Link Prediction for Free-Format Text

Majumdar, Debajyoti

Award date:
2020

Department of Mathematics and Computer Science
Data Mining Research Group

Link Prediction for Free-Format Text

Master Thesis

by
Debajyoti Majumdar
1295071

Supervisors:
Dr. Vlado Menkovski
Dr. Sibylle Hess
Dr. Dmitri Jarnikov
Dr. Dirk Fahland

October 29, 2020

Acknowledgements

This Master Thesis is the final project of my master studies in Data Science in Engineering in the department of Mathematics and Computer Science. I want to express my gratitude to everyone involved in this thesis. First of all, I am grateful to my external supervisor, Dr.Dmitri Jarnikov, for giving me the opportunity to work at Prosus for my master thesis. His feedback has helped me throughout this project and in overcoming roadblocks that I faced during this period. Second of all, I would like to thank my university supervisor, Dr.Sibylle Hess, for her constant support and guidance. Her knowledge has been instrumental in making the thesis better in every sense. Finally, I would like to thank my committee members Dr.Vlado Menkovski and Dr.Dirk Fahland for taking the time to ensure fair and just evaluation of the thesis. I would also like to thank all the people who have indirectly helped me finish the thesis. I want to thank my family for supporting me throughout my master's program and staying strong despite the challenges we all have faced collectively.

Abstract

Online classifieds are advertisements posted online by private citizens for mostly used goods. Customers often have specific requirements for the products they are trying to buy online, for example, a certain RAM specification for a laptop or the brand of a car. Such specifications of the products are called attributes. Classifieds consist of unstructured product descriptions leading to poor search results and recommendations for the interested user. Attribute-level information enables a product description to be represented in a structured format. We propose a semi-supervised solution for converting the unstructured classifieds product descriptions to a structured format by representing the semantic meaning of words by their relationships. Link Prediction has not been explored for predicting attributes from product descriptions to the best of our knowledge. We present a novel approach to constructing a knowledge graph using labeled and/or unlabeled product descriptions. Initial experiments conducted on a public annotated data-set show that we obtain state-of-the-art results, compared to traditional classification models. Next, we extend the link prediction approach to an unlabeled data-set by constructing a knowledge graph using a heuristics-based method. Alternative strategies are proposed to improve the baseline approach's performance by modifying the model and input data. We show that we achieved an increase in performance compared to the baseline approach by applying the proposed strategies. The key contribution is towards having a system contributing primarily to produce relevant search results and recommendations to the user.

Contents

List of Figures

List of Tables

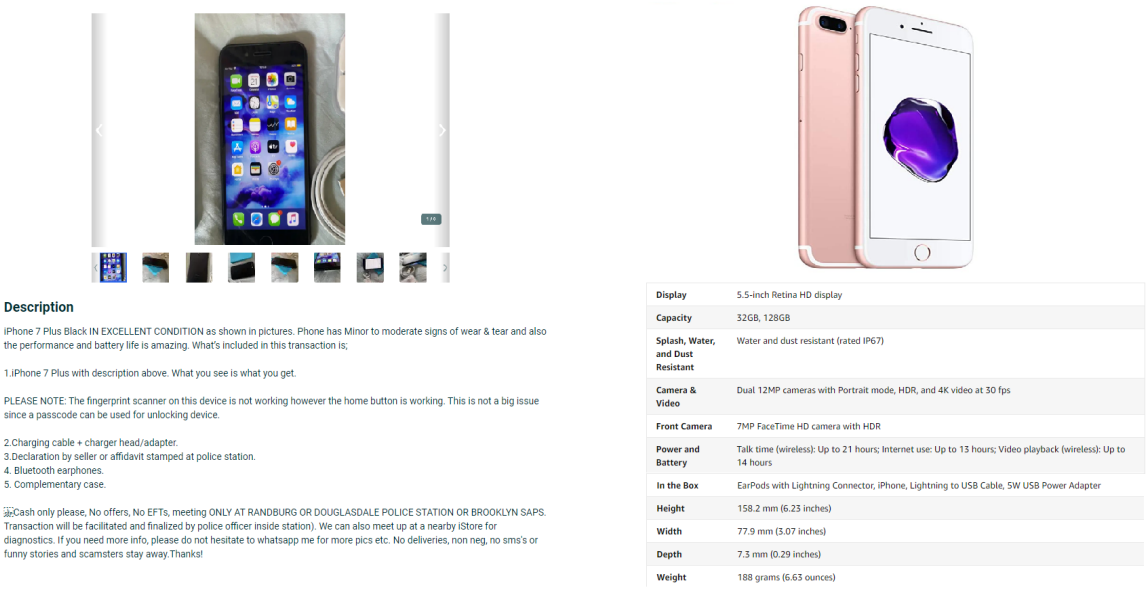
Chapter 1

Introduction

Identifying the semantics of words automatically is relevant for various application areas such as clustering [1], topic modeling [2]. Here, we want to model the relationship between the words, its context and broader meaning. To conclude if the word "apple" refers to a fruit or a brand is decided by the context. The relationships between words help in understanding text and deciphering information. Explicitly creating relationships between words enables a system to interpret language and also to define new relationships which the system has not seen before. One of the benefits of extraction of relationship is to convert unstructured text to a structured format, which is a major challenge in the case of online classifieds. Online classifieds are advertisements posted on online platforms by private citizens for mostly used goods. Typical examples of such platforms are OLX, and Ebay. Classifieds descriptions lack structure when compared to descriptions from e-commerce marketplaces like Amazon, Flipkart, Best Buy, etc. We propose an semi-supervised solution for converting the unstructured classifieds product descriptions to a structured format by representing the semantic meaning of words by their relationships.

