HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

MASTER THESIS

Mr.
Muhammad Usman Khan

Embeddings for Product Data

2022

Unite

Faculty of **Applied Computer and Life Sciences**

MASTER THESIS

Embeddings for Product Data

Author:
Muhammad Usman Khan

Study Programme:
Applied Mathematics in Networks and Data Science

Seminar Group:
MA19w2-M

First Referee:
Prof. Dr. Thomas Villmann

Second Referee:
Dr. Rudolf Sailer

Mittweida, August 2022

Bibliographic Information

Khan, Muhammad Usman: Embeddings for Product Data, 79 pages, 20 figures, Hochschule Mittweida, University of Applied Sciences, Faculty of Applied Computer and Life Sciences

Master thesis, 2022

Abstract

The E-commerce industry has grown exponentially in the last decade, with giants like Amazon, eBay, Aliexpress, and Walmart selling billions of products. Machine learning techniques can be used within the e-commerce domain to improve the overall customer journey on a platform and increase sales. Product data, in specific, can be used for various applications, such as product similarity, clustering, recommendation, and price estimation. For data from these products to be used for such applications, we have to perform feature engineering. The idea is to transform these products into feature vectors before training a machine learning model on them. In this thesis, we propose an approach to create representations for heterogeneous product data from Unite's platform in the form of structured tabular records. These tables consist of attributes having different information ranging from product-ids to long descriptions. Our model combines popular deep learning approaches used in natural language processing to create numerical representations, which contain mostly non-zeros elements in an array or matrix called as dense representation for all products. To evaluate the quality of these feature vectors, we validate how well the similarities between products are captured by these dense representations. The evaluations are further divided into two categories. The first category directly compares the similarities between individual products. On the other hand, the second category uses these dense vectors in any of the above-mentioned applications as inputs. It then evaluates the quality of these dense representation vectors based on the accuracy or performance of the defined application. As result, we explain the impact of different steps within our model on the quality of these learned representations.

I. Contents

II. List of Figures

III. List of Tables

IV. Preface

This thesis was written as a part of a master's degree at Hochschule Mittwedia University of Applied Sciences. The research topic was suggested by Dr. Rudolf Sailer and Dr. Martin Kleinsteuber together with Prof.Thomas Villmann. I wish to express my sincere appreciation to Dr. Sailer, who gave me an opportunity to work on such an exciting, yet challenging topic and the freedom of selecting research direction according to my personal preferences and strengths.

Many thanks go to Mr. Edward Owen Wall for taking the time to give me valuable insights into the topic from his perspective and fields of expertise. His and Dr. Sailer's continuous guidance and support helped me complete this thesis on time. They were always willing to facilitate me through out the whole research and analysis.

The research conducted at Unite Network SE is presented in this thesis. From Hochschule Mittweida, I would like to thank Dr. Marika Kaden and my first supervisor Prof.Thomas Villmann for proofreading the text and giving timely and excellent feedback. I am sincerely grateful to Unite Network SE for giving me the opportunity to work on this topic.

Over the past two and a half years, I had a great time in Unite Network SE. I would like to thank every one in the department of Analytics for putting trust in my skills and showing great interest in this work and giving helpful suggestions. Here, I found myself next to true experts in an environment full of freedom and friendliness, while I received all the resources and support necessary for accomplishing the thesis.

I would like to give a special thanks to my mother for supporting and motivating me since the beginning of this journey.

Muhammad Usman Khan - Mittweida, August 2022

1 Introduction

1.1 Motivation

With the adaptation of e-commerce and e-business services in recent years, the data generated from these e-channels have grown exponentially [5]. The number of sales for the e-commerce market was recorded at an extraordinary scale as trillions of dollars were spent globally in online retail last year [13]. Despite the growing popularity of e-commerce, customer face numerous issues when it comes to product search and finding the best prices for the same product offered by multiple e-shops. Some companies offer product aggregators for consumers to compare and categorize products across different websites and merchants. However, organizing and navigating products across multiple sources is not a trivial task [66].

The advancements in technology have paved the way to address this issue. Among the intensively studied approaches is to create a system that automatically discovers the representations needed for feature identification or classification from raw data [35]. The performance of machine learning algorithms relies majorly on the underlying features or data representation to which they are applied. These information processing tasks can be termed as easy or difficult depending on how the information is represented. Such feature engineering is important but labor-intensive and sheds light on the weakness of different algorithms, i.e., their limitations in extracting and organizing the discriminative information from the data [12].

Feature learning aims to build spaces of representations either from distance constraints or by decomposing samples from complete space into smaller instances. A popular approach nowadays to learn meaningful representations from data is by using Deep Metric Learning (DML) [33] which harmoniously combines deep neural networks with distance metric learning. DML employs neural networks to create dense feature vectors and then uses a distance metric to compute similarities within these vectors [86].

For learning features from data, models based on neural network architecture are preferred over other approaches because of their computational competency and capabilities with other model architectures [17]. Creating numerical representations for product data is heavily discussed these days as they serve as a building block for machine learning algorithms in e-commerce. They can be trained by renowned techniques built using the neural network which results in dense numerical representations of products in real-valued vectors [83].

1.2 Representation Learning

Real-world datasets suffers from noise or corruption, which effects the output of a machine learning model [50]. To combat this problem, robust representations that minimizes noise can be generated from given data [42]. Due to this, we theorize that the performance of machine learning algorithms depends on different representations as they could capture more or less different features and variations in the data. Useful and problem-specific representations can be generated by having specific domain knowledge. Learning these representations of data is, as the name implies, called representation learning. This approach makes it easier to extract useful information when building classifiers or other predictors [12].

Conventional machine learning algorithm mostly focuses on solving problems like classification or regression with manually engineered features from raw data [60]. On the other hand, representation learning emphasizes the challenge of obtaining representations for inputs as a first step, such that these representations could be used to easily solve prior tasks [12]. There are different ways of learning representations, and here we will discuss algorithms based on deep learning methods.

Deep learning methods generally work on the idea of finding parameters that yield the best result for a given problem. This exercise is done by an optimization process, where the algorithm compares the input to the true label and updates its parameters based on the mismatch [9]. On the contrary, in representation learning, these algorithms focus on representations learned by the layers of the neural network. With the goal of learning representations without having any true labels, the algorithm focuses less on the error minimization [3].

Typically, deep learning algorithms consist of multiple non-linear transformations for learning representations of data. However, a representation learning algorithm traditionally consists of a shallow model that focuses on learning transformations such that essential features within the data are preserved [3]. Deep neural network based representation learning has gained popularity in natural language processing (NLP), image processing, and computer vision applications [60]. Representation learning has become a highly sought-after approach in the machine learning community, with regular workshops conducted at most of the leading conferences [12].

Learning useful representations from untagged data is still a challenging task, but improvements made over pre-existing algorithms could result in wide-reaching benefits. Recently, there has been continuous development in un/self-supervised representation learning, with significant improvement in results on text processing and visual tasks. When developing an unsupervised learning method, the objective is to develop a generator that represents useful information about the data making it possible to solve the problem at hand. These are usually made possible by exploiting prior knowledge about

structures in the data, rather than from the predefined labels [8].

1.3 Embeddings

An embedding can be defined as the process of transforming one thing into another. In mathematics, embedding refers to a transformation function f that maps objects from space $\mathbb{A}$ to an euclidean space $\mathbb{R}^n$ given as $f : \mathbb{A} \rightarrow \mathbb{R}^n$. The mapping from this embedding function f is both injective and structure-preserving. Here, injective means different elements from space $\mathbb{A}$ are always mapped to different elements of $\mathbb{R}^n$ or for every $a_1, a_2 \in \mathbb{A}$, $f(a_1) \neq f(a_2)$. Structure-preserving, on the other hand, means if some property holds for $a_1, a_2, .., a_m$ where m represents the number of elements in A, the same property holds for $f(a_1), f(a_2), ..., f(a_m)$. For example, if embeddings preserve distance given by d between elements in $\mathbb{A}$ and $\mathbb{R}^n$, then for every $a_1, a_2 \in \mathbb{A}$, $d(a_1, a_2) \approx d(f(a_1), f(a_2))$.

Working with real-world non-numeric data like text, images, or graphs is challenging as machine learning expects its input to be a numerical value. Generally, data mining algorithms rely on computing the distances between the elements in a euclidean space, which forces us to map the original, raw data into such space [64]. Embeddings are an important development in the field of machine learning and are formally defined as a dense numerical representation vector, i.e., of real-valued that encodes information from the original real-world data [76].

Embeddings make it possible to mathematically process real-world inputs like text and images. For example, in the context of machine learning embeddings generated by an artificial neural network map objects such as images, text, audio, etc., from an object space U into an n-dimensional euclidean vector space $\mathbb{R}^n$. These embeddings are produced in a way such that $f(U)$ is similarity preserving and has the same structure as U. Similarity preserving means the elements similar to each other in U are similar in euclidean space as well. For illustration, we suppose v_i as embeddings for $u_i \in U$, then some similarity measure S between embeddings $S(v_i, v_j)$ reflect the similarity between objects u_i and u_j [73].

Embeddings or neural embeddings offer a unique and impressive solution in machine learning to solve a variety of problems. It is a dense representation of high-dimensional data that ideally preserves essential information in the data while changing dimensionality. An embedding is always learned and can also be reused across models. Performing machine learning tasks on large sparse input vectors can be made easier with embeddings. They can encode versatile relationships between those structures and have become a core ingredient in modern machine learning [40]. Embedding provides task-specific dictionaries that encode variables into machine learning model-specific input. Embeddings have interpenetrated the data scientist's toolkit and significantly changed

how computer vision and NLP work

Other than deep neural networks, there are simpler methods for generating embeddings as well, and the methodology to be used depends on the problem. One of the popular approaches is having simple encoding per category, but this would only work well for a small number of categories (for example, cities or industry). A disadvantage of using embeddings is that it reduces the model's interpretability. An ideal embedding would try to capture most of the input's semantics by clustering semantically similar inputs in the embedding space. In this thesis, we will create embeddings for both categorical and descriptive text fields.

1.4 Related Work

Various text encoding techniques that create dense representations for words and documents have gained popularity in the NLP context over the past few years. Some of these techniques have been used to create embeddings for data generated from e-commerce websites. One is based on using Bidirectional Encoder Representations from Transformers BERT model [22], which works by pre-training deep bi-directional representations from untagged data. ProBERT [85] is a product classification model that uses BERT to predict the category of a product based on product titles and descriptions.

Meta-Prod2Vec [77] is another product embedding algorithm that is used to recommend similar items to a customer with applications like web search, ad targeting etc. It is based on the word2Vec [76] family of model architectures which is one of the prominent encoding techniques used in NLP tasks. Word2vec is based on the distributional hypothesis [26], which states that words similar would be closer in the real-valued space.

Some extensions of word2vec, like The Global Vectors for Word Representation or GloVe algorithm [58], are also used in product embedding tasks. They work by creating a word co-occurrence matrix that calculates statistics across the entire corpus, thereby adding global statistics to the task to generate dense vectors. This model ignores the morphology of these words by having a numerical vector for each word. To solve this, FastText (developed by Facebook) gives a vector representation associated with each character; words being represented as the sum of these representations [15]. This method allows training models on a large corpus quickly and can compute representations for rare words.

Another study used Autoencoder-based learning for disciplining representations in unsupervised and semi-supervised settings. Autoencoders are a kind of neural network used for encoding unlabeled data or dimensionality reduction. Autoencoders are designed in a way such that hidden layers create a bottleneck with a lesser number of neurons than the input layer, which results in the compressed representation of the in-

put. Target-embedding autoencoders can be used in prediction for supervised data, learning transitional latent representations mutually optimized to be both predictable from features as well as predictive of targets-encoding [36].

1.5 Our Goals

In this master's thesis, we propose a method to learn dense representations of text from eCommerce product data that establishes a mapping between heterogeneous product data and embedding space. These embeddings preserve the relationships between the products in the embedding space. Feature learning problems in the e-commerce setting often rely on domain-specific properties having a specific goal in mind. Our goal is to design a domain-independent and generalized algorithm that efficiently learns dense vectors of words for product queries.

We use a neural network model of architecture for creating representations, which is a word2vec [76] model and an extension of the word2vec model doc2Vec [41], used in combination with different attributes. The performance measure is assessed by the alignment between the product's similarity and its representations. This thesis also includes exploratory experiments that support our insights for product embeddings.

Products in the latent space are represented by using vectorized article data which includes descriptive and categorical attributes. The textual information in these articles consists of names, descriptions, IDs, manufacturers, etc. However, this algorithm can be easily extended to include other textual information. Due to the heterogeneity of our data, embeddings can get challenging as categorical attributes accept too many values. A primary challenge for our model is that these numerical feature vectors should effectively capture structural, syntactic, and semantic information.

1.6 Structure of Thesis

This section summarizes the whole structure of the thesis and it is organized as follows:

In chapter 2, we define and explain in detail the preliminary concepts used throughout this thesis.

In Chapter 3, we describe the theory of text embeddings, their applications, and different algorithms for creating word and document embeddings.

In chapter 4, we presented our approach for embedding heterogeneous product data for an unsupervised problem.

In Chapter 5, we give insights on our data set, discuss important data pre-processing steps, and evaluate the embedding technique described in the previous chapter on our dataset. Furthermore, we also provide the implementation details related to our model.

In chapter 6, we discuss the conclusion and future work that could further enhance our model.

2 Preliminaries

In this section, we introduce definitions and preliminary concepts that are considered in the model proposed within this thesis.

2.1 Similarity Measures

In the context of data science, the term similarity measure refers to calculating how data points in space are similar to each other. The similarity measure between dimensions representing features of objects is a numerical value that describes how close or distant objects are from each other. In text analysis, both distances and similarities are used interchangeably, whereas distances are dissimilarity measures. Similarities can be derived from distances by applying a monotonically decreasing function.

In this thesis, the similarity value which is also commonly referred to as the similarity score is normalized in the range of 0 and 1. We bound these similarities into a positive space where the highest similarity between two elements could be 1, representing the two elements are exactly similar. Higher similarity scores represent high similarity between two data points while low values represent the two data points are dissimilar. This helps to keep the interpretation of similarity scores generated by different similarity measures consistent. A similarity measure S given a set of objects P with $S : P \times P \rightarrow [0,1]$ where $[0,1] \in \mathbb{R}$ satisfies the following properties:

- **Non-negativity:** $S(p,q) \geq 0 \quad \forall p,q \in P$
- **Maximum dominance principle:** $S(p,p) \geq S(q,p)$ and $S(p,p) \geq S(p,q) \quad \forall p,q \in P$

We will discuss a few similarity measures here which are used in this thesis to calculate similarities between the embeddings.

