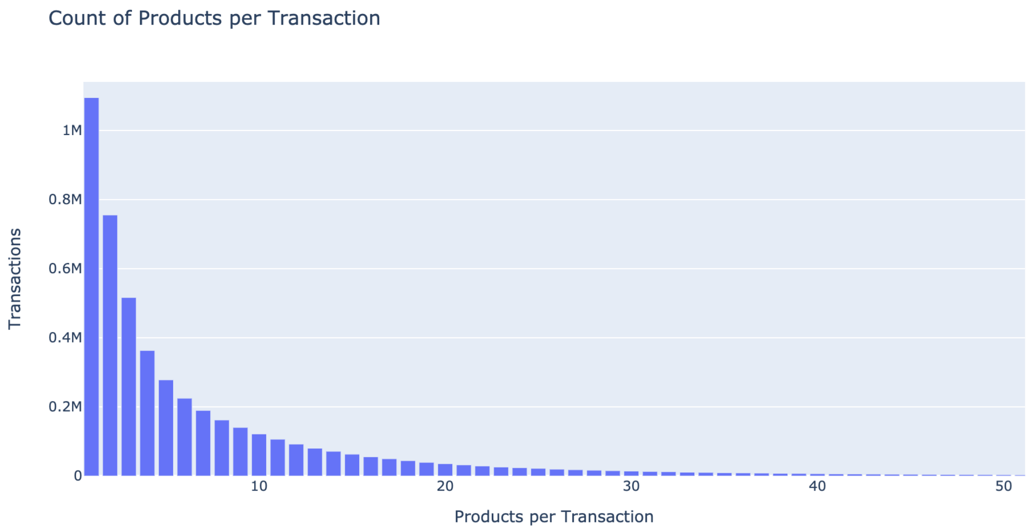

FIGURE 4.4: Count of products per transaction

Figure 4.4 addresses the second question. It shows that most transactions contain very few products. Approximately, a quarter of all transactions are single product transactions, meaning the customers in these instances only purchased one product. We have seen that Word2vec builds a vector representation of a word based on the context of words surrounding the word. By analogy, we can therefore expect that the application of Word2vec to products would only be meaningful if we considered only transactions with more than one product. We should therefore remove all single product transactions as these do not provide any additional information and could potentially make the model less efficient.

Chapter 5

Experiments

In this section we describe the experiments that were undertaken in order to address the three objectives set out in Chapter 1.

5.1 Experiment 1: Training and validating Word2vec results

In order to address the objectives, requires learning a vector representation of each unique product in the training data using an embbeding algorithm such as Word2vec. It is usually convenient to download a pre-trained model and use its vectors to address tasks such as those expressed in the objectives. However, the application of pre-trained vectors is only reasonable if the vocabulary of the text in the current study is a subset of the vocabulary used to learn the pre-trained model. In our case, the equivalent of words were product codes which do not resemble any of the words used in any of the major pre-trained models. As such we trained a Word2vec model from scratch. For this we used Gensim, a topic modelling library for Python, developed by (Rehurek et al. 2010). Gensim provides a simple API to Google's Word2vec and other word embedding algorithms for training. Training the Word2vec model in Gensim still requires the input data to be a list of transaction ids, with each transaction id being a list of product codes (see Figure 4.2).

Training the Word2vec model from scratch is a challenging undertaking. The challenge arises from two main reasons. The first relates to sparsity of data and the second is concerned with computational resources. Data sparsity is the most common in practice and occurs when the available training sample is not truly representative of the population data. Training a Word2vec model with such data, results in embeddings that fail to capture the complete meaning and contextual usage of the word in the population text. The second problem occurs when too much data is available. In that case, training a Word2vec model to achieve usable embeddings requires inordinately large amounts of computational time and memory both of which are very expensive. This is exacerbated when there are many parameters to work with and fine tune.

Hyper-parameter setting:

The most important hyper-parameters to consider when training a Word2vec model in Gensim whether for text or transaction data are: (1) window size, (2) minimum count, (3) size and (4) sg. A description of each of the hyper-parameters is given below:

- **Window size:** The window size refers to the context length and is defined as the number of products surrounding the target product on the left and right. Thus, setting window size = 5 means that for each target product, the context will be ten products surrounding it, or all products in the same transaction if there are less than ten. As can be seen in Figure 4.4, most transactions contain fewer than ten products, so the shopping context is captured quite well. As a result, we initially set the window size to 5.

- **Minimum count:** Minimum count is defined as the minimum number of times a word (a product in our case) must appear in a corpus (transaction data frame in our case). We arbitrarily set this hyper-parameter to 1000. This means that a product that appeared less than 1000 times in the transaction data frame will not have an embedding created for it and will therefore be ignored.

- **Size:** Size refers to the number of dimensions Word2vec maps its words to. Regarding the number of dimensions, there is a trade-off between accurately representing the products (i.e., using one dimension for every product in the data set) and efficiently training the model. According to (Rehurek et al. 2010), using higher-dimensional vectors will improve accuracy, but the marginal benefit of adding more dimensions decreases given a fixed training data set. In terms of running time, doubling the vector dimensions will lead to an increase in estimation time that is similar to the effect of doubling the training data size. Here, as in many previous research work, we selected 100 and found that is was adequate, as changes in this value did not make much of a difference in the output we were seeing.

- **sg:** The sg parameter specifies which Word2vec model architecture to use between Skip-Gram and CBOW. We studied the output from the two different options and found that the differences were very small. We found that the different model architectures seemed to be mirror images of each other when viewed in two-dimensions. Although the product embeddings had different values, the relationship between different product embeddings held firm, meaning the same products were found in the similar space from one model architecture to the other. However, a number of researchers (Ozsoy 2016, Kuzmin 2017) have shown that the Skip-Gram performs better than CBOW when it comes to building product vectors in recommendation tasks. Based on these findings we went ahead with the Skip-Gram architecture.

The training process:

In order to lessen the time of training Word2vec on transaction data, we initially set the most important hyper-parameters of Word2vec to those that have been shown by other researchers to work well as indicated above. In particular, we set window size to 5, size to 100, minimum count to 1000 and sg to 1 meaning Skip-Gram. The model with the initial settings was considered the base model.

The base model was trained on the training data. We then experimented with various different combinations of window size and vector size and each time studied and compared the similarity lists created by the different embeddings in order to determine which output better represented the similarity between the products. A similarity list is created by calculating the cosine similarity between embeddings of a target product and the rest of the products in the training data. A cosine similarity of close to one would indicate products that are similar, in the sense that one can be used as a substitute for the other, whereas a similarity close to zero would indicate that the two products share no commonality between them. We found that the initial settings we adopted, resulted in the best embeddings for the products, that is, embeddings that represent a useful underlying relationship between product embeddings.

Computation of similarity lists between a target product and other products is straight forward and is build into the Gensim library. For example, by providing a target product, Gensim can generate a list of the top most similar products to that target product.

To visualise the possible relationships between products, we use two dimensional scatter plots. In order to do that we need to reduce the dimensions of product embeddings from 100 to 2. The reduction of dimension from 100 to 2 is achieved by a dimension reduction technique known as T-distributed Stochastic Neighbour Embedding (T-SNE) (Hinton et al. 2002). To use T-SNE requires the setting of four hyper-parameters, namely: perplexity, number of components, number of iterations and random state. For our case, values of 40, 2, 2500 and 23 were used respectively. These were selected by iterating through various combinations of values and selecting those that best represented the data graphically.

The Word2vec model is an unsupervised learning method and therefore poses some difficulties in quantifying the extent to which the model is accurate or valid. The approach taken in this study, was to compare similarity lists derived from the base model when the training data was used as input on the one hand, and when validation data was used as input on the other.

Although the two sets of data covered different, but connecting periods, the assumption is that, the buying pattern or distribution pattern of products bought had remained roughly the same - this assumption is not entirely true given that the period from which the validation set was sampled included the festive season. Nonetheless, our expectation is that if the model is consistent and valid, the similarity lists from the two periods would not differ "significantly" from each other.

5.2 Experiment 2: Analogical reasoning tests on product embeddings

The purpose of this experiment is to demonstrate whether product embeddings created by the Word2vec model in Experiment 1 have the ability to accurately categorize characteristics of products that are not provided as input to the model in a supervised manner. For example, in our case, these characteristics include: price and size of the product amongst others.

In order to do so we make use of analogical reasoning tests that were suggested by (Mikolov, Chen, et al. 2013) and discussed at the end of Chapter 3. These tests help to determine whether product embeddings preserve hidden features or characteristics of products that are not necessarily provided for as input to the model. We set up two different experiments in order to carry out these tests. The first type deals with uncovering whether the feature of price has been modelled and preserved in the embeddings while the second deals with whether the feature of package size has been preserved in the embeddings or captured by the model. In the first test, we take arbitrarily selected products that are notorious for being expensive and cheap and in the second experiment we make use of sample products from the carbonated drinks category and try to move from the larger 2 litre products to the smaller 330ml cans.

For each of these experiments we have an expected result, and we will make use of similarity lists to determine whether the actual results are similar or equal to the expected results.

1. The first analogical reasoning test is in the form of an equation below:

$$UpmarketProduct - UpmarketProduct + CheapAlternativeProduct \tag{5.1}$$

   This equation represents three unique products listed below:

$$CoffeeBeans - Salmon + OwnbrandToiletRolls \tag{5.2}$$

   With this equation we would like to determine whether the product embeddings have preserved the characteristic of price sensitivity. From the resultant similarity list of this test we expect to see a list of products that are generally cheaper alternatives as the equation represents the movement from expensive products to cheaper products.

2. The second analogical reasoning test is in the form of an equation below:

$$LargeProduct - LargeProduct + SmallProduct \tag{5.3}$$

   This equation represents three unique products listed below:

$$2LitreOriginal - 2LitreGrape + 0,33LitreOriginal \tag{5.4}$$

With this equation, we expect to see the resultant similarity list to contain majority of products that are 330ML cans or smaller variants of the products used in the test.

5.3 Experiment 3: Comparing relationships between products as created by Association Rules Mining, Recommendation Systems and Word2vec

The purpose of this experiment is to determine whether it is possible to derive from Word2vec, product relationships that have traditionally been created by ARM and Recommendation Systems. In order to do this, we compare Word2vec output to that of ARM and a Recommendation System by examining lists created for an arbitrarily selected target product, where each listing is based on a performance measure of each method. For example, the performance measure used for ARM is support whereas for Recommendation Systems and Word2vec it is cosine similarity.

In the literature review, we discussed a number of disadvantages associated with traditional methods, one of which is, their inability to scale well with large data sets. For example, the amount of data, used to comfortably implement Word2vec in Experiment 1, proved too large to feasibly implement ARM and a simple Recommendation System using Matrix Factorisation. In both cases, any local computer timed out and took days to run on a high performance cluster.

Therefore, in order to proceed with the experiment, where each method would produce satisfactory results for comparison purposes, the sample size of the training data had to be reduced considerably, from six months to approximately one week's worth of transactions, which we refer to as the reduced data set. Apart from that, no additional data pre-processing was done on the reduced data.

Below we describe the specifications that were applied for each model implemented on the reduced data.

- **Association Rules Mining (ARM)**

  In the application of ARM, we are specifically concerned with the support metric of this model, which is expressed by a co-occurrence percentage, given a target product what is the percentage of times other products occurred along with the target product in a transaction. By ranking products by their support values in descending order, we could determine which products have the highest level of complementarity with the target product.

  For this task, we implemented the Apriori algorithm with no hyper-parameters to be set. This is because all we were interested in was to find out the proportion of times a target product appeared together with an arbitrary product in the data set.

- **Recommendation Systems**

  In the application of Recommendation Systems we chose to use the Matrix Factorization algorithm.

  The fact that we do not have user information, transactions id's are used as a proxy for this. The input to the Matrix Factorisation algorithm is a sparse transaction-product matrix created from the reduced data set as described above. The final result is a completed dense transaction-product matrix from which we can compute cosine similarities between products and determine the most similar products to any target product selected.