1.1 Classifieds are an Information Desert

Customers often have specific requirements for the products they're trying to buy online, for example, a certain RAM specification for a laptop or the brand of a car. Such specifications of the products are called attributes. These attributes can take one or more values. For example, attribute colour, could be red, blue, or green. Similarly, a laptop can have 4GB, 8GB, or 16GB of RAM. In e-commerce marketplaces such as Amazon, Flipkart, merchants often invest time in creating detailed and structured product descriptions, leveraging information from the item's manufacturer. Whereas, in online classifieds such as OLX and Ebay, product descriptions are comparatively less structured. This is because sellers are more often consumers themselves, not professional businesses and because all products are in a sense unique (since they're secondhand) as each product description is written by a different user and each product is in a different condition compared to other products. Figure 1.1 shows distinction between product description from online classified and e-commerce marketplace. We can see the online classifieds have unstructured product descriptions compared to e-commerce marketplaces.



Description

iPhone 7 Plus Black IN EXCELLENT CONDITION as shown in pictures. Phone has Minor to moderate signs of wear & tear and also the performance and battery life is amazing. What's included in this transaction is;

1.iPhone 7 Plus with description above. What you see is what you get.

PLEASE NOTE: The fingerprint scanner on this device is not working however the home button is working. This is not a big issue since a passcode can be used for unlocking device.

2.Charging cable + charger head/adapter.
3.Declaration by seller or affidavit stamped at police station.
4. Bluetooth earphones.
5. Complementary case.

Cash only please, No offers, No EFTs, meeting ONLY AT RANDBURG OR DOUGLASDALE POLICE STATION OR BROOKLYN SAPS. Transaction will be facilitated and finalized by police officer inside station). We can also meet up at a nearby iStore for diagnostics. If you need more info, please do not hesitate to whatsapp me for more pics etc. No deliveries, non neg, no sms's or funny stories and scamsters stay away.Thanks!

Display	5.5-inch Retina HD display
Capacity	32GB, 128GB
Splash, Water, and Dust Resistant	Water and dust resistant (rated IP67)
Camera & Video	Dual 12MP cameras with Portrait mode, HDR, and 4K video at 30 fps
Front Camera	7MP FaceTime HD camera with HDR
Power and Battery	Talk time (wireless): Up to 21 hours; Internet use: Up to 13 hours; Video playback (wireless): Up to 14 hours
In the Box	EarPods with Lightning Connector, iPhone, Lightning to USB Cable, 5W USB Power Adapter
Height	158.2 mm (6.23 inches)
Width	77.9 mm (3.07 inches)
Depth	7.3 mm (0.29 inches)
Weight	188 grams (6.63 ounces)

Figure 1.1: Online Classified vs e-commerce marketplace product description. Left: Online Classified description in unstructured format, Right: e-commerce marketplace description in structured tabular format.

Extracting the attributes and values in an automated way would to provide structured information to the buyers without burdening the seller with strict requirements such as asking the user to fill in specific details about the product. In addition, having more structured product descriptions enables the platform to provide:

- **More relevant search results and recommendations:** having an attribute based product description enables the search system to retrieve items based on attribute level information. It would also enable the platform to define search filters based on attributes, allowing the user to find a product of their choice quickly. Also, more relevant products can be recommended to users based on the history of products explored and the attributes associated with them. For example, providing search results based on the brand or color of a product.

- **Better comparison between products:** an attribute based product description would help define a taxonomy for products that would allow comparison with other related products and improve sales forecasts.

Having structured information of a product enable buyers to make an informed choice and avoid the disappointment of receiving a product that does not live up to their expectations.

1.2 Problem Definition

Our goal is to parse the product descriptions from online classifieds and identify the attribute associated with each target word. Figure 1.2 is a high-level diagram providing input and

expected output. The input to the system is a product description and the output is a collection of attributes and values.

Figure 1.2: High-level diagram: Input to the system is a product description and output is pairs of attributes and values. Attributes are shown in dark blue, and values in red.

Traditionally the approach of extracting keywords from text falls under the umbrella of Information Extraction (IE) and, in particular, an instance of Named Entity Recognition (NER) [3]. Supervised machine learning-based systems have been the most successful in the NER task [4]. Prior work explained in Section 2.2.1 provides several supervised machine learning techniques for predicting attributes from product descriptions. However, these techniques require correct annotations of product descriptions in large quantities for training. By annotation we mean that the words in a product description are tagged with attributes. The task of manually annotating text is labour-intensive and needs domain expertise as one needs to know which attribute to tag for the values. One possible approach to deal with this problem is to label small amounts of data and apply Transfer Learning [5]. However, our task is specific to predicting attributes based values from classifieds. Existing pretrained NER models identify generic entity types (person, date, location, geopolitical entity, organisation, etc.); hence Transfer Learning is not suitable.

Unsupervised machine learning techniques for extracting attributes from text has received far less attention as compared to supervised techniques. The unsupervised approaches do not require any manually annotated data unlike supervised approaches. The unsupervised techniques assume that product descriptions follow a particular structure where values are mentioned after attributes. We explain these techniques in more detail in Section 2.2.2.