2.1.1 Euclidean similarity

The euclidean similarity is calculated by modifying euclidean distance which is one of the most commonly used distance metrics. Euclidean distance computes the distance between data points by measuring the length of the segment connecting two data points. Euclidean distance is also known as the L2-norm of a vector. Mathematically, it is calculated as the square root of the sum of the squared distance between two vectors. For two data vectors $X = (x_1, x_2, .., x_n)$ and $Y = (y_1, y_2, ...y_n)$ of same length $n \in N$, euclidean distance is calculated by the formula given in equation 2.1:

$$d_{eu}(X,Y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{2.1}$$

Distance values are higher when data points are far apart from each other and lower when data points are closer to each other. Whereas, similarity values are higher for similar elements and vice versa. The relation between similarity and distance is that one decreases when the other increases. To convert the euclidean distance into a similarity measure, we use the formula given in equation 2.2 which takes the reciprocal of distance. In denominator, 1 is added to bound the value of highest similarity at 1.

$$S_{eu} = \frac{1}{1 + d_{eu}} \tag{2.2}$$

2.1.2 Cosine Similarity

Cosine similarity is a measure that computes the similarity between data vectors based on the differences between their angles. In mathematical terms, this similarity calculates the cosine of the angle between two vectors in the euclidean space. It helps calculate the similarity between vectors of different sizes as vectors considered far apart by euclidean distance due to their sizes can have a smaller angle between their orientations. Cosine similarity is higher for smaller angles between data vectors and smaller for higher angles. Cosine similarity between two data vectors $X = (x_1, x_2, .., x_n)$ and $Y = (y_1, y_2, ...y_n)$ of same length $n \in N$ is given as:

$$Cos(\theta) = \frac{\sum_1^n x_i y_i}{\sqrt{\sum_1^n x_i^2}\sqrt{\sum_1^n y_i^2}} \tag{2.3}$$

2.2 Clustering

Clustering is an important technique to get meaningful insights from unsupervised data based on the notion of distance or dissimilarity. In a given set of data points, a clustering algorithm would assign them into different groups such that data points within a single group have similar features and data points in different groups have highly dissimilar properties. In other words, it tries to collect or segregate similar bunch of elements and assign them into clusters. Here, we will discuss two common but different types of clustering algorithms that are used in this thesis to cluster embeddings.

2.2.1 K-means Clustering

K-means [44] is a popular and simple clustering algorithm used for clustering data points in euclidean space, as it averages data points. Therefore, this averaging in k-means requires performing vector operations and we always don't have the exact vector representation for data points in non-euclidean spaces [75]. This algorithm uses euclidean distance as the measure to compute dissimilarity between data points [18]. K-means belongs to the family of partition-based clustering algorithms that splits the dataset into disjoint partitions or clusters [37]. This algorithm divides the dataset A with m data points $(a_1, a_2, ..., a_m)$ into k clusters $B = (b_1, b_2, .., b_k)$, where k is given as input and $k \leq n$.

In the first step, the algorithm selects k data points randomly as centroids also called prototypes. There are other methods suggested for selecting initial centroids and one of them is described in [29], whereas the standard k-means given in [44] selects initial centroids randomly. K-means is an iterative algorithm and the step-by-step working of the k-means algorithm is given as follows:

1. As we mentioned above, given a set of data points $A = \{a_1, a_2, ..., a_m\}$ belonging to a m dimensional space $\mathbb{R}^n$, and given number of clusters k, it randomly selects k initial centroids as $w_1, w_2, ..., w_k$.
2. Based on initial centroids, it assigns each data point a_i to its closest prototype w_l i.e, with the smallest euclidean distance (2.1) between prototype w_l and a_i. Each data point belongs to one cluster only and the data set is partitioned in k clusters given as $B = (b_1, b_2, ..., b_k)$.
3. Centroids or prototypes in each iteration t are re-computed from each cluster by taking the mean of all data points in a cluster given as:

$$w_l^{(t+1)} = \frac{1}{|B_l^{(t)}|} \sum_{a_i \in B_l^{(t)}} a_i \tag{2.4}$$

4. Keep repeating steps 2 and 3 until assignments of data points do not change or for a given number of iterations (if pre-defined).

Following these steps results in the data points being partitioned into a given number of clusters k. The cost function for k-means algorithm is given in equation 2.5:

$$\mathscr{L} = \sum_{i=1}^{n} \sum_{l=1}^{k} u_{il} ||a_i - w_l||^2 \tag{2.5}$$

where, $u_{il} = 1$ if x_i belongs to cluster l and 0 otherwise, w_l as given in 2.4 is the mean of

data points in each cluster b_l for $l = (1,..,k)$. For all data points lying close to the cluster centers l, the value of u_{il} is 1 and the cost function is then optimized only for computing the cluster centroids w_l. As we emphasized on creating embeddings in this thesis, we implemented k-means clustering which is the simplest algorithm used for clustering data in euclidean space. K-means algorithm also intelligently adpats to new observations.

The k-means algorithm has a few shortcomings as it requires the k to be chosen manually and given as an input. The initialization of clusters affects overall clustering results as different initializations do not produce the same results and are also sensitive to outliers. This method is an NP-hard problem, which means it is computationally difficult for k-means to find the optimal clustering in polynomial time [25].

2.2.2 Balanced Iterative Reducing and Clustering using Hierarchies

Clustering algorithms like K-means clustering is an iterative algorithm that runs multiple iterations over a dataset to group the data points into clusters. This makes it difficult for k-means to scale with regard to processing time and quality as the size of dataset increases [29]. Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) [87] is a scalable clustering algorithm based on hierarchical clustering which creates a tree of clusters organized in a dendrogram. It only scans the dataset once making it a better choice for performing clustering on large datasets.

BIRCH clusters large datasets by first creating smaller summaries preserving as much information as possible and then these summaries get clustered instead of clustering a large amount of data. It transforms data into a tree-like structure where leaves from this tree represents centroids. BIRCH is based on two important concepts Clustering Feature (CF) and CF - Tree Clustering Feature.

It summarizes large datasets into smaller subgroups called Clustering Feature (CF) entries which is a three-dimensional vector representing the information contained within those subgroups. The CF for cluster points a_i is represented as:

$$CF = (n, LS, SS) \tag{2.6}$$

where n is the number of datapoints in the cluster, LS is the linear sum of n points $\sum_{i=1}^{n} a_i$ and SS is the squared sum of all data points within the cluster $\sum_{i=1}^{n} {a_i}^2$. It is possible for CF entries to overlap with other entries.

A CF tree is a height-balanced tree that saves sub-clusters within its leaf nodes. Every non-leaf node in a tree has descendants and these non-leaf nodes are made up of

entries of CF of their children. Therefore, these non-leaf node summarizes the clustering information about their descendants.

The size of a CF tree is defined by two parameters, which are threshold, T., and branching factor, B. The maximum number of entries a CF tree can have within each of its leaf nodes is called a threshold. The branching factor defines the maximum number of sub-clusters or children per non-leaf node.

2.3 Visualization Techniques

Embeddings generated from neural networks project real-world data into an n-dimensional euclidean space where n could be in $100s$. While it is easier to explore and visualize embeddings in two or three-dimensional spaces by presenting them in a scatter plot, it gets harder or more complicated to visualize for higher dimensional spaces. Dimensionality reduction techniques are used to map higher dimensional vectors to lower-dimensional vectors while preserving as much meaningful information as possible, so these vectors with reduced dimensions could be visualized more easily. To visualize the embeddings created from our approach in a 2-dimensional space, we will discuss a popular visualization technique that uses the concept of dimensionality reduction to effectively visualize higher dimensional data:

t-Distributed Stochastic Neighbor Embedding

t-Distributed Stochastic Neighbor Embedding (t-sne) [45] is a widely used technique that applies a non-linear transformation to map higher dimensional data vectors into lower-dimension vectors while preserving structures and neighborhoods from higher dimensional space [14]. t-sne follows a procedure in which a dimension agnostic probability distribution is constructed over the high-dimensional vectors. It finds a lower-dimensional approximation, such that vectors closer to each other in the high-dimensional space are more like to be selected than the vectors distant from each other. In mathematical form, given a higher dimensional dataset $X = x_{i_{i=1}}^{m}$ where $x_i \in \mathbb{R}^n$, t-sne finds a lower dimensional representation for it given as $Y = y_{i_{i=1}}^{m}$ where $y_i \in \mathbb{R}^s$ and $s << n$.

At first, this algorithm finds the points near each other in the higher dimensional space by calculating the euclidean distances between all pairs. Then these distances are transformed into conditional probabilities which represent distances between every pair of points. As it captures the underlying structures, the probability distribution for each datapoint x_j being closer to other points x_i is given by Gaussian:

$$p_{i|j} := \frac{exp(-||x_i - x_j||^2/2\sigma^2)}{\sum_{k \neq j} exp(-|x_k - x_j|^2/2\sigma^2)} \tag{2.7}$$

The above equation is divided by the sum of all other points close to the gaussian center x_i to deal with the possibility of having clusters with different densities. Here σ_j (variance) is a parameter given as an input to the model and the value for it is specified by the expected number of neighbors, called as perplexity. High dimensional data is represented by probability distribution P where the probability is symmetrized ($p_{i|j} = p_{j|i}$ to make the optimization easier.

$$P = (p_{ij})_{i,j=1}^{m}, \quad p_{ij} = \frac{p_{i|j} + p_{j|i}}{2m} \tag{2.8}$$

In the second step, t-SNE creates another probability distribution over the low dimensional vectors and this distribution is represented by Q whereas the data points in low dimensional space are y_i.

$$Q = (q_{ij})_{i,j=1}^{m}, \quad q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq j}(1 + |y_k - y_l|^2)^{-1}} \tag{2.9}$$

In equation 2.9 it uses t-distribution instead of the gaussian distribution which was used for original high dimensional data. It is because the t-distribution avoids crowding of points in the lower dimensional space by increasing the distance between spaces when mapped from high dimensional space. The 't' in the name of t-sne is from this t-distribution used. The goal is to modify these vectors to minimize the variances between the two distributions. In other words, y_i are optimized such that P and Q are close to each other as possible and this is expressed by Kullback-Leibner divergence defined as:

$$KL(P|Q) = \sum_{i=1}^{m} \sum_{j=1}^{m} p_{ij} log(\frac{p_{ij}}{q_{ij}}) \tag{2.10}$$

As this is now an optimization problem which is defined as finding the best values of parameters that minimizes the error by training the model iteratively. It finds optimum parameters by using the gradient descent algorithm which finds the minimum or maximum from a given cost function by calculating the derivative of this function at every point.

$$\frac{\partial KL(P|Q)}{\partial y_i} = 4 \sum_{j=1}^{m} (p_{ij} - q_{ij})(y_i - y_j)(1 + ||y_i - y_j||^2)^{-1} \tag{2.11}$$

By following the optimization process above, low dimensional vector space is aligned with the high-dimensional vector space.

2.4 Cluster Metrics

Assessing the outcome of a clustering algorithm is a very important segment of clustering data and, therefore, numerous evaluation metrics are suggested to evaluate the quality of clustering algorithms [6,55]. These evaluation metrics are also categorized as internal and external, where internal metrics evaluate the quality of clusters only from the given unlabeled data. Whereas, external metrics require ground truth labels to compare the results of clustering algorithms [65]. The evaluation from these external cluster metrics is not as simple as counting the number of errors; rather, it compares the structure and separation of data between the predicted clusters and the given (true) set of clusters.

In this thesis, we discussed a few external metrics belonging to the family of extrinsic measures that can quantify the quality of clusters generated by different methods on our embeddings. The explanations for these metrics are given as follows:

2.4.1 Adjusted Rand Index

Rand Index (RI) [61] is based on the standard idea of comparing the result of the classification scheme to the correct classification. It is one of the most commonly used evaluation metrics for evaluating cluster quality [78]. It computes the similarity between two clusters by counting all unordered pairs of samples which have been assigned to the same or different clusters in class or true labels and predicted clusters. The formula of the Rand Index is:

$$RI = \frac{\text{Count of Pairs in Agreement}}{\text{Total Number of Pairs}} \tag{2.12}$$

The RI value ranges from 0 to 1 with 1 being a perfect match. It suffers from the drawback of yielding a high value for pair of partitions of given samples when the number of clusters is high. To overcome this problem, the Adjusted Rank Index (ARI) is introduced which adjusts the raw RI score by introducing the concept of 'adjusted for a chance' in the RI scheme.

$$ARI = \frac{\text{RI - Expected RI}}{\text{Max(RI) - Expected RI}} \tag{2.13}$$

2.4.2 Fowlkes-Mallows scores

Fowlkes-Mallows scores or Fowlkes-Mallows Index (FMI) [52] is another evaluation metric used to compare the clustering results from two different algorithms. The score given by FMI is based on the similarity between two clustering outcomes, where it considers one as ground truth and the other as predicted clusters. It gives a value between 0 and 1, where 1 means complete similarity between two clustering results and 0 indicates no similarity. The formula for FMI between two clustering results F_1 (ground truth) and F_2 (predicted clusters) is given in equation 2.14:

$$FMI = \frac{TP}{\sqrt{(TP+FN)(TP+FP)}} \tag{2.14}$$

here, TP means true positive which shows the number of data points that belongs to the same clusters in both F_1 and F_2. FP refers to false positive which counts the number of data points belonging to the same cluster in F_1 and different in F_2. FN refers to false negatives and is the count of data points that are in the same cluster in F_2 and different clusters in ground truth F_1. The advantages of using FMI is that it doesn't make any assumptions about cluster structures and can compare clusters with different structures. FMI is a preferable evaluation metric for comparing two unrelated datasets.

2.4.3 V-measure

V-measure [68] is another external cluster metric that quantifies the quality of clustering results based on conditional entropy analysis [28]. V-measure is computed as the combination of homogeneity and completeness score which is similar to how precision and recall are combined in a popular classification metric F-measure [74].