  Implementation of the Matrix Factorization method requires the specification of a number of hyper-parameters, namely: number of latent factors, number of iterations, learning rate and a regularization parameter. In this case, default values were applied as they resulted in satisfactory latent vectors. The default values used were: 3 for number of latent factors, 100 for number of iterations, 0.00002 for the learning rate and 0.02 for the regularisation factor.

  It was not feasible to run the optimization process on a local computer despite the huge reduction in sample size and as such, we deployed the code on an Amazon EC2 instance, where it ran for just over ten hours. This experience highlights the computational burden associated with Matrix Factorization. The method is clearly inefficient and does not scale.

- **Word2vec**

  The Word2vec model used on reduced data differed slightly from the one used in Experiment 1 in terms of the hyper-parameters. For the Word2vec model on the reduced data we set the hyper-parameters as follows: 1 for sg, 5 for window size, 100 for vector size and 100 for minimum count.

The experiment:

We investigated the output of the three methods by considering "SNACK CHEESE TWIRLS BRAND K 110G" and "BRAND G 2L, ORIG" as the target products. For each method, a list of products with index ranging from 1 to 10, similar to a similarity list, were produced. As before, the target product appears at index 0. The products from index 1 to 10 are the highest ranked products based on a score depending on the method used. For example, for ARM in Table (a) the products are ranked on the support metric whereas, for the Recommendation System in (b) and Word2vec in (c), the products are ranked on cosine similarity. Furthermore we added the sub-category associated with each of these products in order to assist in understanding the output.

Chapter 6

Results

In this chapter we present and analyse the results of the experiments undertaken in Chapter 5.

6.1 Experiment 1: Results

6.1.1 Word2vec similarity lists

To find out if the product embeddings represent a useful underlying relationship between products, we examine similarity lists derived from the base model when the input is the training data set. Similarity lists make it possible to uncover subtle but important relationships between products. A similarity list is obtained by computing the cosine similarity between the embedding of an arbitrarily selected product, hereafter referred to as the target product and the embeddings of each of the other products in the data set. This is then followed by selecting the first K products with the highest scores. The cosine similarity is a measure of how similar a product is to the target product. Note that the cosine similarity score for the target product is always 100% as expected. In this study, we created and studied three similarity lists whose target products covered different food products. The objective was to see if the different target food products led to variations in concepts captured or learnt.

Table 6.1 shows a similarity list with $K = 10$ derived from an arbitrarily selected fresh milk product (highlighted in blue) as a target product and ten of the most similar products to it, in the data set, together with their similarity scores.

Rank	Product Description	Score
0	FRESH MILK F/C BRAND A 2L BOTTLE	100.00%
1	FRESH MILK L/F 2% BRAND A 2L BOTTLE	92.33%
2	FRESH MILK F/C U-PASTURE BRAND A 2L CARTON	86.30%
3	FRESH MILK F/C JUG BRAND B 2L PACK	83.70%
4	FRESH MILK F/C SEAL BRAND A 1L CARTON	79.56%
5	FRESH MILK L/F 2% U-PASTURE BRAND A 2L CARTON	75.15%
6	FRESH MILK F/C BRAND C 2L JUG	74.35%
7	FRESH MILK F/C ECO BRAND AN 2L PACK	73.02%
8	MILK UHT PRISM FC CRM BRAND A 1L CARTON	72.22%
9	FRESH MILK L/F SEAL 2% BRAND A 1L CARTON	70.86%
10	FRESH MILK F/C U-PASTURE BRAND B 2L CARTON	70.33%

TABLE 6.1: Similarity list: Fresh milk

We see that out of the ten products with the highest similarity scores with the target product, only one, ranked number 8, is a non-fresh milk product. All the others are fresh milk products. Interestingly, even the product at rank 8, is still a milk product, the only difference is that it is a long life milk product. It is clear from this example that the model has been able to score highly, those products that have similar characteristics or fulfill the same need as the target product. In other words, the model is able to identify substitute products.

Table 6.2 is similar to Table 6.1. Here the target product is a very popular carbonated drink, referred to as "BRAND G 2L ORIG" (highlighted in blue).

Rank	Product Description	Score
0	BRAND G 2L ORIG	100.00%
1	BRAND H 2L GRAPE	89.67%
2	BRAND H 2L ORIG	88.59%
3	BRAND H 2L ORNG	88.41%
4	BRAND H 2L CRMSODA	84.74%
5	BRAND P 2L REG	83.98%
6	SNACK CHEESE CURLS BRAND J 150G PACK	83.65%
7	BRAND H 2L PINE	82.50%
8	SNACK CHIPS BRAND K 100G PACK	82.30%
9	CHIPS THAI SWTCHILLI BRAND I 125G	82.02%
10	SNACK FLINGS BRAND J	81.55%

TABLE 6.2: Similarity list: Popular carbonated drink

Once again, the model is able to identify products that are known or could be considered as perfect variants or alternatives to the target product. Moreover, the majority of the products so identified are not only alternative carbonated drink products but are all 2 litre in volume. The model here has captured not only the concept of similarity in product type but also the concept of similarity in size. This is remarkable given that in the input this information was not specified.

As we go down the similarity list, the scores decrease, and the products identified as being "similar" to the target product are not carbonated drinks anymore; instead, they are the larger variants of the snacking products like potato chips. These products can reasonably be considered as complements to the target product; that is, items that are frequently bought together with "BRAND G 2L ORIG" carbonated drink.

It seems the similarity measure between a target product and the other products in the data set, in this case, captures both the concept of product substitutes and complements. We see substitutes occurring when the identified "similar" products belong to the same category as the target product. Substitutes also appear to have very high values of similarity scores with their target products.

On the other hand, we see product complements occurring when the identified "similar" products belong to a different category from that of the target product. Complements appear to have lower similarity scores with their target product.

The next similarity list in Table 6.3 is based on a popular steak product referred to as "BRAND Q BEEF RUMP PER KG" (highlighted in blue).

Rank	Product Description	Score
0	BRAND Q BEEF RUMP PER KG	100.00%
1	BRAND Q PORTERHOUSE PER KG	94.99%
2	BRAND Q BEEF RIB EYE PER KG	90.67%
3	BRAND Q BEEF FILLET PER KG	90.48%
4	BRAND Q T-BONE STKHOUSE CLAS PER KG	88.19%
5	BEEF STEAK RUMP PER KG	88.05%
6	LAMB LOIN CHOPS PER KG	88.01%
7	BEEF STEAK T-BONE PER KG	86.12%
8	LAMB RIB CHOPS PER KG	85.76%
9	NATURAL LAMB LOIN CHOPS CERTIFIED LAMB PER KG	84.83%
10	SOSATIES CHICKEN PER KG	83.76%

TABLE 6.3: Similarity list: Steak

In the similarity list we can see variants of steak cuts. We also see other meat products such as lamb and pork. Here it is possible that these meat products could well serve as both substitutes and complements. The idea of lamb and pork being complementary is highly compelling because it is not unusual in South Africa for meat eaters to eat a variety of meat in one setting. However, the more expensive certified lamb loin chops at number 9 in the table, is more likely to be a substitute than a complementary product mainly by healthy conscious customers.

6.1.2 Visualization of product embeddings

Next we make use of the T-SNE method to reduce the dimensions of the product embeddings from 100 to 2 which enables us to visualize the product embeddings in a scatter plot. Figure 6.1 is a scatter plot of the unlabelled product embeddings for the complete product range.

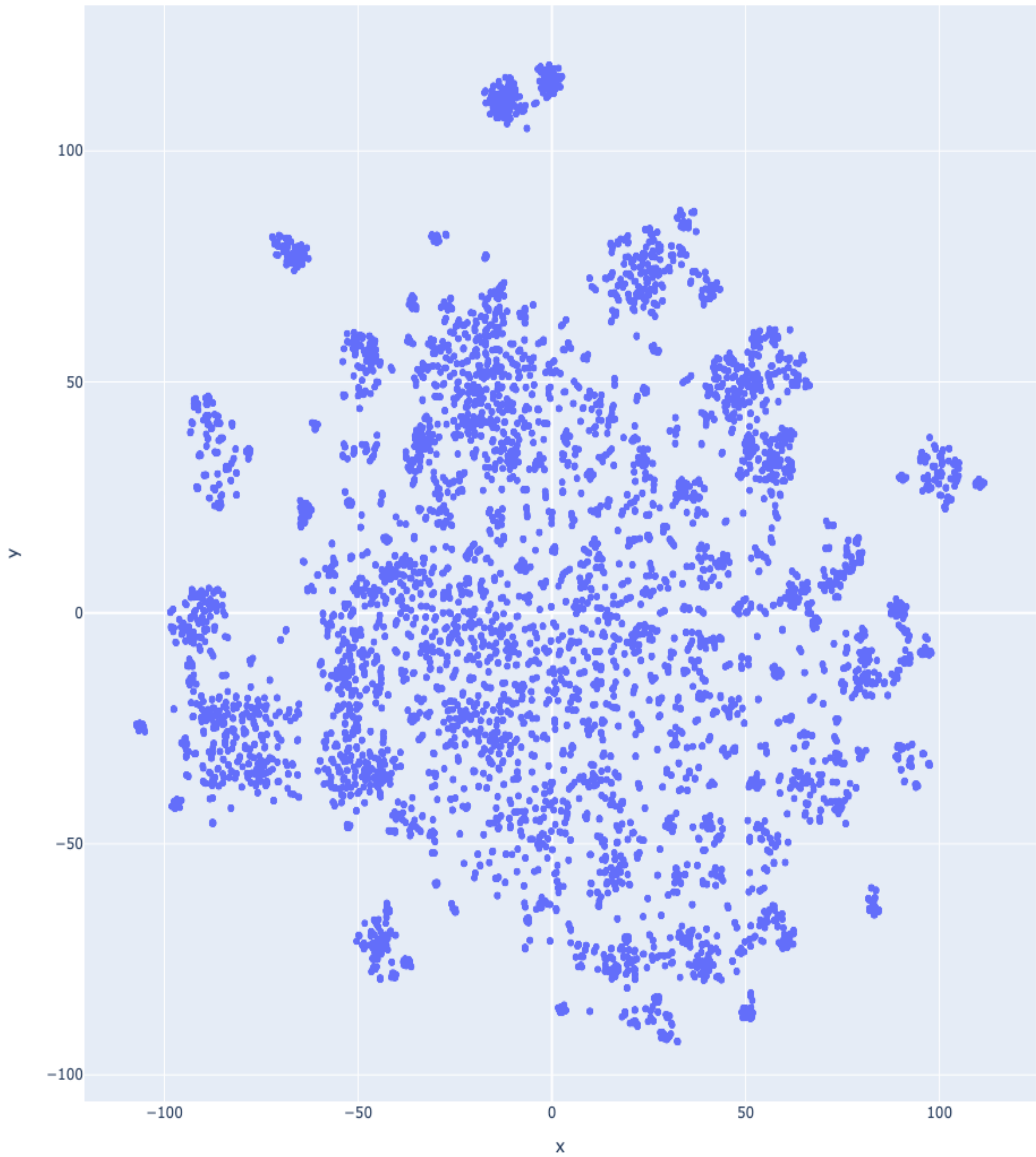

FIGURE 6.1: Scatter plot: Product range

The scatter plot reveals the natural clusters that have been formed by the model. Some of the clusters are so distinct, such as the cluster of points at the top right of the plot. These highly dense clusters indicate high levels of similarity within the cluster.

The next plot Figure 6.2 is a scatter plot of the same product range, except this time we colour the embeddings according to the department to which the product belongs. Department represents the highest level of categorization for products.

FIGURE 6.2: Scatter plot: Department representation

The scatter plot shows that the distinct clusters displayed in Figure 6.1 are mostly groupings of products from the same department. In the top right of the plot, we can see a cluster of non-food products in blue, and in the center, we can see a distinct grouping of fresh meat products in turquoise and a grouping of fruit and vegetables in pink. Grocery items, in red, are fairly scattered throughout the whole product range, forming the largest grouping. Clearly, Word2vec has been able to model and successfully cluster products based on the department they are classified in. This kind of information, as pointed out earlier, was not provided as input to the model, although it seems the model has been able to cluster the products based on this hidden feature of the data very well indeed.