In Section 2.2.3 we explore the semi-supervised approaches that are applicable for predicting attributes from product descriptions. Knowledge graphs model information in the form of entities and relationships between them. In this case, entities refer to target words, context, attributes and relationships refer to the links between these entities. In Figure 1.3 we give an example of a knowledge graph where the word “Apple” is used in two different contexts. Suppose we know that when “Apple” is used in context with “iPhone” it refers to the Brand and when “Apple” is used in context with “Bananas” we know it refers to a Fruit. Given that apple is used in context with “Watch” can we predict if apple refers to a brand or fruit? Such a problem of predicting missing links is considered as a knowledge graph completion problem.

The knowledge graphs can be represented in the form of a tensor and Link Prediction [6] can be applied for predicting missing links. To the best of our knowledge no study has been conducted so far for applying Link Prediction models to predict attributes from textual descriptions. Link Prediction refers to the task of predicting the existence (or probability of correctness) of edges in the graph. One possible approach for knowledge graph completion is by applying Tensor Factorization [7]. Tensor Factorization models can capture semantic representation of words and the level of interaction between these words.

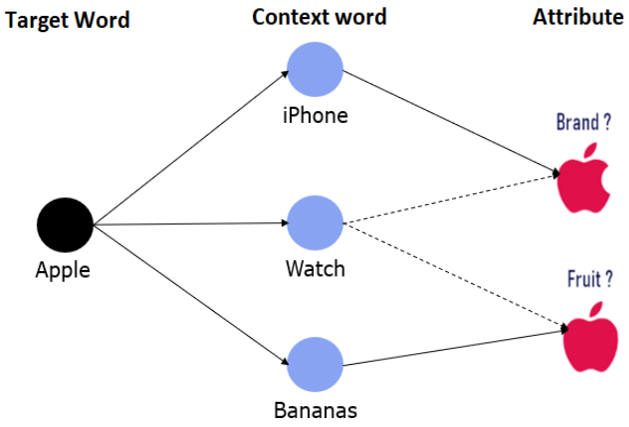


Figure 1.3: Example of a knowledge graph completion problem where the model has to predict whether Apple in context of Watch is a Brand or a Fruit. Bold lines refer to known links and dotted lines refer to missing links.

1.3 Research Questions

We break down our problem statement into the following high-level research questions:

- RQ1: Is knowledge graph completion a suitable model for predicting attributes from product descriptions? If yes, then what would be the best technique out of the existing approaches that can be used?
- RQ2: If knowledge graph completion model can be applied to our use-case, can we extend this approach to unlabeled data? How can we construct a knowledge graph from unlabeled data?
- RQ3: How does the model performance change when data volume is scaled up? We find the impact on the model's performance by increasing the volume of training data. This research question is relevant from the model deployment point of view.

In Section 3.2 we breakdown the above research problems into further sub-problems and give a detailed insight into the problems that try to solve.

1.4 Approach

To answer RQ1, we conducted experiments on public data-set and compare our model performance to existing solutions. For RQ2, we provide an automated solution for constructing knowledge graph from unlabeled data. By doing so, the data quality is lowered where most of the facts are either not accurate or not relevant. To handle this noisy data, we propose techniques for improving the performance of the model. To answer RQ3, we consider product descriptions from from multiple categories and illustrate the impact on the model performance by adding product descriptions from one category at a time.

1.5 Contributions

In this thesis, we propose an unsupervised solution for converting the unstructured classifieds product descriptions to a structured format, representing the semantic meaning of words by their relationships between them. In the task of achieving this goal we succeeded in making the following contributions:

- We provide a novel approach for constructing a knowledge graph from unstructured product descriptions.
- We provide a novel interpretable approach for combining the context words while making prediction for the target word.
- We show that our approach outperforms state-of-the-art solution on a public, annotated data-set.
- We define novel strategies for improving the performance of the system by modifications of the model and input data.

1.6 Outline

The outline of the rest of this Master Thesis report is as follows:

- Chapter 2 provides preliminary knowledge and prior work in the domain of Attribute Extraction from text and Link Prediction.
- Chapter 3 provides a deep-dive into the problem statement by breaking each research question down into further sub-questions.
- Chapter 4 consists of the methodologies implemented to answer the first research question.
- Chapter 5 contains the methodologies employed to answer the second research question.
- Chapter 6 provides the results associated with the research questions.
- Chapter 7 details our contributions, concluding remarks, and directions for future work.

Chapter 2

Background

2.1 Preliminaries

We use Tensor Factorization to obtain relationship information from natural language. In this section, we review the definitions and properties of tensors as far as they are relevant for the course of this thesis.

2.1.1 Introduction to Tensors

Tensors is an array of numbers, which typically come from a field (like $\mathbb{R}$). A tensor is typically represented by a three-dimensional array or a data cube, although it could also be more than three dimensions. Below we will go through some of the terminologies and operations associated with tensors.

- **Tensor order:** The order of a tensor is the number of its dimensions. Scalars can be interpreted as zero-order tensors, vectors as first-order tensors, and matrices as second-order tensors. Notation-wise, scalars are denoted by lower case letters $x \in \mathbb{R}$, vectors by bold letters $\boldsymbol{x} \in \mathbb{R}^{I_1}$, matrices by upper case letters $X \in \mathbb{R}^{I_1 \times I_2}$, and higher order tensors by upper case bold letters $\boldsymbol{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$. I_s denotes the number of elements in the respective dimension. Figure 2.1 illustrates the transition from scalars to tensors.