Given a dataset with N data points which has ground truth cluster labels given as $A = \{a_i | i = 1, ..., p\}$ and our resulting set of clusters $B = \{b_i | i = 1, ..., m\}$. Let C be a matrix that represents the clustering solution on given dataset such that $C = \{c_{ij}\}$, where c_{ij} represents data points that are member of ground truth cluster a_i and an element of cluster b_j. The homogeneity and completeness are defined as:

Homogeneity score

A homogeneity score is defined as a measure that calculates how many of the elements within the cluster are similar or belong to the same ground truth cluster. The formula for homogeneity score is given in equation 2.15, which is calculated as the conditional entropy of ground truth clusters given our clustering outcomes.

$$h = 1 - \frac{H(A|B)}{H(A)} \tag{2.15}$$

To satisfy homogeneity criteria, the clustering algorithm must assign all the data points from a ground truth cluster to a single cluster. In this perfect case, the value of $H(A|B)$ is equal to 0. These conditional entropies from the homogeneity score formula are given as:

$$H(A|B) = -\sum_{b=1}^{|B|}\sum_{a=1}^{|A|}\frac{c_{ab}}{N}log(\frac{c_{ab}}{\sum_{a=1}^{|A|}c_{ab}}) \tag{2.16}$$

$$H(A) = -\sum_{a=1}^{|A|}\frac{\sum_{b=1}^{|B|}c_{ab}}{p}log(\frac{\sum_{b=1}^{|B|}c_{ab}}{p}) \tag{2.17}$$

Completeness score

A completeness score is defined as a measure that calculates how many of the elements in each true cluster are also elements of a single predicted cluster. Completeness is considered as symmetrical to homogeneity [68]. The formula for homogeneity score is given in equation 2.15, which is calculated as the conditional entropy of predicted clusters given ground truth clusters.

$$c = 1 - \frac{H(B|A)}{H(B)} \tag{2.18}$$

To satisfy completeness criteria, the clustering algorithm must assign all the data points from a ground truth cluster to a single cluster. In this perfect case, the value of $H(B|A)$ is equal to 0.

$$H(B|A) = -\sum_{a=1}^{|A|}\sum_{b=1}^{|B|}\frac{c_{ab}}{N}log(\frac{c_{ab}}{\sum_{b=1}^{|B|}c_{ab}}) \tag{2.19}$$

$$H(B) = -\sum_{b=1}^{|B|} \frac{\sum_{a=1}^{|A|} c_{ab}}{p} log(\frac{\sum_{a=1}^{|A|} c_{ab}}{p}) \tag{2.20}$$

Using these homogeneity and completeness scores the formula for V-measure is given as:

$$V - measure = \frac{(1+\beta)(Homogeneity * completeness)}{(\beta * Homogeneity * completeness} \tag{2.21}$$

The default value for β is 1 and we used this value of β in our thesis to evaluate the clustering results on embeddings. For the default value of β, v-measure is equivalent to another cluster evaluation metric normalised mutual information [32]. β value other than 1 is used when we assign different weights to homogeneity and completeness.

3 Text Embeddings

Text data carries more information and has more chances of exploring valuable insights as compared to quantitative data. To make use of this information, NLP algorithms aim to gain a human-like understanding of the text. This helps to inspect the vast amount of text and gather relevant insights from it. These algorithms generate text embeddings, which are a type of embeddings that represents textual data in a euclidean space of n dimensions. These embeddings from the euclidean space can then be used to formulate numerous models for the classification of structured and unstructured textual data [48].

Formally, text embedding is defined as representing words as dense numerical vectors that capture the actual and semantic meaning of the words [47]. These are calculated by capturing the relatedness between individual words in a corpus and preserving them in the embedding space [31]. That is, words that appear in a similar context or the words that appear frequently together tend to have similar meanings and therefore they should have similar representations.

For example, text embedding help represent the words in a sentence such as Profession or title by a dense vector that is comparatively smaller. Each cell within this dense vector represents a different characteristic of the word and contains a value between 0 and 1. These values are then used to compare the syntactic and semantic similarities between different words in the input sentence. The embeddings of a given profession or title are closer to the embedding of a similar one in the embedding space and these similarities are calculated by distances or angles.

Recent developments in language modeling which uses neural networks have made it possible to model language as distributions over words. By learning to predict the next word based on previous words, these models have been shown to automatically internalize linguistic concepts such as sentences, subclauses, and even sentiment [4].

Previously, word embeddings were trained on use cases with defined training objectives and model architecture. These embeddings were used in applications like language translations or recommender systems [51]. In this thesis, we generate embeddings in an unsupervised manner intending to preserve relationships and similarities from input text. We provide a theoretical understanding of text (word and document) embeddings along with their dimensionality and also our model achieves state-of-the-art results.

There are multiple ways of projecting words into a multidimensional vector space. They all share the common goal of capturing as much semantic and contextual information in the text as possible. There are several different methods of formulating embedding for words from simple one-hot encoding to calculating word count or co-occurrence

matrix to using deep neural network architectures. A few of the well-known methods are discussed in the following subsections:

3.1 One-Hot Encoding

One-Hot Encoding [21] is one of the simplest concepts for vector representation in Natural Language Processing. It is useful for representing categorical variables as numerical vectors. Each cell is represented as a binary vector that has only the index of the integer marked as one and all zeros. It is easier to compute and fast but does not capture the real semantic value of a word in a sentence. An example of one-hot encoding a sentence is given in 3.1:

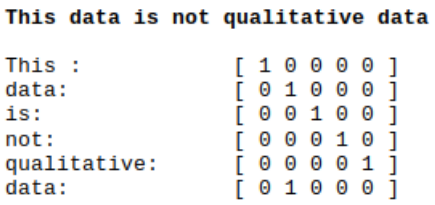

Figure 3.1: Example to represent the one-hot encoding of a sentence

Here this sentence has a vocabulary size of 5 with one word "data" being repeated. However, in practical applications of word embeddings, the vocabulary from text or documents is rather huge. Each sentence within these documents is of variable length and each word has a different frequency in the corpus. Consequently, if we represent such data by applying one-hot encoding, then the resulting embeddings are sparse and would have an unmanageable number of dimensions. This phenomenon is also known as the Curse of Dimensionality [11]. As simplicity comes with drawbacks, one-hot encoding does not capture semantic information and only tends to work well with categorical data.

3.2 Term Frequency - Inverse Document Frequency

Term Frequency - Inverse Document Frequency (TF-IDF) [70] is a count based technique of embedding that determines the importance of individual terms in a sentence or document. Whereas, a document could have multiple sentences in it. The encoding process is similar to One-Hot Encoding. Alternatively, it is a statistical measure that assigns a count value to each term instead of a 1. In our explanations below, the target term which could be a word or multiple words like "New York" is referred as t, a single document as d, and the set of all documents as D. The TF-IDF value is a multiplication

of Term frequency and Inverse document frequency values which are described in the next part:

3.2.1 Term Frequency

Term frequency (TF) [70] count the occurence of a word or a particular term in a document out of all words. It is calculated as the ratio of the number of particular term or target term in the document to the total number of terms in the document. Count of target terms in the given document d is represented as $f_{t,d}$ in equation 3.1.

$$tf(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \tag{3.1}$$

3.2.2 Inverse Document Frequency

Inverse document frequency (IDF) [70] compares the occurrence of target words among other words in a document. IDF value is calculated between the total number of documents and the total number of document in which the target term appears. At this point, the algorithm is not effected by the number of times the target term appears in the document.

$$idf(t,D) = log(\frac{|D|}{count(d \in D; t \in d)}) \tag{3.2}$$

This shows that the key idea for TF-IDF is the importance of a term is inversely related to the term's frequency across all documents. By multiplying the results from $tf(i,d)$ and $idf(t,D)$ above, we can get our final TF-IDF value.

$$tfidf(t,d,D) = tf(t,d) \cdot idf(t,D) \tag{3.3}$$

TF-IDF score for a term shows its relevancy, as more important terms will have a higher score and a score approaching 0 shows the term is less relevant. TF-IDF has the advantage of being simpler to compute and computationally cheap. But, it also suffers from the curse of dimensionality as TF-IDF vector sizes can get huge. Also, this method doesn't capture the semantic meaning of the words. It does consider the importance of words by their occurrence, but still cannot understand the context of it.

3.3 Word2Vec

Word2Vec [46] is a neural network based word embedding technique developed at Google, which is one of the integral techniques for solving many NLP problems. It is a shallow neural network that uses a single hidden layer to create embeddings of words. It takes a large corpus of words as input and produces an embedding space, comprising several hundred dimensions, such that each unique word in the corpus is assigned a corresponding vector in the space. The size or dimension of these embedding vectors is defined as part of the model. Input words are mapped or positioned in the embedding space such that syntactically and semantically similar words in the corpus are located in close proximity to one another in the embedding space.

The architecture of word2vec is similar to an autoencoder's, as it takes a large input vector, and encodes it to smaller dense vectors using a hidden layer which is a standard fully connected (Dense) layer. Then instead of decoding it back to the original input vector, it gives output probabilities of target words. The weights of the hidden layer are used as a proxy presentation for input words. These weights are calculated by taking the product of input words. As we cannot feed a word directly as a string into a neural network, alternatively, we convert string words into one-hot vectors, which we have already described above.

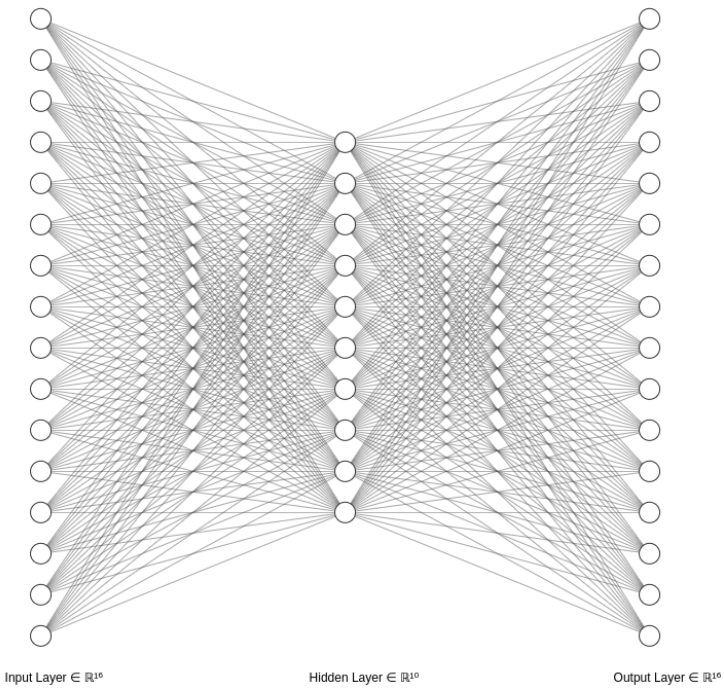

Figure 3.2: Word2vec model architecture with inputs as one-hot vectors of given words

The architecture of a simple word2vec model is given in Figure 3.2 above, where R^{16} in

the input and output layer means it has 16 neurons (dimension), while R^{10} shows that the hidden layer has a dimension of 10. The output layer in this architecture is also a dense layer with a softmax activation, where this activation function transforms the real number vectors from the output layer into probabilistic vectors [53]. It yields a probability vector of the same embedding size defined in the model for each word in the input. This probability indicates the similarity between the target word and other words in the corpus. These vectors can also be used to group similar words based on their distances which is also the key effectiveness of word2vec.

As Word2Vec is based on the idea of the Distributional Hypothesis [69]: the word embeddings generated by it are estimated based on their occurrences in the text. Words like “man” and “woman” would have very similar embedding to each other. Arithmetic operations can be performed on these embedding vectors which gives close approximates of the resulting embedding. For example, embedding vector of "man" subtracted by embedding vector of "woman" and added by "boy" would yield an embedding very close to the vector of "girl".

Word2Vec is a particularly computationally-efficient predictive model for learning word embeddings from raw text [76]. Word2Vec itself is not a deep neural network, but the embeddings created by Word2Vec can be understood by deep neural networks. Word2vec representation is created using two different variants: the Continuous Bag-of-Words model (CBOW) and the Skip-Gram model.

3.3.1 Continuous Bag-of-Words Model

The first proposed architecture is a feedforward neural network where the projection layer is shared among all words. This architecture is called a bag-of-words model as it does not maintain the history of the order of words for projection. Moreover, it not only uses words before the target word but also words after the target word; this helps obtain the best performance on the task of predicting the current (middle) word by building a log-linear classifier with F future and F history words [46].

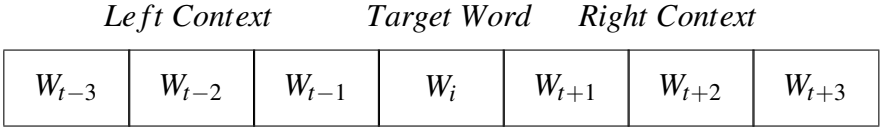

Figure 3.3: CBOW sequence of words

Here F is defined as the half size of the context window. The objective of the Continuous Bag-of-Words (CBOW) model [76] is given a list of context words (surrounding words) in a sentence, the network will predict the probability for each word in the vocabulary to predict the target word ŷ. The idea behind this model is given as considering the phrase "This is my thesis", we will choose our target word to be "is" and our context words to be ["This", "my", "thesis"]. This model will take the embeddings of the context words to try and predict the target word.

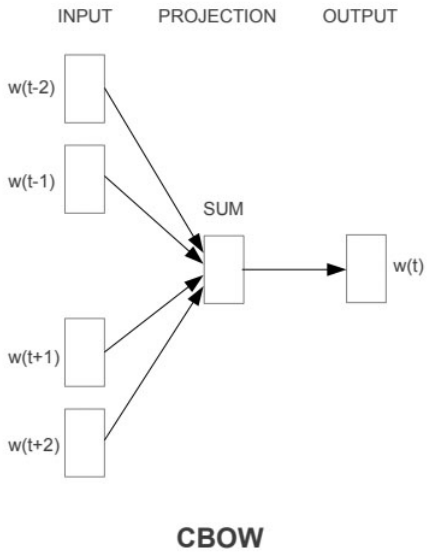

Figure 3.4: It shows the CBOW architecture that uses context words to predict the current or target word. This figure is taken from [46]

In the CBOW neural network architecture given in figure 3.3.1, the hidden layer has a linear activation function and the activation function for the output layer is softmax. Inputs are one-hot encoded vectors of words from the corpus of dimension M, where M is the size of the vocabulary. The hidden layer generates vectors of dimension N which is the embedding size of the model. The values of the hidden layer given an input data x having C context words is:

$$h = \frac{1}{C} W^T \sum_{c=1}^{C} x_c \tag{3.4}$$

The output layer also generates a vector of dimension M. The weights between the input and the hidden layer are given by matrix W which has dimensions M x N. Similarly, the weights between the hidden layer and the output layer are represented by $W^{'}$ and are of dimension N x M. The value of the output layer is given as u in equation 3.5.

$$u = W'^T h$$
$$u = \frac{1}{C} W'^T W^T \sum_{c=1}^{C} x_c \tag{3.5}$$

u is defined as the value of the output layer before applying the softmax activation function. After applying the softmax activation function on the value above in the output layer we get the probabilistic vector y.

$$y = Softmax(u)$$
$$y = Softmax(\frac{1}{C} W'^T W^T \sum_{c=1}^{C} x_c) \tag{3.6}$$

Like all other neural networks, it has weights and during the training, its goal is to adjust those weights to optimize a loss function. However, in the case of word2vec, these weights, in the end, are used as the word embedding vectors. As we want to compare two probabilistic vectors we will use a popular information theory measure i.e, cross-entropy to compare the distances between two vectors. Cross-entropy in this case is derived from the formulation of the loss function and is given in 3.7:

$$H(\hat{y}, y) = -\sum_{j-1}^{|M|} y_j log(\hat{y}_j) \tag{3.7}$$

In our specific case, the given target word y is a one-hot vector, which has value 1 at index j^* (and value 0 at all other positions). Then the cross-entropy loss function defined in equation 3.7 simplifies to:

$$H(\hat{y}, y) = -y_{j^*} log(\hat{y}_{j^*}) \tag{3.8}$$

As we train our model against the context-target word windows which is the conditional probability of predicting the target word w_0 given C context words given as $P(w_0|w_{c,1}, w_{c,2}, ..., w_{c,C})$. The loss function to be optimized is defined in equation 3.9:

$$
\begin{aligned}
\mathscr{L} &= -log(\mathbb{P}(w_0|w_{c,1}, w_{c,2}, ..., w_{c,C})) \\
\mathscr{L} &= -log(y_{j^*}) \\
\mathscr{L} &= -log[Softmax(u_{j^*})] \\
\mathscr{L} &= -log(\frac{exp(u_{j^*})}{\sum_i exp(u_i)}) \\
\mathscr{L} &= -u_{j^*} + log\sum_i exp(u_i)
\end{aligned} \tag{3.9}
$$

Here u_{j^*} is defined as the value of the output layer before applying the softmax activation function, and C is the number of context words. To minimize this loss function the model learns the optimal values for weight matrix W and W'. In the neural network domain, this optimization is done using the concept of backpropagation and gradient descent.