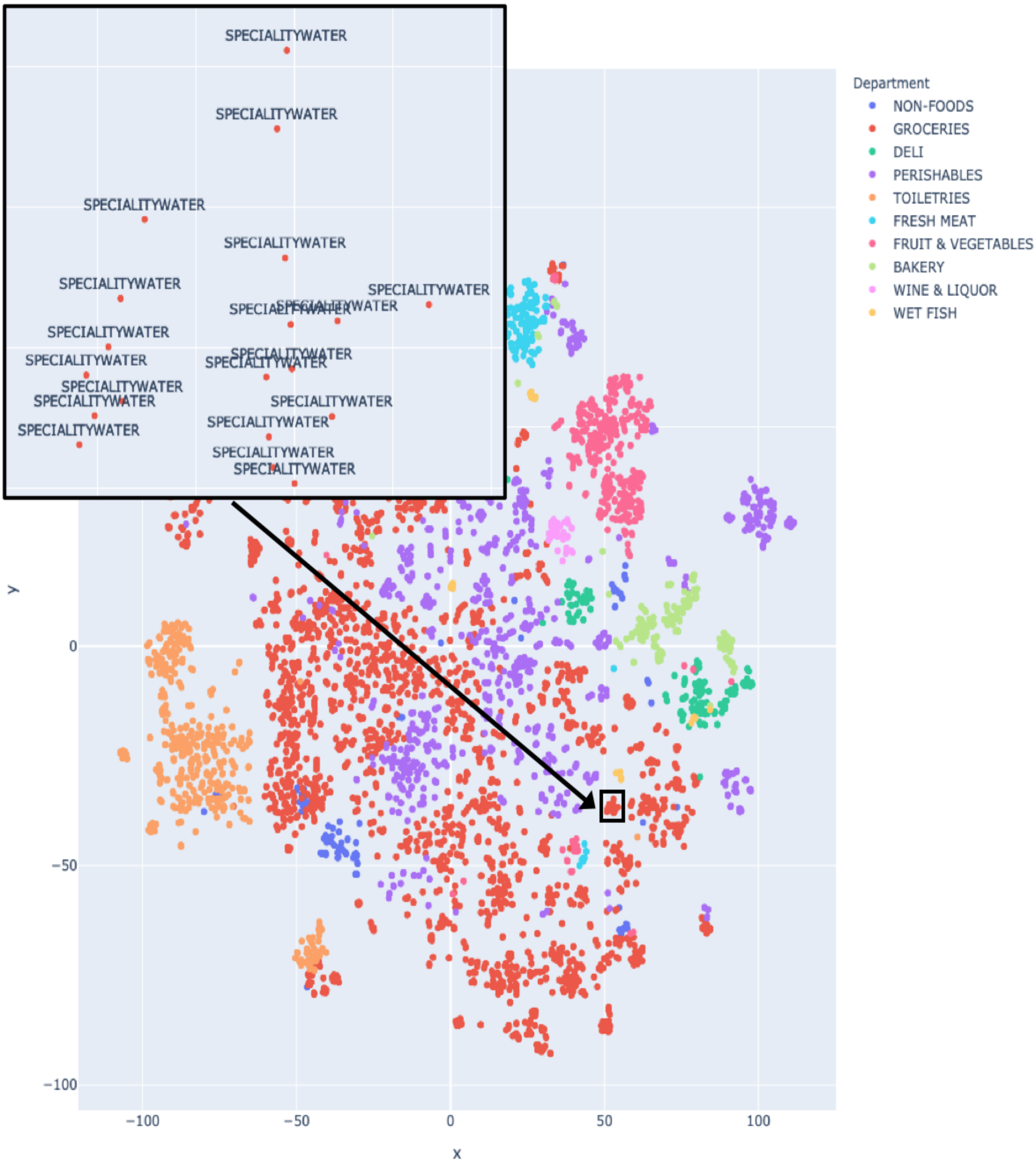

FIGURE 6.3: Scatter plot: Speciality water zoomed

Figures 6.3 through 6.5 show zoomed in clusters of products that form part of the larger groceries cluster. In Figure 6.3 we see that, speciality water products have high similarity between each other and have created a dense cluster in two dimensional space. Once again, confirming that the model produces clusters of products that are similar in characteristic and which are potentially alternatives to each other.

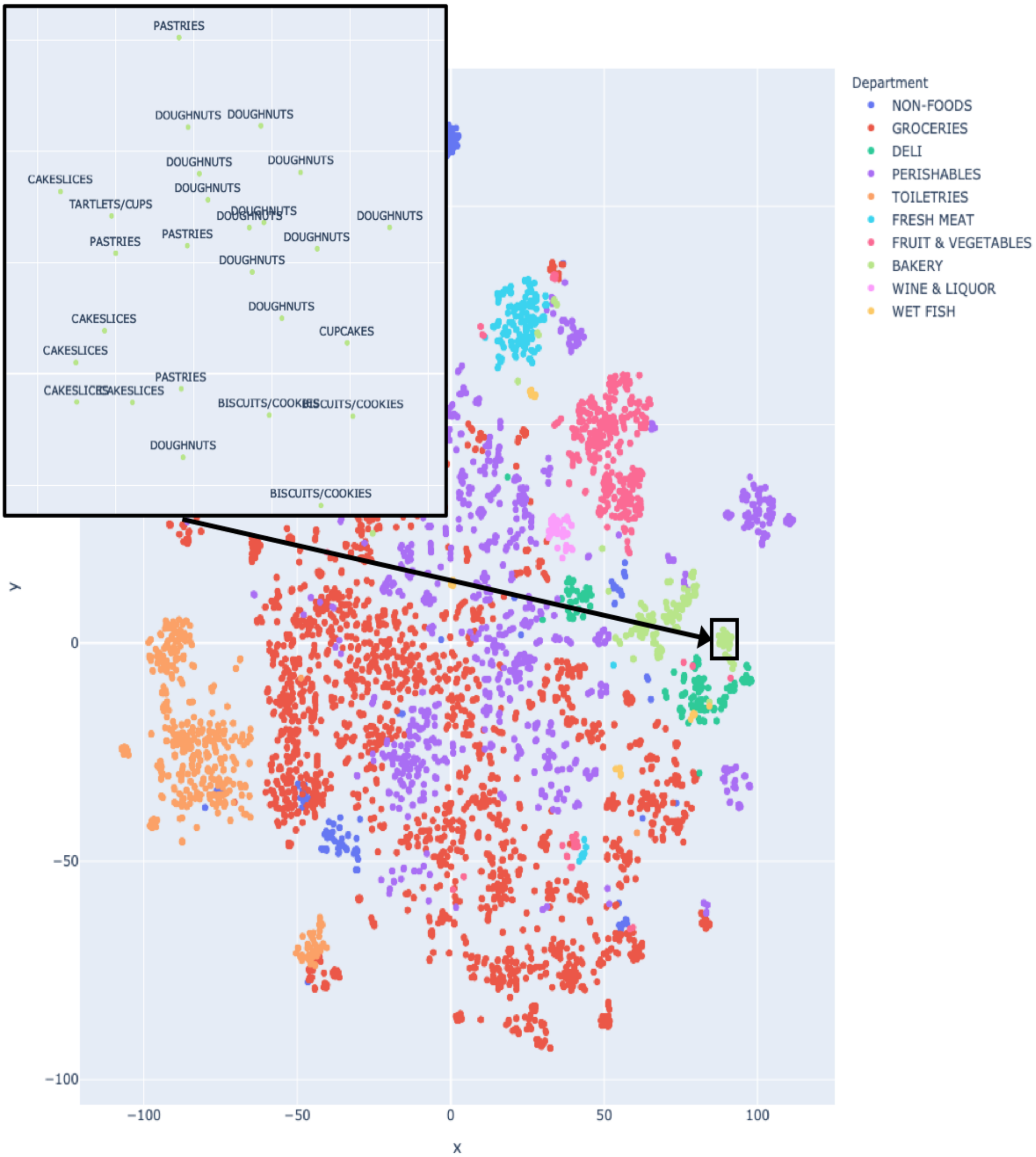

FIGURE 6.4: Scatter plot: Bakery zoomed

Figure 6.4 shows similar results with respect to bakery products such as doughnuts, biscuits and pastries. These products are all found in the bakery section in store.

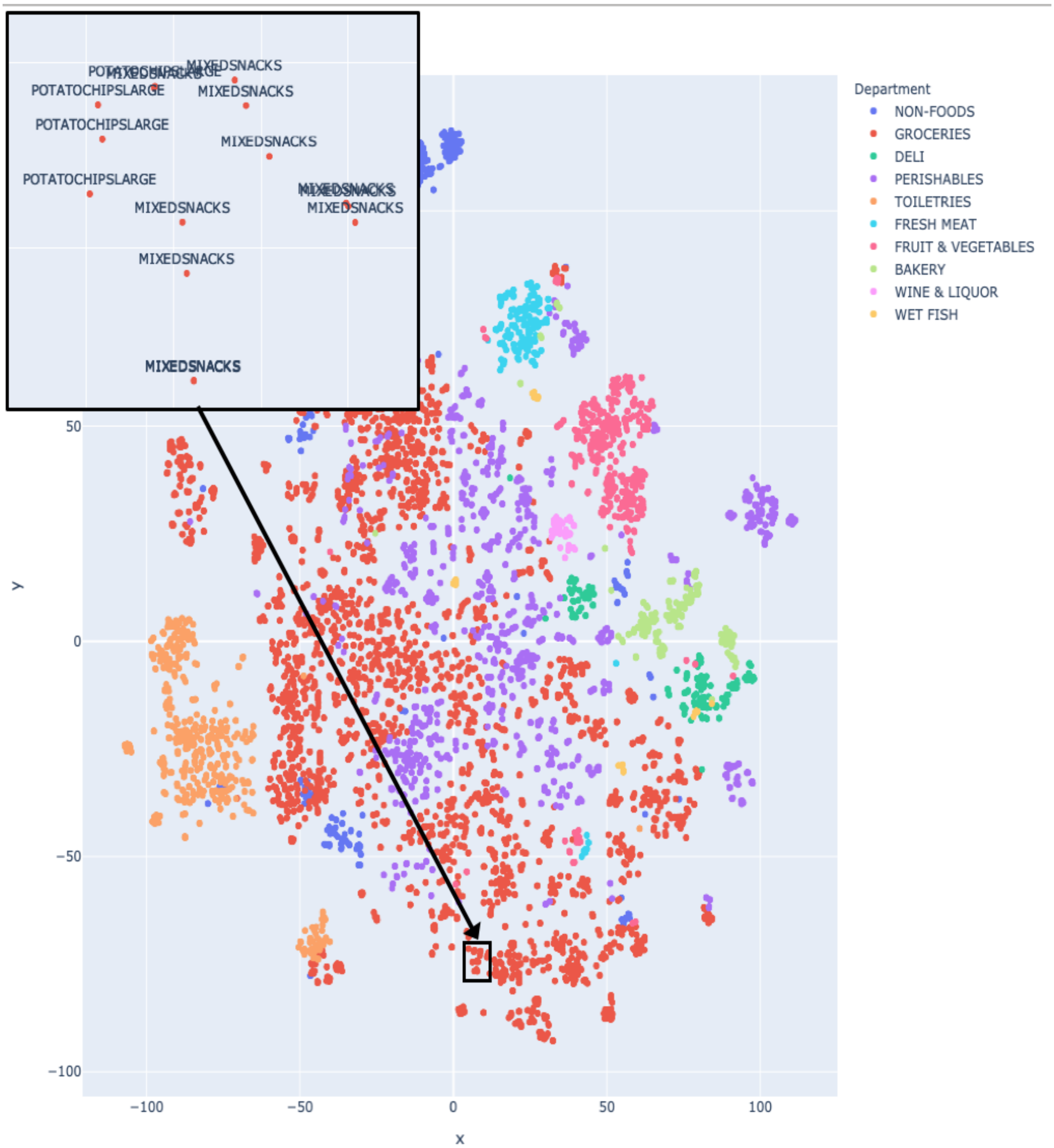

FIGURE 6.5: Scatter plot: Snacking zoomed

Figure 6.5 shows distinct clusters of potato chips and mixed snacks. What is interesting here is that, similar products that are of different type but could fulfil a similar need are clustered near to each other.