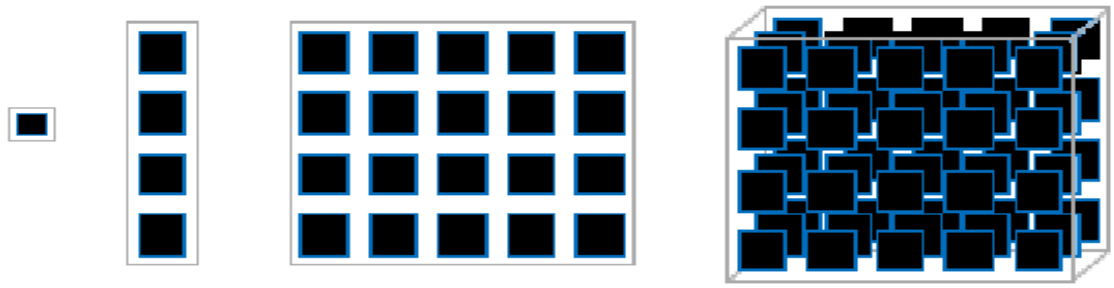


Figure 2.1: Visualisation of scalars, vectors, matrices and tensors respectively. $x \in \mathbb{R}, \mathbf{x} \in \mathbb{R}^4, X \in \mathbb{R}^{4\times5}, \boldsymbol{X} \in \mathbb{R}^{4\times5\times3}$.

- **Tensor Indexing:** We can create subarrays (or subfields) by fixing some of the given tensor's indices. Fibers are created when fixing all but one index. Slices (or slabs) are created when fixing all but two indices. For a third order tensor the fibers are given as $\boldsymbol{X}_{:jk} = \boldsymbol{x}_{jk}$ (column), $\boldsymbol{X}_{i:k}$ (row), and $\boldsymbol{X}_{ij:}$ (tube). The slices are given as $\boldsymbol{X}_{::k} = X_k$ (frontal), $\boldsymbol{X}_{:j:}$ (lateral), $\boldsymbol{X}_{i::}$ (horizontal). Figure 2.2 and Figure 2.3 depict the fibers and slices of third order tensor.

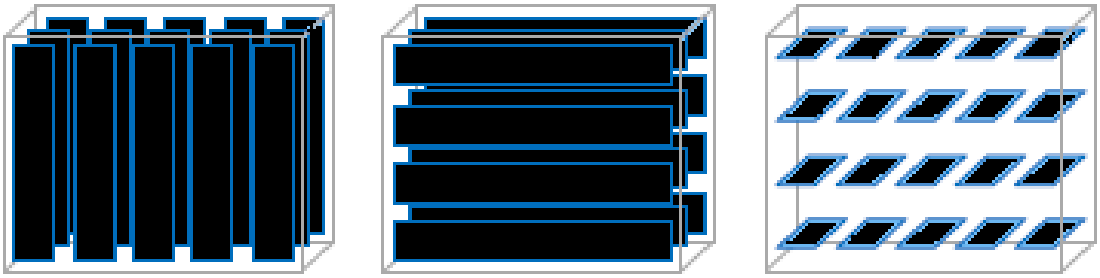


Figure 2.2: Column, row, and tube fibers of a third order tensor. Fibers are created when fixing all but one index.

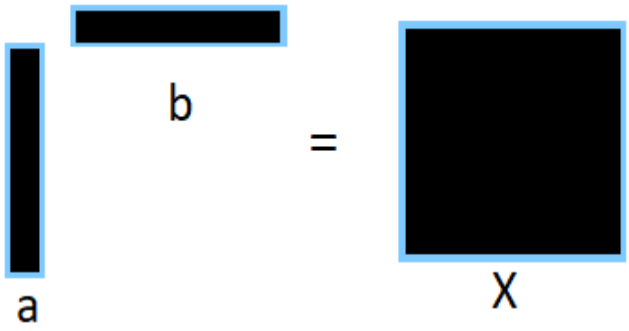


Figure 2.4: Outer product of two vectors results in a matrix. $X = \boldsymbol{a} \otimes \boldsymbol{b} = \boldsymbol{a}\boldsymbol{b}^T$

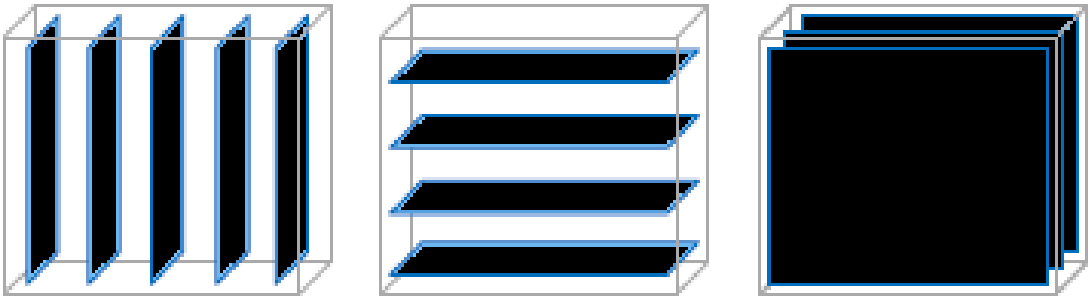


Figure 2.3: Lateral, horizontal and frontal slices of a third order tensor. Slices are created by fixing all but two indices.