3.3.2 Continuous Skip-Gram Model

The continuous skip-gram model [76] can be imagined as the opposite of the CBOW model. In this architecture, it takes the current word (which is the target word in CBOW) as an input to the log-linear classifier with the continuous projection layer. It then tries to accurately predict the words before and after this current word which were context words in CBOW. In simple terms, this model essentially tries to learn and predict the context words around the specified input word.

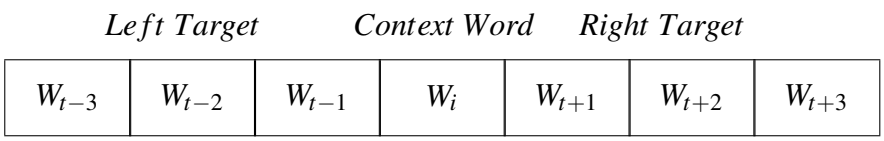

Figure 3.5: Skip-gram sequence of words

Based on experiments assessing the accuracy of this model it was found that increasing the context window range improves the prediction quality given a large range of word

vectors, however, it also increases the computational complexity. Since the more distant words are usually less related to the current word than those close to it, we give less weight to the distant words by sampling less from those words in our training examples [46].

In Skip-gram also F is defined as the half size of the context window. Given a context word, the skip-gram model will predict C target words. The idea behind this model is given as considering the phrase "This is my thesis", we will choose our context word to be "is" and our target words to be ["This", "my", "thesis"].

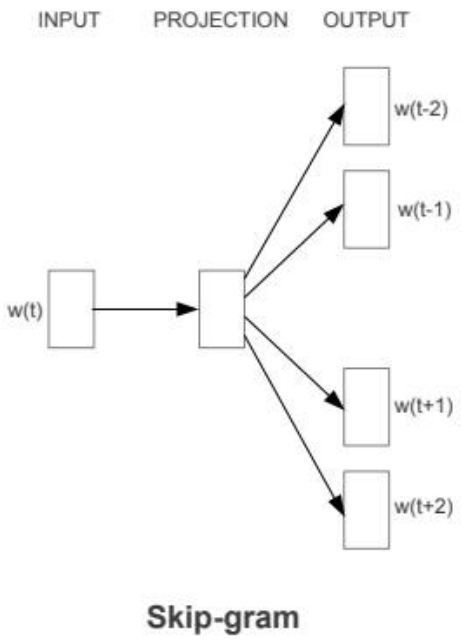

Figure 3.6: The Skip-gram architecture uses one target word to predict context words as shown in [46]

The architecture of skip-gram is given in figure 3.3.2 and is similar to the CBOW model with one hidden layer. As it is the opposite of CBOW, the architecture for it is also an inverse, where it takes one word as a one-hot encoded vector for input and the output layer gives probabilistic vectors for multiple target words. Because of similar architecture, the hidden layer values in skip-gram are calculated similarly as given in the CBOW model with it having one context word only.

$$h = W^T x \tag{3.10}$$

Output layer in skip-gram predicts C context words and the value for it is referred as u_c, and is given in equation 3.11:

$$u_c = W'^T W^T h \qquad c = 1,..,C \tag{3.11}$$

The values of the output layer after applying the softmax function are again probabilistic vectors y_c in this variant as well.

$$y_c = Softmax(W'^T W^T h) \qquad c = 1,..,C \tag{3.12}$$

The loss function of the Skip-gram model is also trained against the context-target word pairs where C is the number of target words to be predicted and M is the size of the vocabulary. In mathematical terms, it is defined as the conditional probability $\mathbb{P}(w_{c,1}, w_{c,2}, ..., w_{c,C}|w_0)$. The loss function to be optimized for skip-gram is given in the equation 3.13:

$$\begin{aligned}
\mathscr{L} &= -log(\mathbb{P}(w_{c,1}, w_{c,2}, ..., w_{c,C}|w_0)) \\
\mathscr{L} &= -log(\prod_{c=1}^{C} \mathbb{P}(w_{c,i}|w_0)) \\
\mathscr{L} &= -log(\prod_{c=1}^{C} \frac{exp(u_{c,j^*})}{\sum_{j=1}^{|M|} exp(u_{c,j})}) \\
\mathscr{L} &= -\sum_{c=1}^{C} u_{c,j^*} + \sum_{c=1}^{C} log \sum_{j=1}^{|M|} exp(u_{c,j})
\end{aligned} \tag{3.13}$$

Similar to CBOW, this loss function can also be minimized by using backpropagation and gradient descent.

3.4 Doc2Vec

Following the success of word2vec, the same group of researchers who developed word2vec [76] came up with this approach doc2vec [41], which extends the idea of creating vector representations of more than single words. Doc2Vec [41] creates numeric representations of a document where a document could have multiple sentences (list of words). To create the representation for documents, doc2vec introduces the concept of a paragraph vector.

As the name paragraph suggests, these documents do not have any defined length or logical structures like words and can be of any length. They propose a Paragraph Vector

that learns fixed-length feature representations for each given document from different inputs of variable lengths ranging from the phrase, or sentence to a paragraph, or a large document.

The architecture of doc2vec is similar to word2vec with an extra unique paragraph vector (or paragraph id) per document given as input. As the doc2vec model creates representations for documents, it requires a set of documents as a corpus. For each given word, a word vector Z is generated and for each document, a document id D is generated. This model trains weights for the hidden layer given as W and for the output layer given as W'. Word vectors are one-hot vectors with a dimension of $1\text{x}M$ where M is the size of the word vocabulary. The document Id vector has a dimension of $1\text{x}B$, where B is the number of total documents. The weight matrix from the hidden layer has dimensions $M\text{x}N$ for word vectors and $B\text{x}N$ for document vectors where N is the embedding dimension.

Analogous to word2vec, there are two ways to create document representations in the doc2vec model: Paragraph Vector Distributed Memory (PV-DM) and Paragraph Vector Distributed Bag-of-Words (PV-DBOW).

3.4.1 Paragraph Vector: A Distributed Memory Model

PV-DM [41] approach for learning paragraph vectors is similar to the CBOW approach in word2vec for learning word vectors. As in the CBOW context, multiple words are used to predict a target word which captures the semantic similarity between all the words in a sentence. We will make use of this approach in our paragraph vectors in a similar manner. Given many context words taken from the document, the paragraph vector can be used to predict the next words within the document.

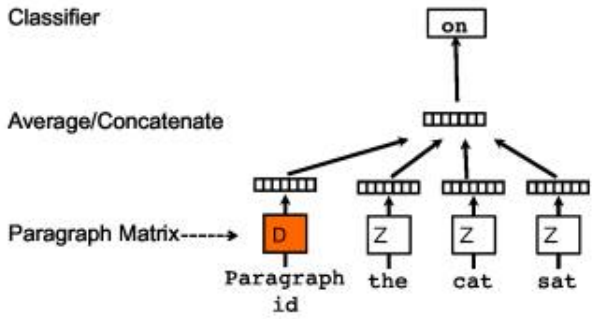

Figure 3.7: A framework for learning word vectors. The context of three words (“the,” “cat,” and “sat”) is used to predict the fourth word (“on”). This figure and explanation are given in [41].

In the figure 3.4.1, it feels familiar with the architecture of CBOW in fig 3.3.1 with the

extension of having paragraph id. Every word is mapped to a unique word vector represented by a column in matrix Z and a unique paragraph id is assigned for each paragraph or document represented by a column in matrix D. PV-DM model concatenates or averages the paragraph vector and word vectors in context. These concatenated/averaged vectors are then used by the classifier to predict the next word in a paragraph.

We define contexts as a fixed number of words to be sampled by a sliding window over the paragraph. All context windows from the same paragraph have access to the same paragraph vector but not across paragraphs. The word vector matrix Z, however, is shared across paragraphs. I.e., the vector for the word "king" is the same across all paragraphs. This paragraph id token can be thought of as another word. This id acts as a memory that remembers what is missing from the current context or sequence of words. For this reason, we often call this model the Distributed Memory Model of Paragraph Vectors (PV-DM).

As the architecture for doc2vec is also a neural network, values of hidden layer and output are similar as CBOW model given in equation 3.4, 3.5 and 3.6. The only change is in the hidden layer weights where D is also added. A sample of fixed length is drawn randomly as a context window from the paragraph and at each step of training that sample is used to update the weights of the model.

The total number of parameters to be learned within the model are given as N x p + M x q. Here, N is the embedding dimension, B is defined as the total number of paragraphs/documents in the corpus, and M is the complete word vocabulary across all paragraphs. p and q are the mapped dimension for paragraph vector and words respectively. The loss function of the PV-DM model is also trained against the context-target word pairs with the paragraph id D being part of the context.

$$\mathscr{L} = -log(\mathbb{P}(w_0|w_{c,1}, w_{c,2}, ..., w_{c,C}, D)) \tag{3.14}$$

Similar to other word embedding algorithms, after training these paragraph embedding vectors can be used for different machine learning applications such as clustering algorithms or regression, etc. The biggest advantage of the PV-DM approach is that they maintain or consider the word order which helps to preserve a lot of semantic information for different words within the paragraph [41].

3.4.2 Paragraph Vector Distributed Bag of Words (PV-DBOW)

PV-DBOW [41] is another variant of doc2vec that doesn't concatenate or averages the paragraph vector with the word vectors to predict the target word or next word within

the sample window like PV-DM. This method suggests ignoring the context words in the input but forcing the model to predict words randomly sampled from the paragraph in the output.

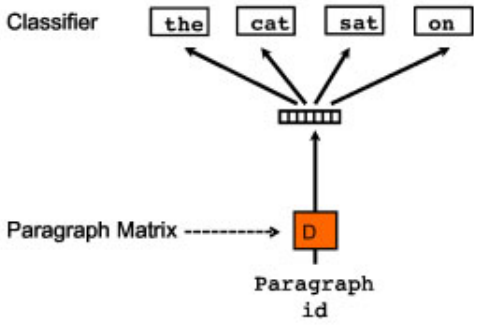

Figure 3.8: This architecture is called as Distributed Bag of Words version of paragraph vectors which uses the paragraph id or document id to predict the context words. This figure is also taken from [41].

The architecture of PV-DBOW is shown in figure 3.4.2 where it samples a fixed context window and a random word within that context window which forms a classification task given the paragraph vector in each iteration. This model does not maintain or use any ordering of the words or context for prediction. It also refers to the paragraph id as D and other nomenclature used for it is the same PV-DM model defined above. The training procedure is also similar with the hidden layer weights W only having D as input and the output layer predicts probability vectors for all target words.

This approach is similar to the skip-gram model in Word2vec as it uses the target word to approximate the context words. The loss function for this model is similar to word2vec skip-gram defined in equation 3.13. The primacy of this model is that it stores less information as word vector weights are not stored in this like PV-DM. It only stores the softmax weights from each iteration. This algorithm is faster as there is no need to save the word vectors but has a lower performance than the PV-DM model [41].

4 Embedding Product Data

Traditional methods to classify or cluster similar products use the manual assignment of categories or tags based on categorical or descriptive attributes like brand, size, etc. These categories or tags are then used to recommend similar products in an e-commerce shop. However, this method is limited and not scalable to a large amount of data and this is where the concept of product embeddings is useful.

Embeddings in e-commerce for product data can be defined as a machine learning procedure in which we learn dense numerical representations for products. These representations of products are generated by mapping products in euclidean space and are called product vectors. These embeddings are assigned positions in the embedding space such that, similar products are closer to each other while different products are far from each other. Product knowledge has significant importance in the rapidly evolving e-commerce world and these product vectors show the relation between products. [82].

As the product data from e-commerce portals is in text format, we can use any of the existing NLP algorithms capable of creating embeddings from the text. In our approach, we implemented a mixture of two commonly talked-about NLP algorithms word2vec [76] and its extension doc2vec [41]. Our approach for creating product embeddings is trained without having an NLP-specific goal like language translation or word recommendations. It rather serves as an embedding generator for product data.

Techniques used for embedding products usually have a specific goal such as product recommendation, collaborative filtering, or content-based filtering. Item2vec [10] is a collaborative filtering algorithm that analyzes item similarities based on item relations and uses it for product recommendations. Product2vec [19] and Meta-prod2vec [27] are other algorithms that create embeddings to be used for product recommendations. They use content-based filtering where item similarities are captured based on user sessions and their interactions with different items in a single session.

Our goal is therefore to create a representation for products in the euclidean space, such that the similarities between products are preserved in the embedding space. These embeddings can then be used for numerous downstream tasks instead of having only one of the above-mentioned goals. Therefore, the proposed algorithm only requires information describing the product and, we do not need any labeled data for training. It is an unsupervised product embedding framework that can be used on different datasets from various domains. As the foundation of our approach is based on word2vec, it is both fast and scalable. It can handle all types of attributes: descriptive text, categorical, ordinal, single word/token, and even numerical. These generated product embeddings can then be used for detecting duplicate products. We can also cluster products using their representations and this helps categorize all product data from Unite's platform into

different categories and create taxonomies.