6.1.3 Validation of Word2vec results

We validate the results of the model by comparing similarity lists of the same target product but arising from the training data set and the validation data set. By doing so, we try to determine how consistent the model is in creating the similar embeddings for two non-overlapping data sets.

First, we consider the similarity list for a fresh milk product. Tables 6.4 (a) and (b) show similarity lists for a fresh milk product derived from the training and validation sets respectively.

(a) Training Data
1-January-2018 to 30-June-2018

Rank	Product Description	Score
0	FRESH MILK F/C BRAND A 2L BOTTLE	100.00%
1	FRESH MILK L/F 2% BRAND A 2L BOTTLE	92.33%
2	FRESH MILK F/C U-PASTURE BRAND A 2L CARTON	86.30%
3	FRESH MILK F/C JUG BRAND B 2L PACK	83.70%
4	FRESH MILK F/C SEAL BRAND A 1L CARTON	79.56%
5	FRESH MILK L/F 2% U-PASTURE BRAND A 2L CARTON	75.15%
6	FRESH MILK F/C BRAND C 2L JUG	74.35%
7	FRESH MILK F/C ECO BRAND AN 2L PACK	73.02%
8	MILK UHT PRISM FC CRM BRAND A 1L CARTON	72.22%
9	FRESH MILK L/F SEAL 2% BRAND A 1L CARTON	70.86%
10	FRESH MILK F/C U-PASTURE BRAND B 2L CARTON	70.33%

(b) Validation Data
1-July-2018 to 31-December-2018

Rank	Product Description	Score
0	FRESH MILK F/C BRAND A 2L BOTTLE	100.00%
1	FRESH MILK L/F 2% BRAND A 2L BOTTLE	92.37%
2	FRESH MILK F/C JUG BRAND B 2L PACK	85.25%
3	FRESH MILK F/C U-PASTURE BRAND A 2L CARTON	82.86%
4	FRESH MILK F/C SEAL BRAND A 1L CARTON	80.02%
5	FRESH MILK F/C BRAND C 2L JUG	75.87%
6	FRESH MILK F/C ECO BRAND AN 2L PACK	73.51%
7	FRESH MILK L/F SEAL 2% BRAND A 1L CARTON	72.19%
8	BUTTER SPREAD BRAND X 500G TUB	71.83%
9	FRESH MILK F/C U-PASTURE BRAND B 2L CARTON	71.78%
10	FRESH MILK L/F BRAND C 2L JUG	71.01%

TABLE 6.4: Similarity list: Fresh milk validation set

It is clear that both the training and validation sets return consistent results with respect to the products that have the highest cosine similarity with the target product. The products highlighted in green in the validation similarity list are also found in the training similarity list. Despite small variations in the ordering of the top ten products, 80% of the products are the same in the training and validation sets. The slight variation could well be due to sampling error or festive season buying patterns.

Next we consider the comparison of similarity lists for an arbitrarily selected steak product. Tables 6.5 (a) and (b) show the results.

(a) Training Data
1-January-2018 to 30-June-2018

Rank	Product Description	Score
0	BRAND Q BEEF RUMP PER KG	100.00%
1	BRAND Q PORTERHOUSE PER KG	94.99%
2	BRAND Q BEEF RIB EYE PER KG	90.67%
3	BRAND Q BEEF FILLET PER KG	90.48%
4	BRAND Q T-BONE STKHOUSE CLAS PER KG	88.19%
5	BEEF STEAK RUMP PER KG	88.05%
6	LAMB LOIN CHOPS PER KG	88.01%
7	BEEF STEAK T-BONE PER KG	86.12%
8	LAMB RIB CHOPS PER KG	85.76%
9	NATURAL LAMB LOIN CHOPS CERTIFIED LAMB PER KG	84.83%
10	SOSATIES CHICKEN PER KG	83.76%

(b) Validation Data
1-July-2018 to 31-December-2018

Rank	Product Description	Score
0	BRAND Q BEEF RUMP PER KG	100.00%
1	BRAND Q PORTERHOUSE PER KG	94.96%
2	BRAND Q BEEF RIB EYE PER KG	91.77%
3	BRAND Q T-BONE STKHOUSE CLAS PER KG	89.04%
4	LAMB LOIN CHOPS PER KG	88.68%
5	BRAND Q BEEF FILLET PER KG	87.54%
6	BRAND Q STRIP STEAK PER KG	87.32%
7	BRAND Q BEEF SIRLOIN ON THE BONE	87.22%
8	LAMB RIB CHOPS PER KG	85.67%
9	BRAND Q LADIES STEAK PER KG	85.10%
10	NATURAL LAMB LOIN CHOPS CERTIFIED LAMB PER KG	85.06%

TABLE 6.5: Similarity list: Steak validation set

Once again, we can see that similar results are returned in the validation set. This time, about 70% of the products in the top ten of the validation similarity list were present in the top ten of the training similarity list. The model is clearly able to produce similar results on a completely unseen set of transactions.

6.2 Experiment 2: Results

In this section we present the results of analogical reasoning tests that were carried out in Experiment 2. The first analogical reasoning test was formulated as:

$$CoffeeBeans - Salmon + OwnbrandToiletRolls \tag{6.1}$$

The goal was to determine whether the product embeddings preserve the characteristic of price sensitivity. From the resultant similarity list of this test, we expect to see a list of products that are generally cheaper alternatives as the equation represents the movement from expensive products to cheaper products.

Table 6.6 shows the products considered most similar to the results of the equation. The products in the list are all "OWNBRAND" which are generally cheaper alternatives to more expensive products.

Rank	Product Description	Score
1	TOILET ROLLS OWNBRAND 10S PACK	90.12%
2	PEANUT BUTTER SMOOTH OWNBRAND 400G JAR	84.84%
3	SPREAD MED FAT LITE OWNBRAND	81.35%
4	CORNED MEAT OWNBRAND 300G CAN	81.15%
5	SALT TABLE OWNBRAND 1KG PACK	80.76%
6	RICE PARBOILED OWNBRAND 2KG PACK	80.14%
7	FRZ VEG MIXED OWNBRAND	79.96%
8	SALT TABLE OWNBRAND 500G PACK	78.05%
9	SUNFLOWER OIL OWNBRAND 2L	78.03%
10	DISHWASH LIQ LEMON OWNBRAND 750ML BOTTLE	77.87%

TABLE 6.6: Similarity list: Analogical reasoning test 1

From this example, it seems that the product embeddings have preserved characteristics of products such as price which is not provided as input to the model.

The second analogical test was formulated as:

$$2LitreOriginal - 2LitreGrape + 0,33LitreOriginal \tag{6.2}$$

The goal was to investigate whether product embeddings preserve information relating to the package size of products. From the equation set up above, we expect to see the resultant similarity list to contain a majority of products that are 330ml cans or smaller variants of the products used in the test.

Rank	Product Description	Score
1	BRAND G 330ML CAN	91.46%
2	BRAND G 330ML, ORIG	75.60%
3	BRAND H CAN 330ML, ORNG	72.56%
4	BRAND G 330ML CAN, ZERO	71.51%
5	BRAND H CAN 330ML, GRAPE	70.11%
6	BRAND H GINGER BEER 330ML CAN	69.31%
7	BRAND G 330ML CAN, LITE	68.07%
8	ROLL BOEREWORS CHAMPION EACH	67.54%
9	WATER STILL BRAND AK 500ML BOTTLE	62.75%
10	FRESH JUICE ORANGE INSTORE 500ML	62.62%

TABLE 6.7: Similarity list: Analogical reasoning test 2

Table 6.7 shows that the model has been successful in returning products that are of small variants to the target product. It is clear that the result from this analogical reasoning test corroborates with the expected result.

6.3 Experiment 3: Results

In this section, we present results of experiments in which we investigate the relationship between results obtained by Word2vec and those obtained by a generic Recommendation System and ARM. First we discuss the results for a "MIXEDSNACKS" product referred to as "SNACK CHEESE TWIRLS BRAND K 110G".

(a) Association Rules Mining

Rank	Product Description	Sub Category
0	SNACK CHEESE TWIRLS BRAND K 110G	MIXEDSNACKS
1	BREAD WHITE TOASTER THCK BRAND AA 700G	BREADWHITE
2	BRAND G 2L, ORIG	BOTTLES2LT
3	EGGS X/LARGE BRAND Z 18S TRAY	EGGSEXTRALARGE
4	CUCUMBER ENGLISH WRAPPED 1S EACH	CUCUMBER
5	CHIC M/PORTION BRAND AF PER KG	FRESHCHICKENPORTIONS
6	MILK UHT FULL CREAM BRAND AG 6X1L	UHTMILKFULLCREAM
7	MAYONNAISE BRAND AC 750G BOTTLE	MAYONNAISE
8	SNACK CHEESE BRAND K 150G	MIXEDSNACKS
9	EGGS LARGE OWNBRAND 18S TRAY	EGGS LARGE 18S
10	BUBBLES CHEESE BRAND AV 100G	POTATOCHIPSLARGE

(b) Recommendation Systems

Rank	Product Description	Sub Category
0	SNACK CHEESE TWIRLS BRAND K 110G	MIXEDSNACKS
1	CHIPS CORN SWEET CHILLI BRAND AP 150G	POTATOCHIPSLARGE
2	WATER STILL BRAND AK	MINERALWATER
3	FIRELIGHTERS STRIKERS 1UN PACK	FIRELIGHTERS
4	SWEETS SBERRY MELON TITANIUM 10S	COUNTLINES
5	BEVERAGE COCOA BRAND AX 250G CAN	BEVERAGES30G-250G
6	SAUCE SWT CHILLI OWNBRAND 500ML PACK	SAUCES
7	AIRFRESH REF AFRICAN SUN BRAND AY 240ML	AIRFRESHENERS-AEROSOL
8	ENERGY DRINK RED BRAND AZ 250ML CAN	CANSENERGYDRINKS
9	COATING BRAND BA GOLD 'N CRISPY 200G PACK	BREADCRUMBS
10	MASH POTATO GARLIC BUTTER 104G PACK	DEHYDRATEDVEG

(c) Word2Vec

Rank	Product Description	Sub Category
0	SNACK CHEESE TWIRLS BRAND K 110G	MIXEDSNACKS
1	BUBBLES CHEESE BRAND AV 100G	MIXEDSNACKS
2	SNACK ORIG BRAND J 150G	MIXEDSNACKS
3	BUBBLES SPARE RIB BRAND AV 100G	MIXEDSNACKS
4	SNACK CHICKEN BRAND J 150G	MIXEDSNACKS
5	SNACK POPCORN WHITE CHEDDAR BRAND K 90G	MIXEDSNACKS
6	MARSHMALLOWS PINK&WHITE BRAND AW 120G	PREPACKSSMALL(50G-500G)
7	SNACK SPICY BEEF BRAND K 150G	MIXEDSNACKS
8	SNACK CHEESE BRAND K 135G	MIXEDSNACKS
9	BUBBLES CHSE CHIVES BRAND AV 100G	MIXEDSNACKS
10	BISCUIT VANILLA SPHERES 180G	ASSORTEDBISCUITS

TABLE 6.8: Comparison of methods (Snack product)

Table 6.8 (a) shows results for ARM. There does not seem to be a specific pattern occurring between products with the highest support values with the target product. We know that these products are ranked based on how often (in percentage terms) they occur with the target product.

The products form a mixed bag of goods. This can be seen from the sub-category that products in this list represent, which are quite different from each other. Also, when these product are compared to those found in Figure 4.3 of Chapter 4, we see that most of the products with high support values are also high frequency products. In other words, at high support values, products for a given target product, always seem to be skewed towards the most popular products. As a result, ARM does not seem to provide truly complementary products when high frequency products are weighted more heavily.

Table 6.8 (b) too shows a variety of products being recommended as most similar to the target product. What is strikingly different, in this case, is that the recommendations are not necessarily the most popular products. Instead, the item similarity in Recommendation Systems attempt to increase the range of products bought by a customer by recommending products that similar to those the customer has bought previously.