- **Outer Product:** The vector outer product is defined as the product of the vector's elements. This operation is denoted by the $\otimes$ symbol. Figure 2.4 shows the illustration that outer product of 2 vectors results in a matrix. Equation (2.1) shows the vector outer product of two n-sized vectors $\boldsymbol{a}, \boldsymbol{b}$ is defined as follows and produces the matrix X:

$$X = \boldsymbol{a} \otimes \boldsymbol{b} = \boldsymbol{a}\boldsymbol{b}^T \tag{2.1}$$

  By extending the vector outer product concept to the general tensor product for N vectors, we can produce a tensor $\boldsymbol{X}$ as shown in Equation (2.2):

$$\boldsymbol{X} = \boldsymbol{a}^{(1)} \otimes \boldsymbol{a}^{(2)} \otimes \boldsymbol{a}^{(3)} \otimes ... \otimes \boldsymbol{a}^{(N)} \tag{2.2}$$

- **Tensor Re-orderings-Matricization**: Matricization is the operation that reorders a tensor into a matrix. We will look into the mode-n matricization of a tensor. This operation is also known as unfolding or flattening of a tensor. The mode-n matricization

of a tensor $\boldsymbol{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3 \times \ldots \times I_N}$ is denoted by $\boldsymbol{X}_{(n)} \in \mathbb{R}^{I_n \times (I_1 \cdot \ldots \cdot I_N)}$. Below we provide an example for matricization. Let $\boldsymbol{X}$ be a tensor with the following frontal slices:

$$\boldsymbol{X}_{::1} = X_1 = \begin{bmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{bmatrix} \qquad \boldsymbol{X}_{::2} = X_2 = \begin{bmatrix} 13 & 16 & 19 & 22 \\ 14 & 17 & 20 & 23 \\ 15 & 18 & 21 & 24 \end{bmatrix}$$

Then the three mode matricizations are:

$$X_{(1)} = \begin{bmatrix} 1 & 4 & 7 & 10 & 13 & 16 & 19 & 22 \\ 2 & 5 & 8 & 11 & 14 & 17 & 20 & 23 \\ 3 & 6 & 9 & 12 & 15 & 18 & 21 & 24 \end{bmatrix}$$

$$X_{(2)} = \begin{bmatrix} 1 & 2 & 3 & 13 & 14 & 15 \\ 4 & 5 & 6 & 16 & 17 & 18 \\ 7 & 8 & 9 & 19 & 20 & 21 \\ 10 & 11 & 12 & 22 & 23 & 24 \end{bmatrix}$$

$$X_{(3)} = \begin{bmatrix} 1 & 2 & 3 & 4 & .. & 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 & ... & 21 & 22 & 23 & 24 \end{bmatrix}$$

This tensor operation is useful while understanding the Tensor Factorization model that we describe in Section 4.3.

2.2 Related Work

In this section, we discuss the supervised, unsupervised and semi-supervised approaches for attribute extraction from e-commerce product descriptions.

2.2.1 Supervised Approaches

Most research on attribute extraction from product descriptions has been carried out in the field of supervised learning. The authors in [8] assume retail descriptions are represented as attribute-value pairs. They present techniques for extracting attribute-value pairs for a small set of attributes or labels like age group, degree of brand appeal, and price point. They apply the Naive Bayes classifier [9] for each attribute. This approach is not scalable when dealing with large volumes of data that consist of a large number of attributes as we have to have a different classifier for each attribute leading to issues leading to difficulties in maintenance. The work presented in [10] uses supervised approach for extracting attribute-values from product descriptions related to electronic products using features that are manually defined. The author compares three classification methods in this paper. The author trains a Support Vector Machine (SVM)[11], Gradient Boosting Trees (GBT)[12] and Conditional Random Field (CRF) [13] for classifying each token present in the product listing description.

In [14] the authors presents a technique for attribute-value extraction using a sequence labeling classifier, using Structured Perceptron and Conditional Random Fields (CRF) similar to [15]. They generate a set of feature functions for extracting the brand from the listings. The authors also present a normalization scheme to find the attribute despite the value having multiple variations. A list of key-value pairs is maintained by a team of analysts, which

contains the mapping of variants of value and their corresponding normalized value. This list is maintained by a team of analysts and is, therefore, not a feasible approach for our use case.

The author in [16] presents an approach of Attribute extraction on listings by utilizing Transfer learning. Transfer learning is a general class of strategies where knowledge learned in one domain (for example, a certain attribute) can assist learning in another domain. In this case, the author already has a labeled data-set to begin with and later extends the model to the remaining unlabeled data using Transfer learning. In our case we do not have any labeled data-set to begin with hence this approach is not suitable for us. In [17] the authors present a novel approach of incorporating active learning [18] along with the sequence labeling approach. The authors tackle the Open World Assumption (OWA) problem, i.e., they extract attribute-value pairs that the model has not encountered before. Their model can achieve state-of-the-art performance for discovering attributes that it has not seen before. However, for initial training, the model requires data with labels, which is done manually in this case.