4.1 Embedding Techniques for Structured Data

There have been many attempts in different machine learning fields to construct numerical vectorial representations for structured data. Performing machine learning on structured textual data in form of sequences, trees, graphs, or tables having well-defined schemas is complex as such data does not have a vectorial form [16]. Therefore, many techniques have emerged to construct vectorial representations of structured data, from kernel and distance approaches to the recurrent, recursive, and convolutional neural networks [54]. In this thesis, we worked with a state-of-the-art approach in representation learning to create useful representations for such data.

In our approach, we create embeddings for product data which is in the form of structured tabular records. These structured records have a fixed number of attributes per record. We used a text embedding framework for attribute encodings, which allows us to draw analogies between documents and structured records. Text documents consist of paragraphs, sentences, and words whereas structured records maintain a hierarchy of records, attributes, and words.

Unlike sentences within a document, records do not have a defined sequence or follow any meaningful ordering. There are no relationships between sentences across documents, but attributes within records can be strongly related [16]. Each attribute within a structured record presents different information ranging from categories to descriptive texts. It is also possible for some attributes within a record to be empty.

In this thesis, we introduced a method that leverages all these properties of structured records to train embeddings by implementing word2vec [76] and doc2vec [41] frameworks simultaneously for encoding individual attributes within a record. These embeddings capture the similarities, and relationships between products.

4.1.1 Attribute Embeddings

The foundation of our algorithm is based on the idea of creating individual embeddings for each attribute within the record separately based on the information they contain. Then these individual attribute embeddings can be combined to generate record embeddings. This means in our embedding model, every attribute has its own vocabulary and has embeddings of dimension D_i where i defines the $i(th)$ attribute within the record r. We divided the records within an e-commerce dataset into categorical and descriptive attributes and the embeddings for them are generated as follows:

Categorical attributes

Categorical attributes consist of information that can be categorized or divided into groups such as IDs, company or manufacturer names, etc. They can only take a limited number of possible values. In our data, we have categorical columns like $catalog\_id$, set_id, ean, $manufactuter\_id$ and $manufacturer\_name$. The embeddings for all these attributes are created by using the skip-gram word2vec explained above.

The word2vec [76] model on these categorical attributes is trained separately for each attribute by using vocabulary V_i from attribute i, having an embedding dimension of 300 for each categorical attribute. The embedding dimension N in the word2vec model has a similar tuning mechanism as the number of hidden layers in the deep neural network. We get the best results having the size of embedding dimension at 300. The context window is set to 10.

Going through the skip-gram training procedure, it takes input as $|V_i|$ dimensional one-hot vector, which is mapped into a vector v of dimension 300 via an embedding lookup matrix U of size ($|V|$, N), which gives us the hidden layer of the model. This hidden layer vector v is then mapped by another context embedding matrix V to a final $|V|$-dim vector. In our specific problem, we are interested in the hidden layer weights as they represent the embeddings for given input words.

Subsequently, a softmax activation function is applied to normalize the layer to a probability distribution. It uses a cross-entropy loss function to compare the probability distribution of the final layer vector with input vectors. Optimization is performed by back-propagating the error and word embeddings are updated via gradient descent. The loss function from 3.13 with the parameters from our dataset is given in equation 4.1:

$$
\begin{aligned}
\mathscr{L} &= -log(P(w_{c-10}, w_{c-9}, .., w_{c-1}, ..., w_{c+10|w_c})) \\
\mathscr{L} &= -log(\prod_{j=0, j\neq 10}^{20} P(w_{c-10+j}|w_c)) \\
\mathscr{L} &= -log(\prod_{j=0, j\neq 10}^{20} P(u_{c-10+j}|v_c)) \\
\mathscr{L} &= -log(\prod_{j=0, j\neq 10}^{20} \frac{exp(u_{c-10+j}^T v_c)}{\sum_{k=1}^{|V|} exp(u_k^T v_c)}) \\
\mathscr{L} &= -\sum_{j=0, j\neq 10}^{20} u_{c-10+j}^T + 20log \sum_{k=1}^{|V|} exp(u_k^T v_c)
\end{aligned} \tag{4.1}
$$

Descriptive attributes

Columns or attributes such as $product\_name$ and $product\_description$ have long textual information, and therefore, are considered descriptive attributes. They are composed of a different combinations of words and sentences. Descriptive attributes present discrete data and are difficult to classify into groups. For creating embeddings of such long sentences or paragraphs, we use the PV-DM approach of doc2vec [41].

As we explained above, the architecture for doc2vec is similar to word2vec with it having an extra unique document id for representing each document. In the PV-DM approach of doc2vec, this document id referred to as d remembers what is missing from the context or it acts as a memory. In these descriptive attributes, we pre-process the long string texts (explained in the next section) and tokenize them. In our implementation, we assign a unique tag to each record l within a descriptive attribute to be used as d.

The PV-DM doc2vec model also is trained by using vocabulary V_i from each attribute i, having an embedding dimension of 600. As the embeddings in doc2vec carry information from the whole sentence or paragraph, the embedding dimension for it is more than word2vec. The embedding dimension in doc2vec is also a self-defined parameter and we set it at 600 to effectively capture the relationship between different descriptive attributes. The context window size for it is also 10.

The training for doc2vec is similar to Word2vec, with just an additional document vector or id d which is also taken as a member of the context set. The optimization process is similar to the one defined above for word2vec. At the end of the training process, document embeddings are taken from hidden layer weights. The objective function for PV-DM doc2vec which is similar to cbow word2vec, for our specific parameters is given as:

$$\mathscr{L} = -log(\mathbb{P}(w_0|w_{c,1}, w_{c,2}, ..., w_{c,10}, d_l)) \tag{4.2}$$

4.1.2 Skip Gram Negative sampling

Both word2vec and doc2vec algorithms compute the similarity between context words and target words and assign vector representations to them based on their similarities. This is achieved by using the objective function as defined in 4.1. The numerator in this objection function computes the similarity between the target word and the context word. While denominator is a normalizing factor, which computes the similarity between all other context words (entire vocabulary) and the target word. As the size of vocabulary can be huge with the possibility of having millions of words, the computation gets really slow and unmanageable [72].

To overcome this issue, the concept of negative sampling is being introduced [47]. Negative sampling maximizes the similarity of words in similar context and minimizes it for words occuring in a different context, by defining a new objective function. In simpler terms, instead of looping over the whole vocabulary, it randomly selects some k words other than context words and uses them for the optimization of the objective function. The objective function using negative sampling by Mikolov in his paper [47] is given as:

$$log(\sigma(v_{w_i}^{'T} v_{w_I})) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)}[log(\sigma(-v_{w_c}^{'T} v_{w_I}))] \tag{4.3}$$

The value for k as given in equation 4.3 can be between $1 < k < 20$ depending on the size of the dataset. In our model we set the value of k at 5, which means 5 negative samples are selected using a "unigram distribution" given in 4.3 as $P_n(w)$. By using this distribution, frequent words in the corpus are more likely to be selected as negative samples. Given a list of words, the probability of word w_i being selected as a negative sample using a unigram distribution is calculated as the total number of times w_i appeared in the corpus, divided by the total number of words in the sample.

$$P(w_i) = \frac{f(w_i)}{\sum_{j=0}^{n}(f(w_j))} \tag{4.4}$$

The authors claimed in their paper [47] that they tried different variations in the above formula and raising the power of word count to $3/4$ resulted in the best performance. This change tends to increase the probability for less frequent words to be sampled as well.

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^{n}(f(w_j))^{3/4}} \tag{4.5}$$

To derive the objective function given in 4.3, we assume that (w,c) is a word and context pair in the training data. We can denote this by $P(Z = 1|w,c)$, which gives the probability that this pair belongs to training data. Similarly, the probability of word context pair not belonging to training data is given as $P(Z = 0|w,c)$ = $1 - P(Z = 1|w,c)$. In this probability distribution, we denote the trainable parameters as θ and out of training data words as Z' and the function to be optimized for skip-gram negative sampling is given as:

$$argmax_\theta \prod_{(w,c)\in Z} P(Z = 1|w,c,\theta) \prod_{(w,c)\in Z'} P(Z = 0|w,c,\theta) \tag{4.6}$$

Here, we convert the maximum of products to the maximum for sum of logarithms and substitute $P(Z=0|w,c)$ with $1-P(Z=1|w,c)$

$$argmax_\theta \sum_{(w,c)\in Z} log(P(Z=1|w,c,\theta)) + \sum_{(w,c)\in Z'} log(1-P(Z=1|w,c,\theta)) \tag{4.7}$$

$P(Z=1|w,c)$ is computed using the sigmoid function which is given as:

$$P(Z=1|w,c;\theta) = \sigma(v_c.v_w) = \frac{1}{1+e^{-v_c.v_w}} \tag{4.8}$$

the above equation represents context words as v_c and target words as v_w and the final objective function can be written as

$$argmax_\theta \sum_{(w,c)\in Z} log(\sigma(v_c.v_w)) + \sum_{(w,c)\in Z'} log(\sigma(-v_c.v_w)) \tag{4.9}$$

The above equation is similar to the objective function 4.3 given in [47] summed over the entire vocabulary.

4.2 Triplet Loss on Embedding Vectors

After generating embeddings for both categorical attributes and descriptive attributes using skip-gram negative sampling, we use a triplet loss function on each individual embedding to improve them as defined in this paper [16]. To implement this triplet loss, we iterate over embeddings of all records within all attributes separately. Each iteration in the training of this loss consists of selecting a triplet (anchor, positive target, negative target).

Here, a positive target is defined as the embedding of a single record for the $i-th$ attribute in the current iteration over all records given as $e_i^{(r)}$. A negative sample, given as $e_i^{(\neg r)}$ in each iteration is selected randomly from the set of records within the attribute embedding. We further define a constraint that $e_i^{(r)}$ = $e_i^{(\neg r)}$ which means the embedding for positive target should not be similar to a negative target. The anchor is calculated as the sum of all the remaining attributes omitting the positive target. The formula for calculation of anchor for an attribute i is given as:

$$anchor_i^{(r)} = \sum_{j=1.j\neq i}^{N_t} \frac{1}{N_t} e_j^{(r_t)} \tag{4.10}$$

where $e_i^{(r_t)}$ represents embeddings for all records N within an attribute i. After calculating anchor, positive target, and negative target, we use the triplet loss function [80] to make sure the loss function train embeddings with similar context (identical to the target) and negative samples are separated by margin α [60]. The loss function to be minimized for all records N_{rec} within an attribute is given as:

$$Loss = \sum_{r=1}^{N_{rec}} \sum_{n=1}^{N_{neg}} max(0, d(anchor_r, e^{(r)}, e^{r_n \neq r)}) \tag{4.11}$$

where $d(anchor_r, e^{(r)}, e^{(r_n \neq r)})$ is a function calculating the difference between anchor, positive target and negative target. For a single record within an attribute considering margin as α, anchor as a, positive target as p, and negative target as n, this difference can be calculated as:

$$d(a,p,n) = CosineDistance(a,p) - CosineDistance(a,n) + \alpha \tag{4.12}$$

We train this loss function on each attribute individually for all the records within an attribute for 500 epochs. Margin is set at 0.5 and the loss is optimized using Adam optimizer [39] with a learning rate of 0.001.

4.3 Meta-Embedding

Meta-embedding is defined as the process that tries to learn more accurate embeddings by combining different individual embeddings given as input. An important step in NLP is to select which pre-trained word embeddings to use for a given task. Thus, combining multiple pre-trained embeddings to solve different tasks has become a viable option for more complex tasks like text similarity, classification, semantic relatedness, etc [59]. Meta-embedding has gained popularity among NLP practitioners because of its capability of combining semantics from different embeddings into one [20]. These embeddings have two advantages: first, they perform better than individual embeddings. Second, they include more words than a single embedding [81].

In this thesis, we have separate embeddings for different categorical and descriptive attributes trained using word2vec [76] and doc2vec [41] in an unsupervised manner. Each of these embeddings is trained by a different neural network on a different vocabulary and these diverse embeddings can be combined to learn better performing record embeddings. We will discuss and implement two simple and popular techniques based on concatenation and linear transformation to combine different source embeddings.

4.3.1 Concatenating Source Word Embeddings

The simplest approach for creating meta-embeddings from different embedding sets is to concatenate these embedding sets [81]. Although concatenation looks naive, but it has been proven to be a good baseline of performance for meta-embeddings [38]. In concatenation, the meta-embedding S is generated by concatenating individual attribute embeddings, one each from the pool of n different attributes.

Concatenated meta-embeddings do not require the transformation of individual meta-embeddings to bring them into the same dimensional space before combining. They can be combined directly, and the dimensionality of the concatenated meta-embedding is given as the sum of the dimensionality of all the individual embeddings. In our example, categorical and descriptive embeddings have different sizes and we concatenate them to get record embeddings. The disadvantage with simple concatenation is that the resulting meta-embedding would have a very high dimension, which results in a great increase in training parameters.

The figure below gives an example of concatenating 4 different source embeddings with different embedding dimensions.

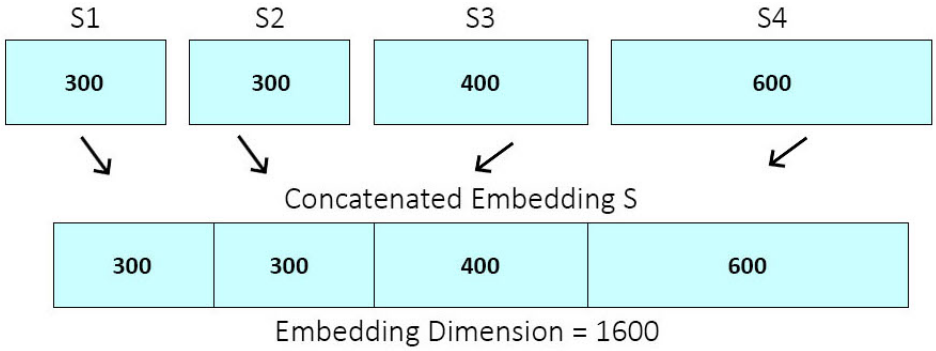

Figure 4.1: Explanation of generating meta-embeddings by concatenating source embeddings

4.3.2 Averaging Source Word Embeddings

As simple concatenation of embeddings results in a higher number of dimensions, a method has been proposed to get a meta-embedding by averaging all source embeddings [38]. This approach might not seem intuitive and evident, as differently trained source embeddings might not have any correlation. Also, it is not straightforward to take an average of individual embeddings having a different number of dimensions.

Despite these problems, averaging can still provide good baseline performance for

meta-embeddings without increasing the dimensions of the embeddings [38]. This averaging first requires a local transformation step where all source embeddings are transformed into higher space if they have different dimensions. This is done by padding individual embeddings with zeros at the end such that the dimensions of each embedding are equal to the one with the highest dimensionality.