Table 6.8 (c) displays the results for Word2vec. As expected, Word2vec produced a list of products that are of similar kind and fulfill similar needs to the target product. All products in the similarity list form part of the "MIXEDSNACKS" sub-category or similar.

From the three tables, we can see that the different methods produced fairly different results, but results that corroborate with what the literature suggests.

The next three tables compare the results of the three approaches (ARM, Recommendation Systems and Word2vec) when the target product is a popular carbonated drink we have analysed in previous experiments referred to as "BRAND G 2L ORIG":

(a) Association Rules Mining

Rank	Product Description	Sub Category
0	BRAND G 2L, ORIG	2L CARBONATED DRINK
1	BREAD WHITE TOASTER THCK BRAND AA 700G	BREADWHITE
2	CUCUMBER ENGLISH WRAPPED 1S EACH	CUCUMBER
3	EGGS LARGE OWNBRAND 18S TRAY	EGGSLARGE
4	CHIC M/PORTION BRAND AF PER KG	FRESHCHICKENPORTIONS
5	ROLLS WHITE HAMBURGER 1S PACK	ROLLSSTANDARD
6	TOMATO PP 1S BAG	TOMATOES
7	WATER STILL BRAND BK 5L BOTTLE	MINERALWATER
8	MINCE BEEF LEAN PER KG	MINCESPECIES
9	EGGS X/LARGE BRAND Z 18S TRAY	EGGSEXTRALARGE
10	BRAND G 2L, ORANGE	2L CARBONATED DRINK

(b) Recommendation Systems

Rank	Product Description	Sub Category
0	BRAND G 2L, ORIG	2L CARBONATED DRINK
1	INSECT MOSQUITO COIL BRAND BB 125G BOX	INSECTICIDESCOILS/MATS
2	POULTRY FOOD GRN BRAND BC 1KG PACK	BIRD/POULTRYFOOD
3	BRAND BD ORIGINAL, 4X2L	2L CARBONATED DRINK
4	TURKISH DELGHT MILK BRAND BE 300G PACK	PREPACKSSMALL(50G-500G)
5	SPICE SHAKER OREGANO SPICE SEASON 22G	SPICES
6	ENERGY DRINK ASSAULT BRAND BF 4X500ML	CANSENERGYDRINKS
7	BABY FOOD FOUR FRUITS BRAND BG 3RD 200ML	BABYFOODSJUNIOR
8	SPICE ENV 3 PERI PERI BRAND BH 7G PACK	SPICES
9	SPECKLED EGGS BRAND BI 125G PACK	PREPACKSSMALL(50G-500G)
10	CURRY POWDER HOT BRAND BJ 100G PACK	CURRYPOWDERS

(c) Word2Vec

Rank	Product Description	Sub Category
0	BRAND G 2L, ORIG	2L CARBONATED DRINK
1	BRAND H 2L, GRAPE	2L CARBONATED DRINK
2	BRAND H 2L, PINE	2L CARBONATED DRINK
3	BRAND H 2L, SPARBERRY	2L CARBONATED DRINK
4	BRAND H 2L, CRMSODA	2L CARBONATED DRINK
5	BRAND H 2L, APL	2L CARBONATED DRINK
6	BRAND H, ROOTBEER 2L BOTTLE	2L CARBONATED DRINK
7	BRAND H, LEMON 2L BOTTLE	2L CARBONATED DRINK
8	BRAND H, GINGER BEER 2L, ORIG	2L CARBONATED DRINK
9	BRAND P 2L, ORNG	2L CARBONATED DRINK
10	BRAND H 2L, LITE	2L CARBONATED DRINK

TABLE 6.9: Comparison of methods (Carbonated drinks product)

Yet again, the results of ARM in Table 6.9 (a) show that the top ten products that are bought together with the target product tend to be popular products too. This fact is further confirmed by results in Figure 4.3 of Chapter 4. There is also no clear pattern arising from the types of products being suggested apart from being the most frequently bought products. It seems that, the support metric, as used in ARM, creates item complements and that these complements are generally skewed by the highest frequency items. This is a similar instance for the most simplistic form of a Recommendation System, which recommends the most popular products to all customers in a blanket manner.

Table 6.9 (b) shows the results of item similarity for the Recommendation System. There does not seem to be a specific pattern between the top ten most similar products to the target product. The products recommended form a mixed bag of goods, that are not always the most popular products. It seems this method is successful in expanding the breadth of a customer's basket by recommending a variety of goods that the customer might have forgotten or not tested in the past.

Table 6.9 (c) once again displays the power of Word2vec as it has produced a similarity list of products that are of similar kind and fulfil the same need. All products in Table 6.9 (c) are various kinds of carbonated drinks, in different flavours. But more importantly we can look deeper and see that all these products are all 2L Bottles and mostly from the same brand too.

Chapter 7

Conclusion

In this study we set out to address three issues. The first was to investigate whether the application of Word2vec to transaction data would result in embeddings that capture any useful underlying relationship between products.

The results from the experiments show that indeed, the embeddings created, when Word2vec is applied to transaction data, capture a very powerful relationship between products. This is acutely observed when the embeddings are plotted on a two-dimension scatter plot following a dimension reduction. The scatter plots display distinct clusters of products that are homogeneous or fulfil similar needs. What is remarkable is that the clusters are learned in a completely unsupervised manner – that is, no information or assumptions about the products are given to the model apriori. Furthermore, compared to traditional methods, Word2vec is incredibly scalable. The model ran on a local computer with more than 30 million transactions within minutes whereas it was impossible to implement traditional methods, on this size of data, as local computers would time out.

Secondly we attempted to validate the results by assessing whether there are similar hidden features between products that can be inferred from analogical reasoning tests as in the case of natural language. We have demonstrated with examples, two instances in which analogical reasoning tests were used with product embeddings involving product price and product size. Based on these findings we can conclude that the model indeed has the ability to characterise hidden features of products that are not provided as input to the model in some supervised manner. This was an indication that the product embeddings created by the model were valid and provided a validation of the results that are similarly present in tasks related to natural language.

The third and final issue was to investigate how the underlying relationship between Word2vec product embeddings compare to the relationship represented by item-complements and item-similarities as produced by ARM and Recommendation Systems.

We found that the three methods have different underlying philosophies. Given a target product, ARM produces a list of products that are dissimilar from each other, but which are bought together most often with the target product. So that the top products on the list typically have higher values of the support measure with the target product.

The output given by the Recommendation System using Matrix Factorization, shows no clear pattern among the products at the top of the list with the highest degree of similarity with the target product. The output contains a variety of products as that of ARM, except that the products do not necessarily have high support values. This makes sense, as the aim of a Recommendation System is to recommend products that not only have the highest degree of similarity but also products that have not been consumed by the user before. Apart from computational drawbacks, this method is appropriate for the task as it recommends new products to the customer.

Meanwhile Word2vec, produces a list of products that are similar to the target product, with those at the top of the list having very high values of the cosine similarity measure and belonging to the same sub-category. Products with high levels of cosine similarity with a target product can be seen as products that are good substitutes for each other, as these products are homogeneous and fulfill similar needs. It is clear that Word2vec has created the idea of product-substitutes that has not been provided by any scientific method on this size of data previously.

Furthermore, it seems plausible that the model can provide insight on products that can be regarded as truly complementary when a similar product is not part of the same sub-category as the target product. From what we have seen, is that the model firstly converges to the idea of product substitution and hence our expectation is that subsequent studies using different data samples and hyper-parameters would only lead to this relationship becoming more clear. Meaning the idea of complementary products appearing in similarity lists are a function of a incomplete data sample or model application.

7.1 Business impact

The question still remains, what does this insight provide the retailer in terms of being able to meet its objectives of maximizing sales and increasing profits? In order to answer this question a useful application for the model output needs to be identified.

To this point we set up a meeting and shared the model outputs with the Chief Foods Buyer of the retailer partnered in this research. Together we discussed the results and tried to determine how the model outputs could potentially inform decisions in the business by automating processes and providing scientific answers to questions that might exist.

Firstly it was clear that the value proposition was found in being able to create a delightful shopper experience and that by understanding the customers needs and thought processes - which can only be derived efficiently through data - the retailer was in turn empowered to meet its objectives. Following the idea that when the customer wins, the retailer also wins.

From the meeting it seemed clear that an area that requires scientific intervention and that might be well suited to make use of the model outputs is that of the product assortment plan and store layout. The product assortment plan refers to the sequence of actual product placements, these product placements are determined at a category level where a category is ranged based on the number of drops (An individual shelf from top to bottom) assigned to the category. The latter is concerned with how these categories are connected to each other in creating aisles of similar products and ultimately the general store flow.

In this meeting we discussed the retailers current strategy in terms of the product assortment plan and store layout. Below we discuss both these areas and a potential intervention that Word2vec provides.

Product assortment plan

We discussed this strategy using a arbitrarily selected category as an example, the fruit juice category. The current strategy of the retailer for the fruit juice category was to make use of a technique called brand blocking. Brand blocking is the scenario where products are merchandised according to their brands. In this retailers case, stores generally had 3 drops available for fruit juices and therefore meant that the first drop contained all the flavours and sizes of brand A, the second drop contained all the flavours and sizes of brand B and similarly for the third drop for brand C. Hereby the 3 largest brands of fruit juices where assorted individually in three different drops. From the discussion it was clear that brand blocking was the default strategy for most other categories.

The retailer argued that their original thinking behind the current brand blocking strategy was that a customers first decision when shopping a fruit juice product or any other category for that matter was that of selecting a preferred brand. This meant that brand blocking products within categories would make it easy for a customer to find their preferred brand and then further optimizing consequent decisions - when choosing a specific flavour and size - by constricting the search area to a single drop. This idea acts very much like recommendations on an online entertainment platform or e-commerce store, where items are filtered or suggested to customers and in turn making a users time searching less.

Traditionally these sort of questions, in terms of how a customers makes decisions and what their hierarchy of preferences are, were answered through the use of surveys and focus group workshops. These sort of interventions are expensive, time consuming and sometimes inaccurate with potential biases that may occur.

With the combined understanding of the business problem at hand and the understanding of what the product embeddings represented it seemed logical that this method might be well suited to answer this business question in a scientific and more efficient manner.

It seems logical that product embeddings created by Word2vec might prove quite the contrary to the belief that customers highest order of preference when shopping a product is the brand. Lets take the fruit juice example, the cosine similarity between fruit juice products in terms of Word2vec may indicate that there are higher levels of similarity between products of similar flavour as opposed to products of the same brand. Meaning that we could find orange juices clustered together and grape juices clustered together regardless of brand or size. This could be an indication to how customers shop products and therefore might be more sensible for a retailer to assort fruit juices by flavour rather than brand blocking.

In Table 6.2 the similarity list of a popular carbonated drink provides a analogy to this question. Here we see that the top 6 most similar products are a variety of different brands and flavours but all of size two litre. This indicates that two litre carbonated drinks are more similar to one another than, various sizes of the same brand or flavour. Using this output we can suggest to the retailer that it might be more sensible to assort carbonated drinks according to size rather than using the default strategy of brand blocking.

The potential benefit of accurately modelling a customers decision making process and hierarchy of preferences are countless. Assorting products based on a measure of product similarity derived from customers shopping habits will potentially mitigate harm that is caused by stock outs which negatively impacts a customers shopping experience and affects the customers perception of the retailer. By adequately understanding which products are most similar to each other and assorting these products next to or near to each other might lead to a scenario where a customer easily picks up a second best option given the case when a desired product is out of stock. This scenario will retain potential lost sales and further soften damage that stock outs have on shopping experiences and retailer perceptions.

Store layout

We mentioned that the store layout represents how aisles are created by placing specific categories next to each other and ensuring the flow of these categories and aisles are logical. To this question similar suggestions can be made to the retailer by once again using the relationships between product embeddings.

We have seen cosine similarities between product embeddings represent which products are clustered together. The same idea can be applied at a category level. We can use Figure 6.2 as an example to answering this question. By calculating cluster centroids for categories, one can subsequently calculate cosine similarities between these centroids. These cosine similarities between category clusters will then represent which categories are the most similar to one another. Once again this measure can be used to place categories with the highest degree of similarity close to one another and those with the lowest degree of similarity furthest from one another. The idea is that these category placements should ultimately lead to aisles where products that fulfil similar needs are created, making it easy for a customer on a specific shopping mission to get in and out. Once again enhancing the customer experience by reducing the search time and area.