Together these studies provide important insights into the challenges that the authors face while extracting attributes from product descriptions. However, the approaches mentioned thus far remain narrow in focus dealing only with manually labeled data. In our case we are dealing with large volumes of unlabeled data which makes manual labeling labor-intensive and time-consuming. This motivates us to explore some of the unsupervised techniques.

2.2.2 Unsupervised Approaches

Unsupervised approaches for extracting attributes and values have received much less attention compared to supervised approaches. The authors in [15] automatically extract popular product attributes from a collection of customer reviews. Latent Dirichlet Allocation (LDA) [2] is employed to discover latent concepts, which essentially refer to the popular features of the products. Conditional Random Fields (CRF)[13] is used for extracting these popular attributes from product descriptions. The drawback of this approach is that the quality of the popular attributes depends on consumer reviews' quality. Each product category would have its own set of popular attributes; hence a different topic model would be required for extracting popular attributes from customer reviews. The authors in [19] provide a novel approach of attribute extraction from text by constructing a graph using the word co-occurrence statistics. Their approach is based on the hypothesis that similar products are described by a similar set of attributes; hence the terms corresponding to attributes would be more frequent than the remaining terms. However, this method can capture only the frequent attributes and tends to miss the attributes that rarely occur in the description. The approach assumes that attribute-value pairs occur together in the description, and a value follows the attribute. However, this might not happen every time. The attribute is not mentioned most of the time in the product description, especially in second-hand product descriptions.

In [20] the authors present a clustering-based approach for identifying attribute-values. They extract noun phrases of n-grams from product descriptions and apply Dice's coefficient [21] as a distance measure for the Group Average Agglomerative Clustering(GAAC) [22] algorithm for computing the clusters. This method's drawback is that both the attribute and its value should occur in the product description, which does not often happen with second-hand

product descriptions.

When exploring unsupervised techniques, we notice that the data is required to follow a particular structure, such as containing the attribute and values in the product description. Although, this is not the case with online classifieds where product descriptions are written in unstructured format. We also realize that supervised techniques achieve better performance, compared to unsupervised techniques. The problem with supervised techniques is that they require labeled data. We want to benefit from the better performance of supervised approaches and also want the increased applicability of unsupervised approach which motivated us to explore semi-supervised approaches for our use-case.

2.2.3 Semi-supervised Approaches

In [23] the authors propose a semi-supervised bootstrapped NER approach for extracting attribute-values from Ebay's clothing and shoe categories and develop an attribute extraction system for four attribute types. They train a sequential classifier and evaluate the extraction performance on a set of manually annotated product descriptions/listings. Their proposed bootstrapped algorithm can identify new brand names corresponding to spelling variants or typographical errors of the known brand names in initial list of attributes. This approach would work for our case, if we had an initial list of attributes to identify.

Extracting attributes from text could be modeled as extracting relationships between words and identifying which relationships are true or false, where true relations indicates words having a semantic relationship. This is where Statistical Relational Learning (SRL) comes into picture. Statistical Relational Learning is a subdiscipline in machine learning that is concerned with knowledge graph representations. Thus the data is represented as a knowledge graph, consisting of nodes (entities) and labeled edges (relationships between entities). The main goals of SRL include the prediction of missing edges, prediction of properties of nodes, and clustering nodes based on their connectivity patterns. In the context of knowledge graphs, link prediction is also referred to as knowledge graph completion. It has been shown that relational models that take the relationships of entities into account can significantly outperform non-relational machine learning methods for this task [24][25]. This motivates us to explore the knowledge graph completion models in more detail. The authors in [26] and [27] utilise pre-trained language models such as BERT [28] for knowledge graph completion. They treat the triples in the knowledge graph as textual sequences and translate the knowledge graph completion problem into a sequence classification problem [29]. This approach provides contextualized representation for entities and relations between them. Although these models provide promising results across the standard knowledge graph data-sets, they do not outperform the pre-existing link prediction models.

Numerous matrix factorization approaches to link prediction have been proposed. Matrix factorization [30] is a class of collaborative filtering models [31]. Specifically, the model factorizes a user-item interaction matrix (e.g., rating matrix) into the product of two lower-rank matrices, capturing the low-rank structure of the user-item interactions. Many approaches to link prediction have been based on different factorizations of a binary tensor representation of the training triples. Tensor factorization models have been used for representing multi-relational data. For example, the interaction between users in social networks can be considered multi-

relational data, and applying tensor factorization algorithms allows the determination of the inter-dependencies occurring on multiple levels simultaneously. This motivates us to explore some of the tensor factorization models.

RESCAL [32] optimizes a scoring function containing a bilinear product between subject and object entity vectors and a full rank relation matrix. Although a very expressive and powerful model, RESCAL is prone to over-fitting due to its large number of parameters, which increases quadratically in the embedding dimension with the number of relations in a knowledge graph.

SimplE [33] is based on Canonical Polyadic (CP) decomposition [7] in which subject and object entity embeddings for the same entity are independent SimplE's scoring function alters CP to make subject and object entity embedding vectors dependent on each other by computing the average of two terms, first of which is a bilinear product of the subject entity head embedding, relation embedding, and object entity tail embedding, and the second is a bilinear product of the object entity head embedding, inverse relation embedding, and subject entity tail embedding.