Suppose we have embeddings with dimension 300 referred as S_1 and S_2, S_3 with embedding dimension 400, and one with dimension 600 referred to as S_4. We did not make any assumption that these source embeddings are trained in the same setting or method. The S_1 and S_2 with dimension 300 will be right padded with 300 zeros and S_3 is padded with 200 zeros so that all source embeddings are in the same space. As the dimensions for all embeddings are the same, we can directly compute the averaged meta-embedding for all padded source embeddings S_1, S_2, S_3, and S_4.

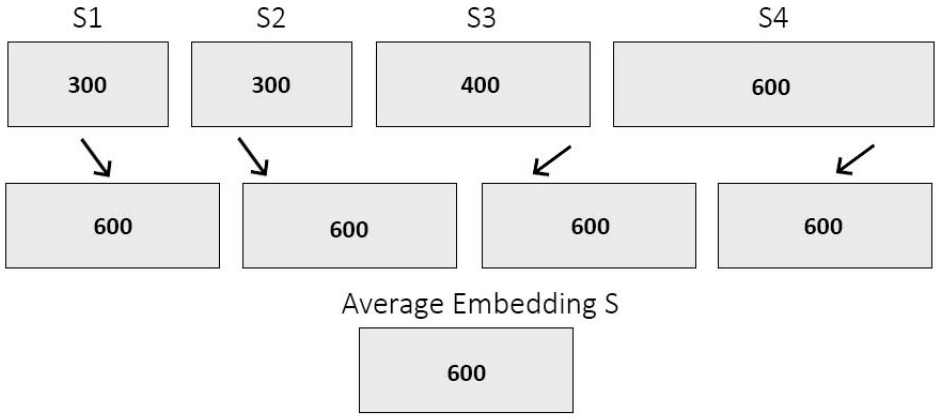

Figure 4.2: Explanation of generating meta-embeddings by averaging source embeddings

4.4 Architecture Of Embedding Model

Figure 4.3 gives a pictorial representation of step by step process of our approach used for creating embeddings of product data in the form of structured tabular records. Each record is separated into categorical and descriptive attributes where total categorical attributes are referred to as M and descriptive attributes are referred to as L. Skip-gram variant of the word2vec model is trained on all M attributes individually having a different vocabulary and the embedding dimension is set at 300. Similarly, embeddings for L descriptive attributes are trained using PV-DM doc2vec, and the resulting embedding dimension is 600.

All of these attribute embeddings having N records are trained with a triplet loss that iterates over all records and assigns anchor, positive and negative samples from all records within an attribute. This loss function then uses cosine distance to modify attribute embeddings such that similar attributes have similar embeddings while preserv-

ing the embedding dimension for the different types of attributes. After training triplet loss these embeddings are combined by concatenating or averaging individual attribute embeddings as described above to create record embeddings. These record meta-embeddings represent the final embedding for the given product data.

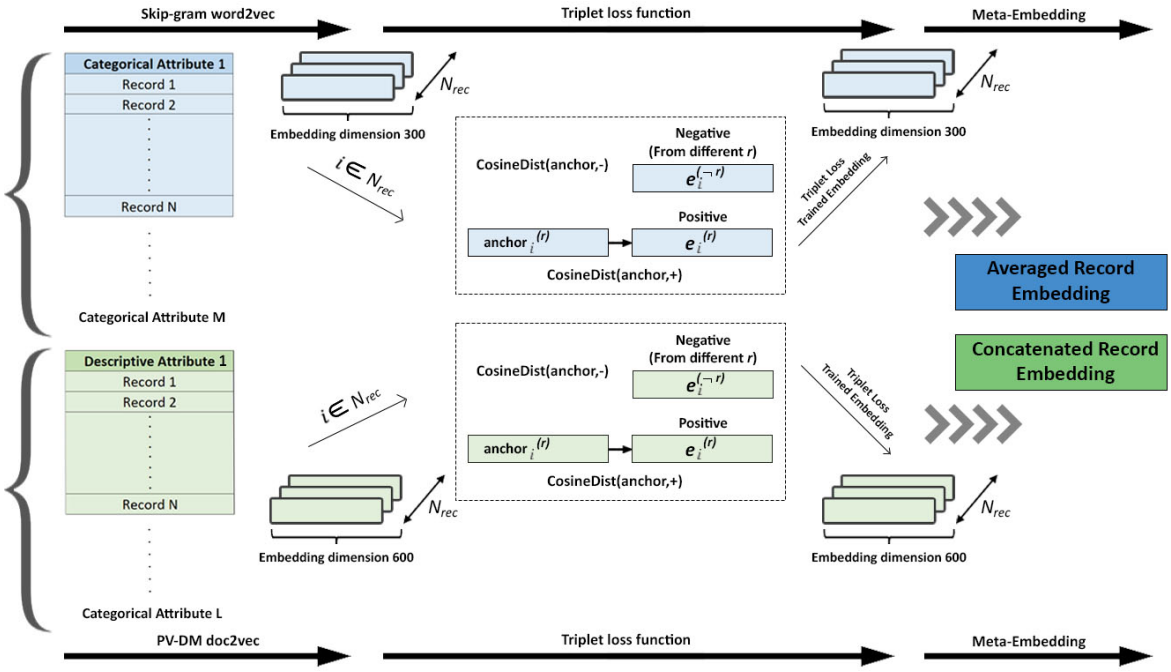

Figure 4.3: Architecture of our embedding model with every step

5 Experimental Setup and Evaluations

In this chapter, we will apply and evaluate the embedding model explained above on Unite's product data. This chapter is further divided into four sections. The first section describes the dataset in detail and explains our pre-processing steps on the data before applying the model. In the second section, we evaluate the performance of our model on the dataset defined in the previous section using different evaluation techniques. The third section summarizes the result from our embedding model. The last section explains the python implementation of our approach.

5.1 Data

5.1.1 Data Description

The dataset we used for our experiment is an article or product data gathered from Unite's e-commerce platform. It has different attributes describing the product details. Out of those attributes, we have trained our model on 7 relevant attributes for creating product embedding, which are $product\ name$, $description$, ean (international article number), $catalog\ id$, $set\ id$, $manufacturer\ id$ and $manufacturer\ name$. To evaluate our model, we have learned embeddings by using $10,000$ records from this dataset. As $product\ name$ best describes an individual product, some of our evaluations on this dataset are only visualized using $product\ name$.

5.1.2 Data Pre-processing

In most cases, text data generated from different streams are not entirely clean and require some pre-processing. Here, we explain a few vital pre-processing steps before creating representations for the given data. These pre-processing steps help transform or shape data to be best understood by our model. They play an important role in improving the overall performance of the algorithm.

Unlike many other datasets, where the correctness of a data value can be reasoned with its occurrences, quantitative statistics of article data do not necessarily imply correctness. That is to say, the fact that a certain manufacturer occurs less than the others must not indicate an error, but it can be due to uneven numbers of existing manufacturers on the market.

Cleaning Null Values

In the first step, from our dataset gathered from Unite's e-commerce platform, we will omit all records in which the attribute *product name* is empty. Null values in *product name* usually occur due to a problem with the source system that captures the product data. As most of our visualization is based on *product name* and we also analyze the similarity of the product embeddings based on their names. It is, therefore, important not to have null values in *product name*.

In all other attributes, we replace the null or empty attribute with the word 'None' to have a consistent token representing all the empty values within all attributes. We also lowercase all the words from all attributes before training the model. Missing values in our data, before replacing them with *None* for all attributes other than *product_name* are shown in figure 5.1. White lines in this mentioned figure on the y-axis represent the number of missing values or *Nan* in each column.

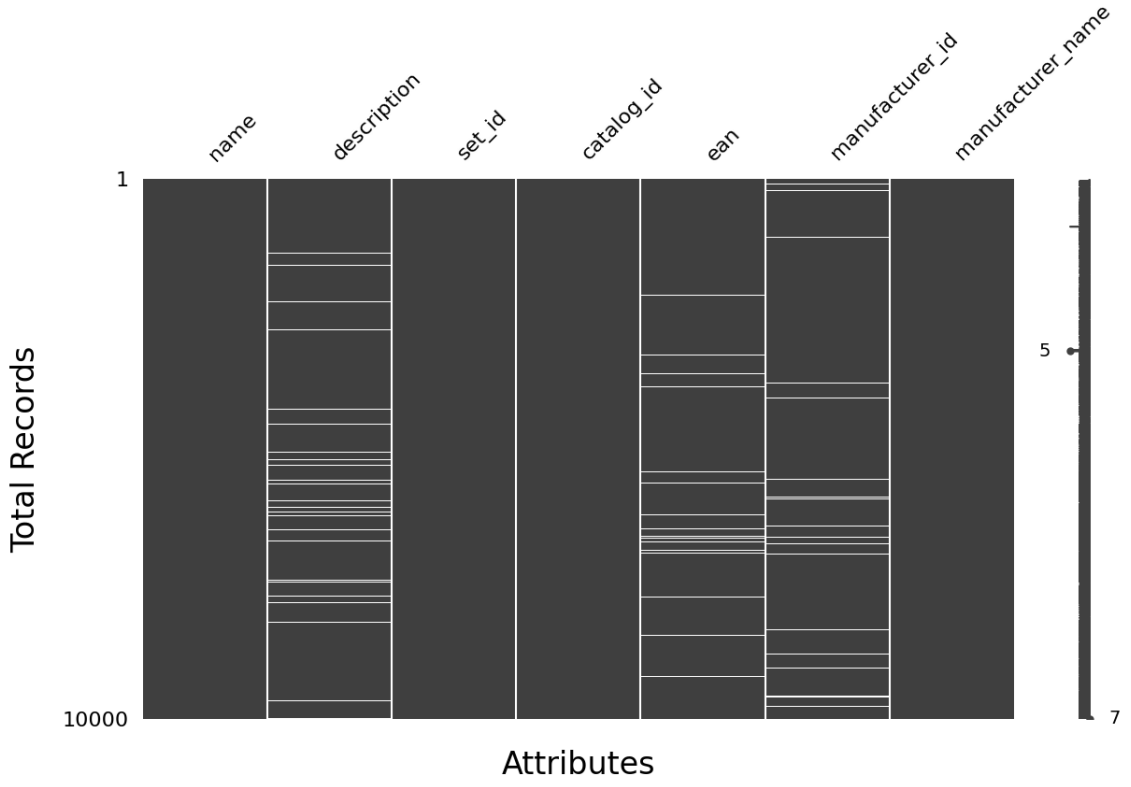

Figure 5.1: Showing missing values in a matrix where whites represent missing values in each attribute

Removing Punctuations and Special Characters

In text processing tasks, the first pre-processing step is to eliminate any special characters in the input dataset. Word2vec and doc2vec algorithms described above work on capturing similarities between words, and punctuations do not have any semantic similarity with other words. Therefore, we remove punctuations and special characters from both our categorical and descriptive attributes.

Removing White Spaces

In our categorical attributes, we want the values to be processed as single words. Attributes like $manufacturer\ name$ can consist of multiple words. For these attributes to be processed by word2vec as single words we replace the white spaces with hyphens (-). Removing spaces made capturing the similarity between different values within the attribute easier. For example, a manufacturer name like $under\ armour$ would be transformed into $under\text{-}armour$.

Tokenizing words

Tokenizing words is defined as the process of transforming the descriptive columns such that each word is isolated. It divides each string text into lists of substrings. We tokenize our descriptive attributes such that each sentence after the removal of punctuation and stop words consist of individual tokens. After tokenizing it, they are tagged with a paragraph-id for each sentence.

5.2 Evaluations

Word embedding models are extensively been used in different NLP tasks that require capturing maximum semantic information [49]. Various techniques have been suggested in the literature to evaluate the quality of word embeddings [79]. Despite the omnipresence of word embeddings in NLP, there is still an ongoing debate about the capability of different methods for evaluating word embeddings. To date, there is no consensus on one single approach that best evaluates these models [24].

In this section, we will discuss different techniques that can comprehensively evaluate embeddings of products generated from structured data. Here, We will go through some of the widely-used and popular evaluation techniques. These evaluation techniques are divided into two categories: intrinsic evaluations and extrinsic evaluations.

5.2.1 Intrinsic Evaluations

Intrinsic evaluations are used to check the quality of an embedding irrespective of any defined natural language processing tasks. [79]. In text analytics, these evaluation methods directly test the quality of the embeddings by checking syntactic and semantic similarities between individual words or phrases. These intrinsic evaluation methods are further divided into absolute intrinsic and comparative intrinsic evaluation.

Intrinsic evaluations are defined as experiments in which relations in word embeddings are evaluated individually and results are based on the comparisons between those evaluations [79]. Comparative evaluations are based on human judgment where given different embeddings from various models, an assessor evaluates or judges which method performs the best [7]. Here, we will discuss only the absolute intrinsic evaluation methods as comparative intrinsic evaluators require additional resources.

Record Similarity

Record similarity is defined as the method that checks the distance between similar records in the embedding space. Also referred to as 'semantic relatedness', it measures the extent to which a record shares similarities with other records in the corpus. These semantic similarity or semantic relatedness tasks rate semantic proximities between pairs of words using different similarity measures [67]. These method is usually evaluated by maintaining a list of pairs of words along with their similarity values [71].

In our example, we will evaluate and show the top 10 similar products to a randomly chosen product embedding (mentioned on top of the table) using cosine similarity and euclidean similarity. These similar articles will then be used to gauge the performance of the model's performance. Its interpretability will be assessed by exercising human judgment. We will evaluate the record similarity from embeddings created by using both explained meta-embedding techniques. The tables below show the similarity between product embeddings using $product\ name$ only. Some of these $product\ names$ are similar as other attributes like ean, $catalog\ id$, or $manufacturer\ name$ within this record embedding are different.

Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	
Product Name	Similarity
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.8916
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.8914
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.881
tesa Folienband extra Power Universal, 50 mm x 10 m, schwarz (8756348)	0.8875
tesa Reparaturband extra Power, schwarz, 10m x 50mm	0.8863
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.8838
tesa Reparaturband schwarz 10mx50mm Power Perfect, extra Power	0.8832
Gewebeband ext.Power 56348 schwarz L.10m B.48mm Rl.TESA	0.8831
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.8830
Tesa Extra Power Universal schwarz 48 mm x 10 m Gewebeklebeband	0.8826

Table 5.1: Top 10 Similar products to a given product by using cosine similarity on Averaged meta-embeddings

Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	
Product Name	Similarity
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.7374
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.7351
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.7343
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.7286
Gewebeband ext.Power 56348 schwarz L.10m B.48mm Rl.TESA	0.7280
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.7262
tesa Folienband extra Power Universal, 50 mm x 10 m, schwarz (8756348)	0.7236
Tesa Extra Power Universal schwarz 48 mm x 10 m Gewebeklebeband	0.7151
tesa Reparaturband extra Power, schwarz, 10m x 50mm	0.7150
tesa extra Power Universal 10m 50mm schwarz	0.7144

Table 5.2: Top 10 Similar products to a given product by using cosine similarity on Concatenated meta-embeddings

Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	
Product Name	Similarity
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.6652
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.6649
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.6607
tesa Folienband extra Power Universal, 50 mm x 10 m, schwarz (8756348)	0.6594
Tesa Extra Power Universal schwarz 48 mm x 10 m Gewebeklebeband	0.6574
Gewebeband ext.Power 56348 schwarz L.10m B.48mm Rl.TESA	0.6573
tesa Reparaturband schwarz 10mx50mm Power Perfect, extra Power	0.6572
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.6567
tesa Reparaturband extra Power, schwarz, 10m x 50mm	0.6561
tesa extra Power Universal 10m 50mm schwarz	0.6554

Table 5.3: Top 10 Similar products to a given product by using euclidean distance on Averaged meta-embeddings

Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	
Product Name	Similarity
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.6578
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.6573
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.6563
Gewebeband ext.Power 56348 schwarz L.10m B.48mm Rl.TESA	0.6549
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.6533
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.6532
tesa Folienband extra Power Universal, 50 mm x 10 m, schwarz (8756348)	0.6518
Tesa Extra Power Universal schwarz 48 mm x 10 m Gewebeklebeband	0.6491
tesa extra Power Universal 10m 50mm schwarz	0.6481
tesa extra Power Universal 10m 50mm schwarz	0.6480

Table 5.4: Top 10 Similar products to a given product by using euclidean distance on Concatenated meta-embeddings

We calculated the top 10 most similar articles in terms of similarity to the embedding of a given article where higher values represent high similarity. The tables above show that both concatenated and averaged record embeddings effectively preserve the similarities between object space and the embedding space. The effect of different similarity measures on calculating the nearest neighbors has been studied by using two different measures. As the top 10 articles computed by cosine similarity are similar to euclidean distance, it is safe to conclude that embeddings robustly capture the structure from object space.

The similarity value for concatenated embeddings using cosine distance is less than that of averaged embeddings. It is because the embedding dimension for concatenated meta-embeddings is higher, as it captures more information from a record and results in more dissimilarity between two articles. Value for euclidean similarity is derived from euclidean distance by normalizing and updating it as explained in preliminaries, resulting in both averaged and concatenated embeddings scaled at 0.6.

Visualizing Embeddings

Another technique to evaluate these record embeddings is to visualize them in a 2-dimensional space. In this visualization, records similar to each other would be closer while dissimilar products. As described in our model, these product embeddings usually have very high dimensions, which means it is difficult to visualize these products in a 2-dimensional space. To overcome this problem, we implemented the t-sne technique defined above, which reduced the dimensionality of these embeddings to 2.

After reducing dimensions using t-sne, we visualized these record embeddings from both concatenated and averaged meta-embeddings using manufacturer names and product names. The product names are long strings and difficult to interpret in a static figure. Within these figures, product names are limited to 100, and manufacturer names are limited to 250 for better visibility in a scatter plot.

Figure 5.2: Visualization of Manufacturer Name from concatenated meta-embeddings using t-sne

Figure 5.3: Visualization of Product Name from concatenated meta-embeddings using t-sne

Figure 5.4: Visualization of Manufacturer Name from averaged meta-embeddings using t-sne

Figure 5.5: Visualization of Product Name from averaged meta-embeddings using t-sne

These visualizations show that same manufacturers are closer to each other in the embedding space while different manufacturers are far apart. From figures 5.2 and 5.4, we can claim that our embeddings also capture the semantic similarity between these records as manufacturers like "cederroth" and "holth ausmedical", which are both pharmaceuticals-related companies, are very close. On the other hand companies like ”varta”, a battery manufacturer, are far apart from the pharmaceuticals cluster. Also, in figure 5.2, clusters from the similar category of manufacturers are closer.

From figures visualizing embeddings using the product name, we can interpret again that similar products are very close to each other in the embedding space and different products are far apart. Clusters for similar kinds of products like mice and batteries that both belong to electronics are closer than others. In these visualizations, both concatenated and averaged meta-embeddings seems to have similar quality of embeddings as they plot records in the more or less same manner.

5.2.2 Extrinsic Evaluations

Extrinsic evaluations are based on the ability to use embeddings as input features for performing different tasks like classification, clustering, etc. They use these record embeddings as feature vectors for supervised machine learning tasks [7]. For example, in an NLP context, these extrinsic evaluations could be used for tasks like part-of-speech tagging [43], named-entity recognition [84], language translation [23] and sentiment analysis [62]. Nevertheless, by the above explanation of extrinsic evaluations, we hypothesize that the performance of an embedding model can be evaluated using any downstream task.

In our example, we used these record embeddings for clustering tasks. As defined in the dataset, we have 10,000 records, and we used clustering results from a hierarchical clustering performed on Unite’s dataset, which served as a benchmark. To compare these pre-existing clustering results, we clustered our record embeddings by using K-means 2.2.1 and BIRCH 2.2.2. In both of these algorithms, the number of clusters is set equal to the number of clusters in pre-existing clustering. Afterward, these clustering results were compared and evaluated using different external cluster metrics like ARI 2.4.1, FMI 2.4.2, and v-measure 2.21, where we also show the homogeneity 2.15 and completeness 2.18 scores used for calculating v-measure. Cluster metrics evaluated on clusters formed by two different clustering techniques are given as follows:

K-means Clustering

For k-means clustering on these record embeddings, the k (number of clusters) is taken as 278. This is the number of clusters we have from another clustering algorithm on source data and it runs for 10 iterations. We will also evaluate the impact of the triplet

loss function by evaluating cluster metrics on these meta-embeddings trained both with and without triplet loss.

Cluster Metric	Avergaed Meta-Embedding	Concatenated Meta-Embedding
ARI	0.109	0.119
FMI	0.117	0.125
Homogeneity	0.570	0.576
Completeness	0.595	0.598
V-measure	0.582	0.586

Table 5.5: Evaluating cluster metrics on record embeddings generated without triplet loss clustered by using k-means

Cluster Metric	Avergaed Meta-Embedding	Concatenated Meta-Embedding
ARI	0.592	0.728
FMI	0.60	0.727
Homogeneity	0.870	0.903
Completeness	0.901	0.921
V-measure	0.885	0.912

Table 5.6: Evaluating cluster metrics on record embeddings generated with triplet loss clustered by using k-means

All of these cluster metrics are evaluated between 0 and 1, where 1 represents the highest or exact similarity between two different clustering outcomes. In table 5.5, we can see that meta-embeddings generated from attributes not trained with triplet loss give values close to 0 for ARI and FMI in both concatenated and averaged meta-embeddings. These values from these metrics represent that the clustering result has significantly less or no similarity to the ground truth clustering we are comparing it with. V-measure along with homogeneity and completeness score values are close to 0.5, which also means there is a low correlation between the two clustering results.

On the other hand, when we compare the v-measure score from table 5.6, which are triplet loss trained embeddings, the value is close to 1 for both concatenated and average meta-embeddings. This result shows a strong correlation between the two clustering results. Almost 90% of the elements are correctly clustered as given in ground truth clustering. ARI score of 0.72 while using concatenated embeddings for clustering

represents an adequate recovery of the true (ground truth) clustering results. FMI score greater than 0.7 also represents a considerable similarity between the two clustering algorithms.

BIRCH Clustering

BIRCH is a hierarchical clustering algorithm that performs clustering in a single iteration by scanning all data at once. It transforms data into a smaller tree-like structure by using CF trees. The hyperparameters within BIRCH clustering are branching factor, threshold, and the number of clusters. In our example, we initialize the number of clusters with 278 which taken from prior clustering results, the branching factor is set at 50 and the threshold is 0.1. Like k-means clustering, we will evaluate it using the cluster metrics mentioned above and also the impact of triplet loss on these cluster metrics.

Cluster Metric	Averaged Meta-Embedding	Concatenated Meta-Embedding
ARI	0.020	0.056
FMI	0.053	0.076
Homogeneity	0.396	0.497
Completeness	0.542	0.565
V-measure	0.458	0.529

Table 5.7: Evaluating cluster metrics on record embeddings generated without triplet loss clustered by using BIRCH

Cluster Metric	Averaged Meta-Embedding	Concatenated Meta-Embedding
ARI	0.687	0.690
FMI	0.694	0.696
Homogeneity	0.908	0.908
Completeness	0.921	0.922
V-measure	0.914	0.915

Table 5.8: Evaluating cluster metrics on record embeddings generated with triplet loss clustered by using BIRCH

Similar to results from cluster metrics on k-means clustering, we can deduce that clustering performed on meta-embeddings generated from individual attributes without triplet loss generate clusters completely different from the ground truth. As shown in table 5.7, the values for ARI and FMI in both concatenated and averaged meta-embeddings are close to 0. V-measure value from this is also close to 0.5, which again means that there is low correlation between the two clustering results.

The v-measure score from table 5.8 is better than k-means clustering, and this represents the clustering results from BIRCH clustering on both meta-embeddings are almost correlated to true clustering. Homogeneity and completeness values are both more than 0.9, where the homogeneity score represents 90% of the elements within our clusters belonging to the same given true clusters. The completeness score shows that more than 90% of elements from true clusters are contained within the same clusters from the BIRCH algorithm. ARI score also represents a sufficient recovery of original clusters from both meta-embeddings. FMI score for both embeddings is similar and close to 0.7 representing a reasonable similarity between the two outcomes.

5.3 Results

Both intrinsic and extrinsic evaluations show that our model effectively creates embeddings for the given product data. These embeddings preserve the semantic and structural similarities from the original product records and they can be used to effectively cluster similar records. In intrinsic evaluations, we don't see much difference in using concatenated or averaged meta-embeddings for both record similarity and visualization of these embeddings.

For extrinsic evaluations, we evaluated different cluster metrics on k-means clustering and BIRCH clustering. All these metrics compare the two clustering results differently and the interpretation of all of these scores individually is given in the evaluations above. From the tables above, it is shown that concatenated meta-embeddings yield slightly better clustering results than averaged meta-embeddings as it captures more information by preserving all dimension from individual attributes. However, the training time for clustering on concatenated meta-embeddings is also more than averaged meta-embeddings. As the clusters we are using as ground truth was performed using hierarchical clustering, the results from averaged meta-embeddings using BIRCH are notably better than k-means.

From both intrinsic and extrinsic evaluations, we can conclude triplet loss technique used in our approach have a significant impact on the quality of embeddings. Cluster metrics evaluated on embeddings trained without triplet loss show no correlation between the two clustering results. Also, visualization for embeddings that are not trained using triplet loss is scattered as shown in appendices in figures B.1, B.3 and B.2, B.4. They are loosely coupled, and similar products and manufacturers are not as close as in embeddings trained with triplet loss.

5.4 Implementation Details

The algorithms were written using Python 3 programming language. We used conda (Anaconda) [1] as a virtual environment for managing packages used in implementation across different servers. Pandas [56] and NumPy [30] are the essential python packages used for pre-processing data. We used gensim [63], which is a python library that provides an easy implementation for both word2vec and doc2vec algorithms. Other python packages used for triplet loss and visualization include tensorflow [2], matplotlib [34], and scikit-learn (sklearn) [57].

All experiments were executed on a machine with an M1 pro chip powered by 10 cores (8 high-performance cores and 2 efficiency cores) and 16GB of RAM, running macOS 12.4. No GPU acceleration is involved in the experiment. The dataset used for training algorithms has been limited to 10,000 records. The total training time for creating embeddings on both categorical and descriptive attributes using word2vec and doc2vec was 25 minutes. For running triplet loss on each attribute separately for 500 epochs, the total training time was 5 hours. The training for all of these embeddings was parallelized by using multiprocessing.

The hyperparameters value we used for implementing word2vec and doc2vec in gensim are listed below:

Hyperparameters of word2vec

Following are the hyperparameters we used for implementing Skip-gram word2vec along with their explanations:

- vector_size = 300: This is the embedding dimension
- window = 10: It is the size of context window
- negative = 10: Number of samples for negative sampling
- min_count=1: Minimum occurence of the word in the corpus to be considered
- workers = 10: All CPU workers to be used
- epochs = 10: Number of times the algorithm scans the corpus
- sg = 1: 0 means CBOW and 1 means skip-gram
- hs = 0: This means hierarchical softmax is not used

Hyperparameters of Doc2vec

Following are the hyperparameters we used for implementing Skip-gram word2vec along with their explanations:

- vector_size = 600: This is the embedding dimension
- window = 10: It is the size of context window
- negative = 10: Number of samples for negative sampling
- min_count=2: Minimum occurence of the word in the corpus to be considered
- workers = 10: All CPU workers to be used
- dm = 1: 0 means PV-DBOW and 1 means PV-DM
- hs = 0: This means hierarchical softmax is not used
- alpha = 0.025: Initial learning rate
- min_alpha = 0.05: Minimum learning rate as the training progresses

6 Conclusion and Future Work

In this thesis, we presented an approach for creating embeddings for heterogeneous product data from Unite's e-commerce platform in the form of tabular records having different attributes presenting the product details. This model uses popular NLP approaches word2vec and doc2vec in a non-NLP context to create embeddings for each attribute separately. These attribute embeddings present that associations between records in a table can be learned by creating the context in a similar way to learning the relationship between words in a sentence. Our approach also further optimized these embeddings to better preserve relationships in an attribute.

Product embeddings generated by our model have been evaluated both qualitatively and quantitatively by intrinsic and extrinsic evaluation measures. We can deduce from the results of these evaluations, that embeddings created using this model can also be extended to datasets other than e-commerce or product data. This approach provides a generic framework for creating embeddings for data in the form of structured records or tables. The evaluation techniques discussed also illustrate the usability of embeddings generated by this approach for downstream tasks.

Further improvements in this approach can be made by using a meta-embedding technique that uses a non-linear transformation to combine individual attribute embeddings. This kind of technique will better preserve and combine the information from all attribute embeddings. We also need to generalize this framework to generate embeddings for new and unseen products by using the vocabulary from existing embeddings and without re-running the whole process.

Bibliography

[1] Anaconda software distribution. Anaconda Inc., 2016.

[2] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, L. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, T. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, P. Viégas, F.and Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[3] A. Achille and S. Soatto. An overview on data representation learning: From traditional feature learning to recent deep learning. *CoRR*, abs/1611.08331, 2017.

[4] A. Akbik, D. Blythe, and R. Vollgraf. Contextual string embedding for language modeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1638–1649. Association for Computational Linguistics, 2018.

[5] S. Alrumiah and M. Hadwan. Implementing big data analytics in e-commerce: Vendor and customer view. *IEEE Access*, 9, 2021.