From this study we can't say that it is fact that a customer shops a category or store layout in the way that we can infer similarity from product embeddings. The only way that this can be proven is by creating test and control stores in order to test whether the output of this model produces an incremental increase in sales and revenue. Nevertheless we have still found an application of a method to answer a business question in a cheap, fast and scientific manner, that has historically been answered by experience or through surveys and group focus workshops.

Appendix A

Code

```
##############################################################################################################

# GJ DE SWARDT
# MSC DATA SCIENCE THESIS
# GENSIM WORD2VEC

##############################################################################################################
#1.) SETUP

##Import libraries
import pandas
import numpy
import warnings
import gensim.models
from sklearn.manifold import TSNE
import plotly.express
import timeit
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt

##Settings
warnings.filterwarnings('ignore')

##############################################################################################################
#2.) IMPORT DATA

##2.1) Product table
##Read in product table
df_product = pandas.read_csv("~/Documents/projects/thesis/product_table.csv", index_col=[0], dtype='str')

##Rename columns
df_product.columns = ['product_code',
                      'product_description',
                      'sub_category',
                      'category',
                      'sub_department',
                      'department']

##2.2) Transaction table
##View Dataframe
print(df_product)

##Read in transaction table
df_transaction = pandas.read_csv("~/Documents/projects/thesis/training.csv", index_col=[0], dtype='str')

##Rename columns
df_transaction.columns = ['transaction_id',
                          'product_code']

##View Dataframe
print(df_transaction)

##############################################################################################################
#3.) EXPLORATORY DATA ANALYSIS

##Create a dataframe containing metrics of unique counts
##Count uniques
df_unique_products = df_transaction['product_code'].nunique()
df_unique_transactions = df_transaction['transaction_id'].nunique()

##Put in dataframe form
metrics = {'unique_products':[df_unique_products], 'unique_transactions':[df_unique_transactions]}

##Create DataFrame
df_metrics = pandas.DataFrame(metrics)

##View dataframe
df_metrics.head()

##Create a plot to show number of products per transaction
##Number of products per transaction
df_counts_per_transaction = pandas.DataFrame(df_transaction.groupby('transaction_id')['product_code'].nunique())
df_counts = pandas.DataFrame(df_counts_per_transaction['product_code'].value_counts()).reset_index()
```

```
##Rename columns
df_counts.rename(columns={'index':'products_per_transaction', 'product_code':'transactions'}, inplace=True)

##View dataframe
df_counts.head()

##Create a figure object for Count of Products per Transaction plot
fig = plotly.express.bar(df_counts,
                         x='products_per_transaction',
                         y='transactions',
                         labels={"products_per_transaction": "Products␣per␣Transaction",
                                 "transactions": "Number␣of␣Transactions"},
                         title="Count␣of␣Products␣per␣Transaction")

##View figure
fig.show()

##Save figure to repository
fig.write_image('~/GitHub/msc_data_science_thesis/tables_and_figures/data_product_per_transaction.png')

##########################################################################################################################
#4.) MODEL TRAINING

##Create a list
product_list = df_transaction.sort_values(['transaction_id', 'product_code'])[['transaction_id', 'product_code']].values
                              /.tolist()

##Create product corpus
product_corpus = []
sentence = []
new_transaction_fid = product_list[0][0]
for (transaction_fid, prod_code) in product_list:
    if new_transaction_fid != transaction_fid:
        product_corpus.append(sentence)
        sentence = []
        new_transaction_fid = transaction_fid
    sentence.append(str(prod_code))

##Train word2vec model
start = timeit.default_timer()
model = gensim.models.Word2Vec(product_corpus, window=5, size=100, workers=4, min_count=1000, sg=1)
stop = timeit.default_timer()
print('Time:␣', stop - start)

##Save the model
model.save("~/GitHub/msc_data_science_thesis/model/model.model")

##Load model
model = gensim.models.Word2Vec.load("~/GitHub/msc_data_science_thesis/model/model.model")

##Function to return product description rather than product code
def toProductName(id):
    return df_product[df_product['product_code'] == id]['product_description'].values.tolist()[0]

##Function to create similarity list using cosine similarity
def most_similar_readable(model, prod_code):
    similar_list = [(prod_code, 1.0)] + model.wv.most_similar(str(prod_code))
    return [(toProductName((id)), similarity) for (id, similarity) in similar_list]

##########################################################################################################################
#5.) RESULTS
#5.1) SIMILARITY LISTS

##SHC BEEF RUMP PER KG
pandas.DataFrame(most_similar_readable(model, '000001984699'), columns=['product_description', 'score']).head(11)

##FRESH MILK F/C CLOVER 2L BOTTLE
pandas.DataFrame(most_similar_readable(model, '000001425907'), columns=['product_description', 'Score']).head(11)

#COCA COLA 2L BOTTLE
pandas.DataFrame(most_similar_readable(model, '000001422814'), columns=['product_description', 'score']).head(11)

#PEANUT BUTTER SMOOTH YUM YUM 400G JAR
pandas.DataFrame(most_similar_readable(model, '000001545557'), columns=['product_description', 'score']).head(11)

##########################################################################################################################
#5.2) SCATTER PLOT VISUALIZATION

##T-SNE dimension reduction
labels = []
tokens = []

for word in model.wv.vocab:
    tokens.append(model[word])
    labels.append(df_product['product_description'][df_product['product_code'] == word].to_string(index=False))

tsne_model = TSNE(perplexity=40, n_components=2, init='pca', n_iter=2500, random_state=23)
new_values = tsne_model.fit_transform(tokens)

##Add x and y co-ordinates to dataframe
df_tsne_coordinates = pandas.DataFrame({'product_description':labels, 'x':new_values[:,0], 'y':new_values[:,1]})
df_tsne_coordinates['product_description'] = df_tsne_coordinates['product_description'].str.strip()
df_product_tsne = pandas.merge(left=df_product, right=df_tsne_coordinates, how='inner', on='product_description')

##View dataframe
df_product_tsne.head()
```

```
##Plot the product universe unlabelled
fig_scatterplot_universe = plotly.express.scatter(data_frame=df_product_tsne,
                                                  x=df_product_tsne['x'],
                                                  y=df_product_tsne['y'],
                                                  width=1000,
                                                  height=1000)

##View figure
fig_scatterplot_universe.show()

##Save figure
fig_scatterplot_universe.write_image('~/GitHub/msc_data_science_thesis/tables_and_figures/scatterplot_universe.png')

##Plot with Department as coloured
fig_scatterplot_department = plotly.express.scatter(data_frame=df_product_tsne,
                                                    x=df_product_tsne['x'],
                                                    y=df_product_tsne['y'],
                                                    hover_data={'sub_category'},
                                                    labels={'department': 'Department',
                                                            'sub_category': 'Sub␣Category'},
                                                    color=df_product_tsne['department'],
                                                    width=1000,
                                                    height=1000)

##View figure
fig_scatterplot_department.show()

##Save figure
fig_scatterplot_department.write_image('~/GitHub/msc_data_science_thesis/tables_and_figures/scatterplot_department.png')


plotly.express.scatter(data_frame=df_product_tsne,
                       x=df_product_tsne['x'],
                       y=df_product_tsne['y'],
                       color=df_product_tsne['category'],
                       width=1000,
                       height=1000).show()

#######################################################################################################################
#5.3) DENDROGRAM VISUALIZATION FOR HIERARCHY OF NEEDS

##Subset data based on categories of interest
array = ['FRESHMILKFATFREE', 'FRESHMILKLOWFAT', 'FRESHMILKFULLCREAM', 'FRESHMAAS']
products_to_plot = df_product_tsne.loc[df_product_tsne['Sub␣Category'].isin(array)]

cluster_data = products_to_plot.loc[:,['x', 'y']]
cluster_data.head(5)

linkage_method = linkage(cluster_data, 'average')
labels_to_plot = list(products_to_plot['Product␣Description'])

matplotlib.pyplot.figure(figsize=(12, 10))
dendrogram(linkage_method, orientation='right', labels=labels_to_plot, distance_sort='descending', show_leaf_counts=True)
matplotlib.pyplot.show()

#######################################################################################################################
#6.) ANALOGICAL REASONING TESTS

##Coff beans - Salmon + House toilet rolls (successful)
vec = model.wv['000001140402'] + model.wv['000005905777'] - model.wv['000002161507']
model.most_similar([vec])

##Picanha - Pilchards + Frisco
vec = model.wv['000005020889'] - model.wv['000004230058'] + model.wv['000005418202']
model.most_similar([vec])

#NEW: Carbonated drinks
#2L - 2L + 330ML
vec = model.wv['000001422814'] - model.wv['000001423026'] + model.wv['000001664562']
model.most_similar([vec])

#######################################################################################################################
#7.) QUERY PRODUT COMPLEMENTS FROM WORD2VEC

##First order products by sequence of occurence before using predict_output_word

df_test = pandas.DataFrame(model.predict_output_word(['000001425907'], topn=10), columns=['Product␣Code', 'Probability'])
df_test

pandas.merge(df_test, df_product, how="left")

#######################################################################################################################
#8.) ADDITONAL

##Create a dataframe to plot
df_runtime = pandas.DataFrame({'vector_size': [100, 200, 400, 600], 'run_time': [237, 292, 407, 546]})

##View dataframe
df_runtime.head()

##Plot run time for different size of vectors
fig = plotly.express.line(df_runtime,
                          x='run_time',
                          y='vector_size',
                          title="Model␣run␣times␣for␣different␣vector␣sizes",
                          text='run_time',
                          labels={"run_time": "Run␣Time␣(Seconds)",
```

```
                                    "vector_size": "Vector␣Size␣(n)"})

fig.update_traces(textposition='top␣center')
fig.show()

##Save figure
fig.write_image('~/GitHub/msc_data_science_thesis/tables_and_figures/model_run_times.png')

########################################################################################################################
#9.) MODEL SIDE BY SIDE FOR DIFFERENT PARAMETERS

#9.1) Cbow

##Train using CBOW
start = timeit.default_timer()
cbow_model = gensim.models.Word2Vec(product_corpus, window=6, size=100, workers=4, min_count=1000, sg=0)
stop = timeit.default_timer()
print('Time:␣', stop - start)

##T-SNE dimension reduction
labels = []
tokens = []

for word in cbow_model.wv.vocab:
    tokens.append(cbow_model[word])
    labels.append(df_product['product_description'][df_product['product_code'] == word].to_string(index=False))

tsne_model = TSNE(perplexity=40, n_components=2, init='pca', n_iter=2500, random_state=23)
new_values = tsne_model.fit_transform(tokens)

##Add x and y co-ordinates to dataframe
df_tsne_coordinates = pandas.DataFrame({'product_description':labels, 'x':new_values[:,0], 'y':new_values[:,1]})
df_tsne_coordinates['product_description'] = df_tsne_coordinates['product_description'].str.strip()
df_product_tsne2 = pandas.merge(left=df_product, right=df_tsne_coordinates, how='inner', on='product_description')

##View dataframe
df_product_tsne2.head()

##Plot with Department as coloured
plotly.express.scatter(data_frame=df_product_tsne2,
                       x=df_product_tsne2['x'],
                       y=df_product_tsne2['y'],
                       color=df_product_tsne2['department'],
                       width=1000,
                       height=1000).show()


########################################################################################################################
#10.) WORD2VEC ON SUBSET