The Tucker decomposition [34] decomposes a tensor into a core tensor and multiple matrices. Each factor matrix contains embeddings of subject, relation, and objects of the triple, and the core tensor contains the weights for the interactions between the components of the triple. Tucker is also known for its fully expressive nature which means that, for any ground truth over all entities and relations, there exist entity and relation embeddings that accurately identify true relations between entities. In [35] the authors show that Tucker decomposition, despite being a linear model for link prediction achieves state-of-the-art results across all standard data-sets. Considering all the approaches that we have explored so far, link prediction models have not been used for predicting attributes in product descriptions. Given that Tucker achieves state-of-the-art results across the standard knowledge graph data-sets, we choose this model to fit the data. In Chapter 4 we explain the Tucker decomposition in more details and its applicability for knowledge graph completion.

Chapter 3

Problem Exposition

In this chapter we provide details regarding the data used to answer our research questions and provides insights into the challenges we face while answering each research question.

3.1 Data Understanding

For performing our experiments and answering the research questions, we use the following data-sets:

- Labeled data-set: Public data-set [36] with labeled data consisting of electronics e-commerce product descriptions. It is composed of 941 product descriptions, where some words from each product description are manually labeled (tagged) by experts. The product descriptions are in the English language. Figure 3.1 is an illustration of a labeled product description. We use the Best Buy data-set for answering RQ1 which is to verify if knowledge graph completion is a suitable model for predicting attributes from product descriptions.

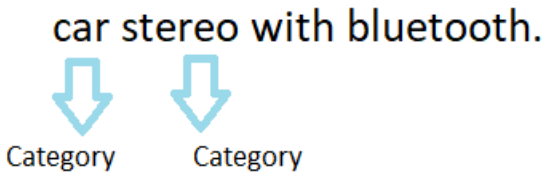


Figure 3.1: Example of a labeled product description from Best Buy data-set. Car and stereo are the values and Category is the attribute/label for these words.

- Unlabeled data-set: OLX [37] has provided us access to their listings with each listing consisting of a title, description, and images of a product. Figure 3.2 shows a listing of a binocular. The listings are unlabeled, i.e., the product descriptions do not have any attributes/labels associated with the words. For our experiments, we use the

title and description from every listing. Every product description in the OLX data-set has category information associated with it. For example: the listing in Figure 3.2 is associated to category *Consumer electronics/Photographic equipment/Binoculars and telescopes/Binoculars*. In RQ2 we pose the question of extending the knowledge graph completion model to unlabeled data. For our experiments we consider product descriptions from three categories from the domain of Consumer Electronics. Table 3.1 contains information about the number of listings and category information of the product descriptions used to answer RQ2.

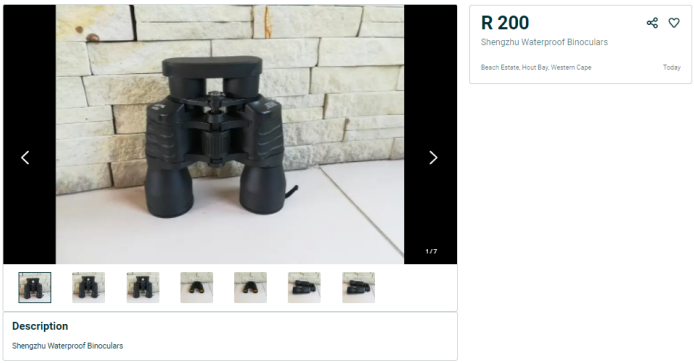


Figure 3.2: Example of a listing from OLX data-set. We use the title and description from the listing for our experiments.

Category Information	# of listings
Ipod nano player	429
Meizu Phone	283
Binoculars and telescopes	255
Total	**967**

Table 3.1: OLX data-set information for RQ2. These categories are under the domain of Consumer Electronics.

3.2 Detailed Research Questions

In Section 1.3, we pose the high-level research questions that we answer in this thesis. Here we break our research questions down into further sub-questions. Each sub-question is represented as RQi_j where i_j refers to the j^{th} sub-problem in the i^{th} research question (RQ).

- RQ1: in RQ1, we propose using knowledge graph completion for predicting attributes from product descriptions. However, we cannot apply the knowledge graph completion model directly to the data. We break the research question down into the following sub-questions:
  - $RQ1_1$: how to construct a knowledge graph using product descriptions and their corresponding labels? The data needs to be provided in a triple format to apply

knowledge graph completion model for predicting missing links. We present an approach for constructing the knowledge graph in Section 4.1.

- $RQ1_2$: how can we apply the Tensor Factorization Tucker decomposition technique to the knowledge graph for predicting attributes from product descriptions? This research question was from an implementation point of view. Can we use the already existing code-base for Tucker decomposition and begin our experiments? The answer to this sub-question is provided in Section 4.3.

- $RQ1_3$: how can we use the surrounding/context words' knowledge while making predictions for a target word? In Section 4.4 we provide a novel approach for this research question.

- RQ2: By answering RQ1, we show that we can extract attributes from product descriptions by having labeled data. In RQ2, we want to apply the knowledge graph completion technique to unlabeled data. How can we construct a knowledge graph from unlabeled data? Can we use the same link prediction model used in RQ1? To answer this research question, we break our research question down into the following sub-questions:

  - $RQ2_1$: how to construct a knowledge graph from unlabeled data? Can we use an external database to create a knowledge graph that we can apply to our use-case? We answer this research question in Section 5.2.