[6] E. Amigo, J. Gonzalo, J. Artiles, and F. Verdejo. A comparison of extrinsic clustering evaluation metrics based on formal constraints. *Information Retrieval Journal*, 12:461–486, 2009.

[7] B. Amir. A survey of word embeddings evaluation methods. *CoRR*, abs/1801.09536, 2018.

[8] P. Bachman, W. Buchwalter, and R. D. Hjelm. Learning representations by maximizing mutual information across views. *CoRR*, abs/1906.00910, 2019.

[9] R. Baraniuk, D. Donoho, and M. Gavish. The science of deep learning. *Proceedings of the National Academy of Sciences*, 117(48):30029–30032, 2020.

[10] O. Barkan and N. Koenigstein. Item2vec: Neural item embedding for collaborative filtering. *CoRR*, abs/1603.04259, 2016.

[11] R. Bellman and R. Kalaba. A mathematical theory of adaptive control processes. *Proceedings of the National Academy of Sciences*, 45(8):1288–1290, 1959.

[12] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new

perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[13] F. Bianchi, J. Tagliabue, and B. Yu. Query2Prod2Vec: Grounded word embeddings for eCommerce. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Papers*, pages 154–162. Association for Computational Linguistics, 2021.

[14] A. Bibal, V. Vu, G Nanfack, and B. Frénay. Explaining t-sne embeddings locally by adapting lime. 10 2020.

[15] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information. *CoRR*, abs/1607.04606, 2016.

[16] A. Borthwick and Y. L. A. Sim. Record2vec: Unsupervised representation learning for structured records. In *ICDM*, 2018.

[17] S. Cao, X. Wang, and K. M. Kitani. Learnable embedding space for efficient neural architecture compression. In *International Conference on Learning Representations*, 2019.

[18] C. Chang, W. Liao, Y. Chen, and L. Liou. A mathematical theory for clustering in metric spaces. *CoRR*, abs/1509.07755, 2015.

[19] F. Chen, X. Liu, D. Proserpio, I. Troncoso, and F. Xiong. Studying product competition using representation learning. *CoRR*, abs/2005.10402, 2020.

[20] B. Danushka and O. James. A survey on word meta-embedding learning. *Proceedings of the 31st International Joint Conference on Artificial Intelligence*, 2022.

[21] A. Deshpande and M. Kumar. *Artificial Intelligence for Big Data: Complete Guide to Automating Big Data Solutions Using Artificial Intelligence Techniques*. Packt Publishing, 2018.

[22] J. Devlin, M. Chang, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[23] B. Dzimtry, C. Kyunghyun, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR*, 2014.

[24] M. Faruqui, Y. Tsvetkov, P. Rastogi, and C. Dyer. Problems with evaluation of word embeddings using word similarity tasks. In *Proceedings of the 1st Workshop on*

Evaluating Vector-Space Representations for NLP. Association for Computational Linguistics, 2016.

[25] U. Fayyad, C. Reina, and P. S. Bradley. Initialization of iterative refinement clustering algorithms. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, page 194–198. AAAI Press, 1998.

[26] J. R. Firth. A synopsis of linguistic theory 1930-55. *Studies in Linguistic Analysis (special volume of the Philological Society)*, 1952-59, 1957.

[27] V. Flavian, S. Elena, and C. Alexis. Meta-prod2vec - product embeddings using side-information for recommendation. *CoRR*, abs/1607.07326, 2016.

[28] R. M. Gray. *Entropy and Information Theory*. Springer Publishing Company, Incorporated, 2nd edition, 2011.

[29] G. Hamerly and C. Elkan. Alternatives to the k-means algorithm that find better clusterings. In *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, page 600–607. Association for Computing Machinery, 2002.

[30] R. C. Harris, K. J. Millman, S. J. V. D. Walt, R. Gommers, Virtanen. P., D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, S. Picus, M. Hoyer, M. Kerkwijk, M. Brett, A. Haldane, J. F. D. Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 2020.

[31] G. Hinton and S. Roweis. Stochastic neighbor embedding. In *Advances in Neural Information Processing Systems*, volume 15. MIT Press, 2002.

[32] H. V. D. Hoef and J. Warrens, M. Understanding information theoretic measures for comparing clusterings. *Behaviormetrika*, 46, 2018.

[33] E. Hoffer and N. Ailon. Deep metric learning using triplet network. In *Similarity-Based Pattern Recognition*, pages 84–92. Springer International Publishing, 2015.

[34] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 2007.

[35] C. Janiesch, P. Zschech, and K. Heinrich. Machine learning and deep learning. *CoRR*, abs/2104.05314, 2021.

[36] D. Jarrett and M. V. D. Schaar. Target-embedding autoencoders for supervised representation learning. 2020.

[37] X. Jin and J. Han. *Partitional Clustering*, pages 766–766. Springer US, 2010.

[38] C. Joshua and B. Danushka. Frustratingly easy meta-embedding – computing meta-embeddings by averaging source word embeddings. *CoRR*, abs/1804.05262, 2018.

[39] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR*, 2015.

[40] Y. LeCun, , Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436–44, 2015.

[41] Q. Lee and T. Mikolov. Distributed representations of sentences and documents. *31st International Conference on Machine Learning, ICML*, 4, 2014.

[42] J. Li, C. Xiong, and S. Hoi. Learning from noisy data with robust representation learning, 2021.

[43] Z. Li, M. Zhang, W. Che, T. Liu, and W. Chen. Joint optimization for chinese pos tagging and dependency parsing. *Audio, Speech, and Language Processing, IEEE/ACM Transactions on*, 22:274–286, 2014.

[44] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[45] L. V. D. Matten and G. Hinton. Viualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[46] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *1st International Conference on Learning Representations, ICLR*, 2013.

[47] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013.

[48] U. Naseem, I. Razzak, S. K. Khan, and M. Prasad. A comprehensive survey on word representation models: From classical to state-of-the-art word representation language models. *CoRR*, abs/2010.15036, 2020.

[49] N. Nayak, G. Angeli, and C. Manning. Evaluating word embeddings using a representative suite of practical tasks. In *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP*, pages 19–23. Association for Computational Linguistics, 2016.

[50] Z. Nazari, M. Nazari, M. S. S. Danish, and D. Kang. Evaluation of class noise impact on performance of machine learning algorithms. 08 2018.

[51] A. Neelakantan, T. Xu, R. Puri, A. Radford, J. M. Han, J. Tworek, Q. Yuan, N. Tezak, J. W. Kim, C. Hallacy, J. Heidecke, P. Shyam, B. Power, T. E. Nekoul, G. Sastry, G. Krueger, D. Schnurr, F. P. Such, K. Hsu, M. Thompson, T. Khan, T. Sherbakov, J. Jang, P. Welinder, and L. Weng. Text and code embeddings by contrastive pre-training. *CoRR*, abs/2201.10005, 2022.

[52] A. Nemec and R. Brinkhurst. The fowlkes–mallows statistic and the comparison of two independently determined dendrograms. *Canadian Journal of Fisheries and Aquatic Sciences*, 45:971–975, 04 2011.

[53] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *CoRR*, abs/1811.03378, 2018.

[54] B. Paaßen, C. Gallicchio, A. Micheli, and A. Sperduti. Embeddings and representation learning for structured data. *CoRR*, abs/1905.06147, 2019.

[55] J. Palacio-Niño and F. Berzal. Evaluation metrics for unsupervised learning algorithms. *CoRR*, abs/1905.05667, 2019.

[56] The pandas development team. pandas-dev/pandas: Pandas, 2020.

[57] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 2011.

[58] J. Pennington, R. Socher, and C. Manning. GloVe: Global vectors for word representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 14:1532–1543, 2014.

[59] S. C. R and R. K. Dubey. Meta-embeddings for natural language inference and semantic similarity tasks. *CoRR*, abs/2012.00633, 2020.

[60] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[61] W. M. Rand. Objective criteria for the evaluation of clustering. 66:846–850, 1971.

[62] K. Ravi and V. Ravi. A survey on opinion mining and sentiment analysis: Tasks, approaches and applications. *Knowlede based systems*, 89:14–46, 2015.

[63] R. Řehůřek and P. Sojka. Software framework for topic modelling with large corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50. ELRA, 2010.

[64] W. Ren, Y. Miche, I. Oliver, S. Holtmanns, K. Bjork, and A. Lendasse. On distance mapping from non-euclidean spaces to euclidean spaces. In *Machine Learning and Knowledge Extraction*, pages 3–13. Springer International Publishing, 2017.

[65] E. Rendón, I. Abundez, A. Arizmendi, and E.M. Quiroz. Internal versus external cluster validation indexes. *International Journal of Computers and Communications*, 5:27–34, 01 2011.

[66] P. Ristoski, P. Petrovski, P. Mika, and H. Paulheim. A machine learning approach for product matching and categorization. *Semantic Web*, 9:707–728, 2018.

[67] A. Rogers and A. Drozd. Intrinsic evaluations of word embeddings: What can we do better? In *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP*, pages 36–42. Association for Computational Linguistics, 2016.

[68] A. Rosenberg and J. Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. pages 410–420, 2007.

[69] M. Sahlgren. The distributional hypothesis. *Italian Journal of Linguistics*, 20, 2008.

[70] C. Sammut and G. I. Webb. *Encyclopedia of Machine Learning*. 2010.

[71] T. Schnabel, I. Labutov, D. Mimno, and T. Joachims. Evaluation methods for unsupervised word embeddings. 2015.

[72] R. Socher, M. Mohammadi, and R. Mundra. Cs 224d: Deep learning for nlp. 2015.

[73] S. Srinivasan, K. Ramamritham, and A. Kumar. Inducing conceptual embedding spaces from wikipedia. International World Wide Web Conferences Steering Committee, 2017.

[74] B. M. Sundheim. Overview of the fourth message understanding evaluation and conference. Association for Computational Linguistics, 1992.

[75] B. Szalkai. Generalizing k-means for an arbitrary distance matrix. *CoRR*, abs/1303.6001, 2013.

[76] M. Tomas, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *Proceedings of Workshop at ICLR*, 2013, 2013.

[77] L. Vasile, E. Smirnova, and A. Conneau. Meta-prod2vec - product embeddings using side-information for recommendation. *CoRR*, abs/1607.07326, 2016.

[78] S. Wagner and D. Wagner. Comparing clusterings - an overview. *Technical Report*, 2007.

[79] B. Wang, A. Wang, F. Chen, Y. Wang, and C. J. Kuo. Evaluating word embedding models: Methods and experimental results. *CoRR*, abs/1901.09785, 2019.

[80] K. Weinberger and L. Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10, 2009.

[81] Y. Wenpeng and S. Hinrich. Learning word meta-embeddings. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1351–1360. Association for Computational Linguistics, 2016.

[82] D. Xu, C. Ruan, E. Korpeoglu, S. Kumar, and K. Achan. Product knowledge graph embedding for e-commerce. *CoRR*, abs/1911.12481, 2019.

[83] D. Xu, C. Ruan, E. Körpeoglu, S. Kumar, and K. Achan. Theoretical understandings of product embedding for e-commerce machine learning. *CoRR*, abs/2102.12029, 2021.

[84] J. Xu, H. He, Xu. Sun, X. Ren, and S. Li. Cross-domain and semisupervised named entity recognition in chinese social media: A unified model. volume 26, pages 2142–2152, 2018.

[85] H. M. Zahera and M. A. Sherif. Probert: Product data classification with fine-tuning bert mode. *Proceedings of Mining the Web of HTML-embedded Product Data Workshop*, 2020.

[86] D. Zhang, Y. Li, and Z. Zhang. Deep metric learning with spherical embedding. *CoRR*, abs/2011.02785, 2020.

[87] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. page 103–114. Association for Computing Machinery, 1996.

Appendices

A Record Similarity Without Triplet Loss

Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	
Product Name	Similarity
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.896
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm	0.792
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.771
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.735
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.718
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.712
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.688
Gewebeband ext.Power 56348 schwarz L.10m B.48mm Rl.TESA	0.682
Gewebeband extra Power® 56348 schwarz Länge 10 m Breite 48 mm Rolle TESA	0.616
Tesa Folienband extra Power Universal 50 mm x 10 m schwarz	0.582

Table A.1: Top 10 Similar products to a given product by using cosine similarity on Averaged meta-embeddings wihtout triplet loss

Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	
Product Name	Similarity
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.893
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm	0.782
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.773
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.714
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.709
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.694
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.692
Gewebeband ext.Power 56348 schwarz L.10m B.48mm Rl.TESA	0.660
Gewebeband extra Power® 56348 schwarz Länge 10 m Breite 48 mm Rolle TESA	0.583
Tesa Folienband extra Power Universal 50 mm x 10 m schwarz	0.561

Table A.2: Top 10 Similar products to a given product by using cosine similarity on Concatenated meta-embeddings without triplet loss

Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	
Product Name	Similarity
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.6621
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.6619
tesa Folienband extra Power Universal, 50 mm x 10 m, schwarz (8756348)	0.66
tesa Reparaturband extra Power, schwarz, 10m x 50mm	0.6595
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.6594
Gewebeband extra Power® 56348 schwarz Länge 10 m Breite 48 mm Rolle TESA	0.6592
Tesa Extra Power Universal schwarz 48 mm x 10 m Gewebeklebeband	0.6574
tesa Reparaturband schwarz 10mx50mm Power Perfect, extra Power	0.6553
tesa extra Power Universal 10m 50mm schwarz	0.655
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.654

Table A.3: Top 10 Similar products to a given product by using euclidean distance on Averaged meta-embeddings without triplet loss

Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	
Product Name	Similarity
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.968
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.955
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.948
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.947
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.947
Gewebeband ext.Power® 56348 schwarz L.10m B.48mm Rl.TESA	0.946
Gewebeband ext.Power 56348 schwarz L.10m B.48mm Rl.TESA	0.9455
Tesa Folienband extra Power Universal 50 mm x 10 m schwarz	0.940
Tesa Extra Power Universal schwarz 48 mm x 10 m Gewebeklebeband	0.933
Spannungsprüfer DUSPOL digital Benning	0.9327

Table A.4: Top 10 Similar products to a given product by using euclidean similarity on Concatenated meta-embeddings without triplet loss

B Embedding Visualization Without Triplet Loss

Figure B.2: Visualization of Product Name from concatenated meta-embeddings without triplet loss using t-sne

Figure B.3: Visualization of Manufacturer Name from averaged meta-embeddings without triplet loss using t-sne

Figure B.4: Visualization of Product Name from averaged meta-embeddings without triplet loss using t-sne

Erklärung

Hiermit erkläre ich, dass ich meine Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die Arbeit noch nicht anderweitig für Prüfungszwecke vorgelegt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Mittweida, 31. August 2022