##Combine product table with new transaction table
df_combined = pandas.merge(df_transaction, df_product, how='left', on='product_code').dropna()
print(df_combined)

##Test
df_combined = df_combined[df_combined['sub_department'].str.contains('DRYGROCERIES') == True]
print(df_combined)

##Remove all single item transactions
df_combined = df_combined[df_combined['transaction_id'].isin(df_combined['transaction_id'].value_counts()/
                          [df_combined['transaction_id'].value_counts()>=2].index)]
print(df_combined)

##Remove non-necessary columns
df_new_transaction = df_combined[['transaction_id', 'product_code']]
print(df_new_transaction)

##New test
df_final = df_new_transaction.head(200000)
print(df_final)

##Implement model on subset
##Create a list
product_list = df_final.sort_values(['transaction_id', 'product_code'])[['transaction_id', 'product_code']].values.tolist()

##Create product corpus
product_corpus = []
sentence = []
new_transaction_fid = product_list[0][0]
for (transaction_fid, prod_code) in product_list:
    if new_transaction_fid != transaction_fid:
        product_corpus.append(sentence)
        sentence = []
        new_transaction_fid = transaction_fid
    sentence.append(str(prod_code))

##Train word2vec model
start = timeit.default_timer()
model = gensim.models.Word2Vec(product_corpus, window=24, size=100, workers=4, min_count=10, sg=1)
stop = timeit.default_timer()
print('Time:␣', stop - start)

##Function to return product description rather than product code
def toProductName(id):
    return df_product[df_product['product_code'] == id]['product_description'].values.tolist()[0]

##Function to create similarity list using cosine similarity
```

```
def most_similar_readable(model, prod_code):
    similar_list = [(prod_code, 1.0)] + model.wv.most_similar(str(prod_code))
    return [(toProductName((id)), similarity) for (id, similarity) in similar_list]

##SNACKS
pandas.DataFrame(most_similar_readable(model, '000005571558'), columns=['product_description', 'Score']).head(11)

##COCA COLA
pandas.DataFrame(most_similar_readable(model, '000001422814'), columns=['product_description', 'score']).head(11)

###########################################################################################################

# GJ DE SWARDT
# MSC DATA SCIENCE THESIS
# MATRIX FACTORIZATION

###########################################################################################################
#1.) SETUP

##Import libraries
import pandas
import numpy
import sklearn
import timeit
from scipy.spatial.distance import pdist, squareform

###########################################################################################################
#2.) IMPORT DATA

#2.1) Product table
##Read in data
df_product = pandas.read_csv('/Users/jdeswardt/Documents/projects/thesis/product_table.csv', index_col=[0], dtype='str')

##Rename columns
df_product.columns = ['product_code',
                      'product_description',
                      'sub_category',
                      'category',
                      'sub_department',
                      'department']

##View Dataframe
print(df_product)

#2.2) Transaction table
##Read in data
df_transaction = pandas.read_csv('/Users/jdeswardt/Documents/projects/thesis/training.csv', index_col=[0], dtype='str')

##Rename columns
df_transaction.columns = ['transaction_id',
                          'product_code']

##View Dataframe
print(df_transaction)

###########################################################################################################
#3.) CLEAN DATA

##Combine product table with new transaction table
df_combined = pandas.merge(df_transaction, df_product, how='left', on='product_code').dropna()
print(df_combined)

##Remove non-necessary departments
df_combined = df_combined[df_combined['sub_department'].str.contains('DRYGROCERIES') == True]
print(df_combined)

##Remove all single item transactions
df_combined = df_combined[df_combined['transaction_id'].isin(df_combined['transaction_id'].value_counts()/
                          [df_combined['transaction_id'].value_counts()>=2].index)]
print(df_combined)

##Remove non-necessary columns
df_combined = df_combined[['transaction_id', 'product_code']]
print(df_combined)

##Create a subset
df_final = df_combined.head(200000)
print(df_final)

###########################################################################################################
#4.) CREATE USER-ITEM MATRIX FOR DISTANCE CALCULATIONS

##Add quantity for pivoting
df_final['quantity'] = 1
print(df_final)

##Create transaction-item matrix
df_matrix = df_final.groupby(['transaction_id', 'product_code'])['quantity'].sum().unstack().reset_index().fillna(0)/
                    .set_index('transaction_id')
print(df_matrix)

##Remove all product columns with 0
df_matrix = df_matrix.loc[:, (df_matrix != 0).any(axis=0)]
print(df_matrix)

###########################################################################################################
#5.) MATRIX FACTORIZATION
```

```
##Matrix factorization function
def matrix_factorization(R, P, Q, K, steps=100, alpha=0.0002, beta=0.02):
    '''
    R: rating matrix
    P: |U| * K (User features matrix)
    Q: |D| * K (Item features matrix)
    K: latent features
    steps: iterations
    alpha: learning rate
    beta: regularization parameter
    '''
    Q = Q.T
    for step in range(steps):
        for i in range(len(R)):
            for j in range(len(R[i])):
                if R[i][j] > 0:
                    # calculate error
                    eij = R[i][j] - numpy.dot(P[i,:],Q[:,j])
                    for k in range(K):
                        # calculate gradient with a and beta parameter
                        P[i][k] = P[i][k] + alpha * (2 * eij * Q[k][j] - beta * P[i][k])
                        Q[k][j] = Q[k][j] + alpha * (2 * eij * P[i][k] - beta * Q[k][j])
        eR = numpy.dot(P,Q)
        e = 0
        for i in range(len(R)):
            for j in range(len(R[i])):
                if R[i][j] > 0:
                    e = e + pow(R[i][j] - numpy.dot(P[i,:],Q[:,j]), 2)
                    for k in range(K):
                        e = e + (beta/2) * (pow(P[i][k],2) + pow(Q[k][j],2))
        # 0.001: local minimum
        if e < 0.001:
            break
    return P, Q.T

##Create input array for function
R = numpy.array(df_matrix)

##Specify parameters
##N: Number of Users
N = len(R)

##M: Number of Transactions
M = len(R[0])

##K: Number of latent factors
K = 3

##Initialize random variables
P = numpy.random.rand(N,K)
Q = numpy.random.rand(M,K)

##Implement model
start = timeit.default_timer()
nP, nQ = matrix_factorization(R, P, Q, K)
stop = timeit.default_timer()
print('Time: ', stop - start)

##########################################################################################################
#6.) CALCULATE COSINE SIMILARITIES

##Create complete matrix
nR = numpy.dot(nP, nQ.T)
nR = pandas.DataFrame(nR)

##Calculation
M_u = nR.mean(axis=1)
item_mean_subtracted = nR - M_u[:, None]
similarity_matrix = 1 - squareform(pdist(item_mean_subtracted.T, 'cosine'))

##Create pandas dataframe
similarity_matrix = pandas.DataFrame(similarity_matrix)

##Snack
##Product of interest
col_idx = df_matrix.columns.get_loc('000005571558')
corr_specific = similarity_matrix[col_idx]

##Create similarity matrix
df_item_similarities_snack = pandas.DataFrame({'Products': df_matrix.columns, 'Score':corr_specific})/
                                    .sort_values('Score', ascending=False).head(11)
print(df_item_similarities_snack)

##Carbonated drink product
##Product of interest
col_idx = df_matrix.columns.get_loc('000001422814')
corr_specific = similarity_matrix[col_idx]

##Create similarity matrix
df_item_similarities_drink = pandas.DataFrame({'Products': df_matrix.columns, 'Score':corr_specific})/
                                    .sort_values('Score', ascending=False).head(11)
print(df_item_similarities_drink)

##########################################################################################################

# GJ DE SWARDT
```

```
# MSC DATA SCIENCE THESIS
# PRODUCT COMPLIMENTS MINING

##########################################################################################################################
#1.) SETUP

##Import libraries
import pandas
import numpy
import plotly.express

##########################################################################################################################
#2.) DATA

#2.1) Product table
##Read in data
df_product = pandas.read_csv('/Users/jdeswardt/Documents/projects/thesis/product_table.csv', index_col=[0], dtype='str')

##Rename columns
df_product.columns = ['product_code',
                      'product_description',
                      'sub_category',
                      'category',
                      'sub_department',
                      'department']

##View Dataframe
print(df_product)

#2.2) Transaction table
##Read in data
df_transaction = pandas.read_csv('/Users/jdeswardt/Documents/projects/thesis/training.csv', index_col=[0], dtype='str')

##Rename columns
df_transaction.columns = ['transaction_id',
                          'product_code']

##View Dataframe
print(df_transaction)

##########################################################################################################################
#3.) PRODUCT COMPLIMENTS MINING

#Select product of choice for compliment mining
##Target transaction containing specific transaction code
df_target_transactions = df_transaction[df_transaction['product_code'] == '000001422814']
df_target_transactions = df_target_transactions[['transaction_id']]
print(df_target_transactions)

##Join back to transaction table
df_new_transaction = pandas.merge(df_target_transactions, df_transaction, how='left')
print(df_new_transaction)

#Combine product table with new transaction table
##Join the tables on Product Code
df_combined = pandas.merge(df_new_transaction, df_product, how='left').dropna()
print(df_combined)

##Remove non-necessary departments
df_combined = df_combined[df_combined['department'].str.contains('NON-FOODS') == False]
df_combined = df_combined[df_combined['department'].str.contains('TOILETRIES') == False]
print(df_combined)

##Remove all single item transactions
df_combined = df_combined[df_combined['transaction_id'].isin(df_combined['transaction_id'].value_counts()/
                         [df_combined['transaction_id'].value_counts()>=2].index)]
print(df_combined)

##Remove non-necessary columns
df_combined = df_combined[['transaction_id', 'product_code']]
print(df_combined)

##Create a subset
df_final = df_combined.head(200000)
print(df_final)

##Add quantity for pivoting
df_final['quantity'] = 1
print(df_final)

##Create transaction-item matrix
df_matrix = df_final.groupby(['transaction_id', 'product_code'])['quantity'].sum().unstack().reset_index().fillna(0)/
                    .set_index('transaction_id')
print(df_matrix)

##Convert the units to 1 hot encoded values
def encode_units(x):
    if x <= 0:
        return 0
    if x >= 1:
        return 1

##Apply function encode_units
df_matrix = df_matrix.applymap(encode_units)
print(df_matrix)

##Add column sum for compliment co-occurences
```

```
df_matrix.loc['total'] = df_matrix.sum()
print(df_matrix)

##Create a dataframe with product description and counts
df_compliments = pandas.DataFrame(df_matrix.loc['total']).reset_index()
df_compliments = df_compliments.sort_values(by=['total'], ascending=False)
print(df_compliments)

##Add total number of transactions for confidence calculation
df_compliments['number_of_transactions'] = df_matrix.shape[0]
print(df_compliments)

##Calculate confidence of each product
df_compliments['confidence'] = df_compliments['total'] / df_compliments['number_of_transactions']
print(df_compliments.head(11))

################################################################################################################################
#4.) PLOT MOST POPULAR PRODUCTS

##Join back to transaction table
df_combined = pandas.merge(df_transaction, df_product, how='left').dropna()
print(df_combined)

##Remove non-necessary columns
df_combined = df_combined[['transaction_id', 'product_description']]
print(df_combined)

##Create dataframe with most popular products
df_most_popular = pandas.DataFrame(df_combined['product_description'].value_counts().head(21)).reset_index()

##Remove first row, complimentary product
df_most_popular = df_most_popular.iloc[1:]

#Rename columns
df_most_popular.rename(columns={'index':'product_description', 'product_description': 'occurrences'}, inplace=True)

##Check dataframe
print(df_most_popular.head(20))

##Plot bar
plotly.express.bar(df_most_popular, x='product_description', y='occurrences', title="Top 20 products by Frequency of Occurrence")

################################################################################################################################
```

Appendix B

Mathematical theory of Word2vec

B.0.1 The Skip-Gram model

The Skip-Gram model assumes that a word can be used to generate the words that surround it in a text sequence. Suppose the text sequence is: “the”, “quick”, “brown”, “fox”, "jumps", "over", "the", "lazy"' and “dog” as shown in Figure B.1 is part of a text corpus. Further, suppose that the target word is "brown" and the window size chosen for the model is 2. Given the target word, "brown", the Skip-Gram model computes the probability of generating the context words:

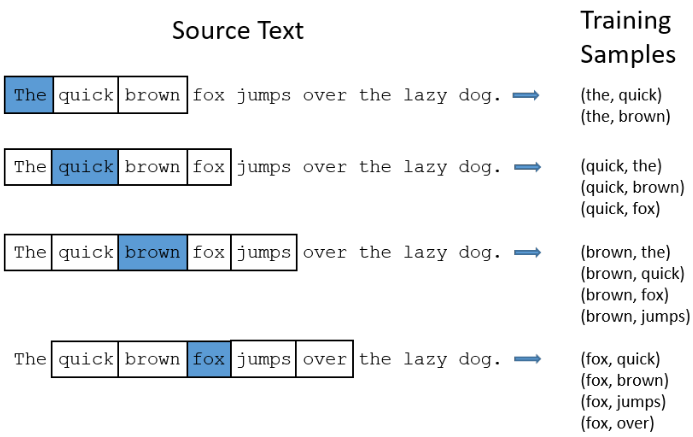

FIGURE B.1: Word2vec illustration
http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

“the”, “quick”, “fox” and "jumps”, that are within a distance of no more than 2 words, given by:

$$P(the, quick, fox, jumps|brown) \tag{B.1}$$

By applying the conditional independence assumption the formula above can be rewritten as:

$$P(the|brown).P(quick|brown).P(fox|brown).P(jumps|brown) \tag{B.2}$$

Now to calculate the probability of context word given the target word, each word must be represented by two sets of vectors, u_w and v_w of dimension d. Use u_w when w is the context word and v_w when w is the target word. Let the context word be denoted as w^c and the target word as w^t. Given this, the probability of generating the context word for any given target word can be obtained by the following:

$$P(w^c|w^t) = \frac{\exp(u_c^T v_t)}{\sum_{i \in V} \exp(u_i^T v_t)} \tag{B.3}$$

where V the set of all unique words or the vocabulary. Assume that a text sequence of length N is given, where the word in position n of the sequence is denoted as w^n. When the context window size is m and using Equation B.2, the likelihood function of the Skip-Gram model is the joint probability of generating all the context words given any target word. That is:

$$\prod_{n=1}^{N} \prod_{-m \leq j \leq m; j \neq 0} P(w^{n+j}|w^n) \tag{B.4}$$

In training the Skip-Gram model, the goal is to learn the parameters of the target word vector and context word vector for each word by maximizing Equation B.4, or equivalently, by minimizing the following loss function:

$$-\sum_{n=1}^{N} \sum_{-m \leq j \leq m; j \neq 0} \log P(w^{n+j}|w^n) \tag{B.5}$$

The optimisation of the loss function is achieved by the application of the SGD algorithm. From B.3 and B.5 we see that the logarithmic conditional probability for the target word vector and the context word vector is give by:

$$\log P(w^c|w^t) = u_c^T v_t - \log\left(\sum_{i \in V} \exp(u_i^T v_t)\right) \tag{B.6}$$

By taking the derivative of the loss function with respect to both u_c and V_t, we get the direction in which we have to move to change the weights in order too maximise the likelihood. It can be shown that the derivative of B.6 with respect to v_t is

$$u_c - \sum_{j \in V} P(w^j|w^t) u_j \tag{B.7}$$

We can similarly obtain the derivative of B.6 with respect to U_c. Once we have both the derivatives, we can use them in our SGD equation to update the weights.

Once training is complete, for any word in the vocabulary, w^n, we obtain its two word vectors v_n and u_n. In applications to NLP tasks, the target word vector v_t in the Skip-Gram model is almost always used as the embedding vector of a word.

B.0.2 Continuous-Bag-Of-Words model (CBOW)

The CBOW model is similar to the Skip-Gram model except it predicts the target word from the context words. Lets take the same piece of text: “the”, "quick”, “brown”, “fox” and “jumps”, in which “brown” is the target word. Given a context window size of 2, the CBOW model is concerned with the conditional probability of generating the target word “brown” based on the context words “the”, “quick", “fox” and “jumps”, that is:

$$P(''brown''|''the'',''quick'',''fox'',''jumps'') \tag{B.8}$$

There are multiple context words in the CBOW model and as such it becomes necessary to average their word vectors and then use the same method as in the Skip-Gram model to compute the conditional probabilities.

As for the Skip-Gram model, let the context word vector be denoted as $v_i \epsilon R^d$ and the target word vector be denoted as $u_i \epsilon R^d$, where i is the index of the word in the corpus. The target word is denoted as w^t and the context words $w_o 1, \ldots, w_o 2m$. Therefore the conditional probability of generating a central target word from the given context word is;

$$P(w_0|W_0) = \frac{exp(u_c^t v_0)}{\sum_{i \epsilon v} exp(u_i^t(v_0)} \tag{B.9}$$

Let the text length be denoted as T and the word at time t be denoted as $w(t)$ with the window size denoted as m. The probability of generating a central target word from context words is thus the likelihood function of the CBOW model;

$$\prod_{t=1}^{T} P(w^{(t)|w^{(t-m)},\ldots,w^{(t-1)},w^{(t+1)},\ldots,w^{(t+m)}}) \tag{B.10}$$

The CBOW models training is quite similar to that of the Skip-Gram model. The maximum likelihood estimation is equivalent to minimizing the following loss function;

$$-\sum_{t=1}^{T} logP(w^{(t)}|w^{(t-m)}, ..., w^{(t-1)}, w^{(t+1)}, ..., w^{(t+m)}) \tag{B.11}$$

so that;

$$LogP(w_c|W_0) = u_c^T \bar{v}_0 - log(\sum_{i \epsilon v} exp(u_i^T \bar{v}_0)) \tag{B.12}$$

Through differentiation, we can compute the logarithm of the conditional probability of the gradient of any context word vector $v_o i (i = 1, \ldots, 2m)$ in the formula above.

We then use the same method to obtain the gradients for other word vectors. Unlike the Skip-Gram model, we usually use the context word vector as the representation vector for a word instead of the central target word.

Bibliography

Agrawal, Rakesh, Tomasz Imieliński, and Arun Swami (1993). "Mining Association Rules Between Sets of Items in Large Databases". In: *SIGMOD Rec.* 22.2, pp. 207–216. URL: http://doi.acm.org/10.1145/170036.170072.

Akter, Wamba (2016). "Big data analytics in E-commerce: a systematic review and agenda for future research". In:

Barkan, Oren and Noam Koenigstein (2016). "Item2vec: neural item embedding for collaborative filtering". In: *2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, pp. 1–6.

Beel, Joeran et al. (2013). "Research paper recommender system evaluation: a quantitative literature survey". In: *Proceedings of the International Workshop on Reproducibility and Replication in Recommender Systems Evaluation*, pp. 15–22.

Bengio, Yoshua et al. (2003). "A neural probabilistic language model". In: *Journal of machine learning research* 3.Feb, pp. 1137–1155.

Bogaardt, Laurens et al. (2019). "Dataset reduction techniques to speed up svd analyses on big geo-datasets". In: *ISPRS International Journal of Geo-Information* 8.2, p. 55.

Bokde, Dheeraj, Sheetal Girase, and Debajyoti Mukhopadhyay (2015). "Matrix factorization model in collaborative filtering algorithms: A survey". In: *Procedia Computer Science* 49, pp. 136–146.

Cooper, Kindra (2019). "Customer Centricity in the Grocery Business According to Kroger CCO". In: URL: https://www.customercontactweekdigital.com/customer-insights-analytics/articles/grocery-retail-customer-centricity.

Duke (2019). "Barriers and Challenges to AI Adoption in Retail". In:

Goldberg, David et al. (1992). "Using collaborative filtering to weave an information tapestry". In: *Communications of the ACM* 35.12, pp. 61–70.

Han, Jiawei, Jian Pei, and Yiwen Yin (2000). "Mining frequent patterns without candidate generation". In: *ACM sigmod record* 29.2, pp. 1–12.

Hinton, Geoffrey and Sam T Roweis (2002). "Stochastic neighbor embedding". In: *NIPS*. Vol. 15. Citeseer, pp. 833–840.

Kavitha, M and ST Tamil Selvi (2016). "Comparative Study on Apriori Algorithm and Fp Growth Algorithm with Pros and Cons". In: *International Journal of Computer Science Trends and Technology (I JCS T)–Volume* 4.

Koren, Yehuda, Robert Bell, and Chris Volinsky (2009). "Matrix factorization techniques for recommender systems". In: *Computer* 42.8, pp. 30–37.

Kotsiantis, Sotiris and Dimitris Kanellopoulos (2006). "Association rules mining: A recent overview". In: *GESTS International Transactions on Computer Science and Engineering* 32.1, pp. 71–82.

Kuzmin, Vitali (2017). "Item2Vec lähenemine soovitussüsteemi jaoks". PhD thesis.

Landauer, Thomas K, Peter W Foltz, and Darrell Laham (1998). "An introduction to latent semantic analysis". In: *Discourse processes* 25.2-3, pp. 259–284.

Levy, Omer, Yoav Goldberg, and Ido Dagan (2015). "Improving distributional similarity with lessons learned from word embeddings". In: *Transactions of the Association for Computational Linguistics* 3, pp. 211–225.

M, Reshma and K R Remesh Raman (2016). "Recommendation System: A Big Data Application". In: *International Journal of Emerging Trends in Science and Technology 2348-9480* 3, pp. 39–46.

Ma, Edward (2018). "3 silver bullets of word embeddings in NLP". In:

Madadipouya, Kasra and Sivananthan Chelliah (2017). "A literature review on recommender systems algorithms, techniques and evaluations". In: *BRAIN. Broad Research in Artificial Intelligence and Neuroscience* 8.2, pp. 109–124.

Manning (1999). *Foundations of statistical natural language processing*. MIT press.

McAlister, Lattin (1983). "IDENTIFYING SUBSTITUTE AND COMPLEMENTARY RELATIONSHIPS REVEALED BY CONSUMER VARIETY SEEKING BEHAVIOR". In:

Micro, Ingram (2018). "6 big data use cases in retail". In: URL: http://www.ingrammicroadvisor.com.

Mikolov, Tomas, Kai Chen, et al. (2013). "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781*.

Mikolov, Tomas, Ilya Sutskever, et al. (2013). "Distributed representations of words and phrases and their compositionality". In: *Advances in neural information processing systems*, pp. 3111–3119.

Nikolova, Hristina Dzhogleva et al. (2014). "The shopper-centric retailer: three case studies on deriving shopper insights from frequent shopper data". In: *Shopper Marketing and the Role of In-Store Marketing*. Emerald Group Publishing Limited.

Ozsoy, Makbule Gulcin (2016). "From word embeddings to item recommendation". In: *arXiv preprint arXiv:1601.01356*.

Raeder, Troy and Nitesh V Chawla (2011). "Market basket analysis with networks". In: *Social network analysis and mining* 1.2, pp. 97–113.

Raza, Kilbourn (2017). "The impact of point-of-sale data in demand planning in the South African clothing retail industry". In:

Rehurek, Radim and Petr Sojka (2010). "Software framework for topic modelling with large corpora". In: *In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Citeseer.

Reidy, Pat (2009). "An Introduction to Latent Semantic Analysis". In:

Salloum, Said A et al. (2018). "Using text mining techniques for extracting information from research articles". In: *Intelligent natural language processing: Trends and Applications*. Springer, pp. 373–397.

Salton, Gerard and Christopher Buckley (1988). "Term-weighting approaches in automatic text retrieval". In: *Information processing & management* 24.5, pp. 513–523.

Sano, Natsuki et al. (2015). "Recommendation system for grocery store considering data sparsity". In: *Procedia Computer Science* 60, pp. 1406–1413.

Schwenk, Holger (2007). "Continuous space language models". In: *Computer Speech & Language* 21.3, pp. 492–518.

Shaukat, Kamran, Sana Zaheer, and Iqra Nawaz (2015). "Association rule mining: an application perspective". In: *International Journal of Computer Science and Innovation* 2015.1, pp. 29–38.

Singh, Monika (2020). "Scalability and sparsity issues in recommender datasets: a survey". In: *Knowledge and Information Systems*, pp. 1–43.

Valle, Mauricio A, Gonzalo A Ruz, and Rodrigo Morrás (2018). "Market basket analysis: Complementing association rules with minimum spanning trees". In: *Expert Systems with Applications* 97, pp. 146–162.

Zhao, Rui and Kezhi Mao (2017). "Fuzzy bag-of-words model for document representation". In: *IEEE Transactions on Fuzzy Systems* 26.2, pp. 794–804.