  - $RQ2_2$: can we use the same knowledge graph completion technique as RQ1 to predict attributes from product descriptions, or do we need to provide alternative strategies for improving the model and input data? We provide an answer to this sub-question in Section 5.5.

- RQ3: how does the model performance change when data is scaled up? We find the impact on the model's performance by increasing the volume of training data. This research question is relevant from the model deployment point of view. Is it better to use one model to predict attributes from all categories of product descriptions or use one model for each category of product descriptions? We provide the experiment and results associated with this particular research question in Section 6.5.

3.3 High Level Approach

In Figure 3.3, we present the pipeline involved in extracting attributes from product descriptions. Product descriptions are not by default in the triple format; hence, we need to figure out how to represent product descriptions as a knowledge graph. We provide a solution to this particular sub-problem in Section 4.1. Based on the literature survey, we decide to use Tucker decomposition[34] for Tensor Factorization. Our goal is to extract attributes from product descriptions. For obtaining prediction for a target word, we want to use the context. We propose a novel approach for predicting an attribute for a target word by considering the context words in Section 4.4. The author in [10] applies different classification methods on the Best Buy public data-set for extracting attributes. We compare our model performance to their approach in Section 6.3.

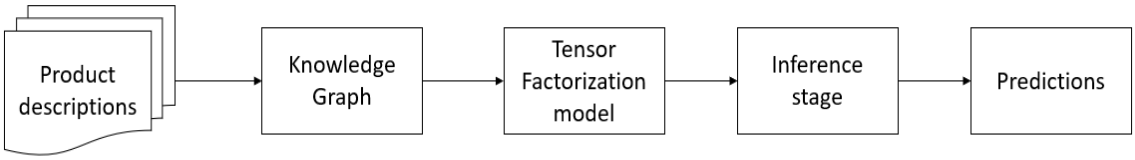


Figure 3.3: High level diagram of the pipeline for predicting attributes from product descriptions. We follow these steps while answering the research questions.

The pipeline remains the same while answering RQ2; however, the technical approach is different. Since we deal with unlabeled data in RQ2, in Section 5.2, we provide a heuristic rule-based solution to construct a knowledge graph from unlabeled product descriptions. We apply the same Tensor Factorization technique as RQ1 to develop a baseline model for our experiments. Based on the experiments' observations and results, we propose numerous strategies for improving the system's performance. Details about these techniques are mentioned in Section 5.5.

Chapter 4

Link Prediction with Labeled Data

In this chapter we answer RQ1 with each section tackling the sub-questions mentioned in Section 3.2.

- In Section 4.1, we present an approach for representing the product descriptions as a knowledge graph.
- Section 4.2 provides an approach for representing the knowledge graph as a binary tensor.
- Section 4.3 gives insight into the Tucker Decomposition model applied to model the semantic relationship between the words of product descriptions.
- Lastly, we tackle the problem of using the context information while predicting an attribute for a target word in Section 4.4.

4.1 Product Descriptions as a Knowledge Graph

The first task is to express the product descriptions as a knowledge graph. We associate the target word and its context to the label. In the knowledge graph, each connection can be represented as a triple. Each triple consists of $(target\_word, context\_word, label)$. In Figure 4.1 we show the target word, context words and labels in a product description. Target word refers to the word which has an attribute/label, context words refers to the words surrounding the target word and label refers to the attribute.

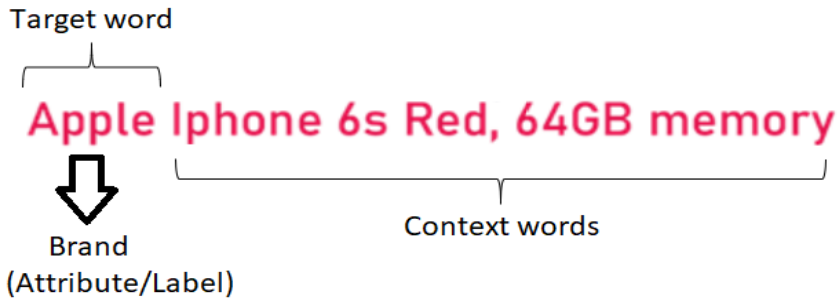


Figure 4.1: Target word, context words and attribute/label in a product description. Target word has an attribute/label, context words refers to the words surrounding the target word and Brand refers to the attribute.

We formalize the knowledge graph creation approach by the following definition.

Definition: Let S be a sentence in the product description consisting of words $(w_1, w_2, ..., w_n)$ where $w_i \in S$ be the tokenized version of S. Given an attribute/label α associated to a target word w_i, the triples are of the format, (w_i, w_j, α) where w_i is the target word, w_j are the context words in the sentence S, where $1 \leq j \leq n$.

Figure 4.2 illustrates the knowledge graph constructed using two product descriptions. Consider the first product description in the figure below: *"car stereo with bluetooth"*. There are *two target words* associated to *label:"Category"*. In this case, $S =$ *"car stereo with bluetooth"*, $(w_1, w_2, ..., w_n) = (car, stereo, with, bluetooth)$, and (α_1, α_2) are the labels $(Category)$. The target words are w_1, w_2, i.e., $car, stereo$. Context words correspond to the words around the target word, i.e., $(w_1, w_2, ..., w_n)$. Similarly, for the second product description, $S =$ *"bmw car with ac"*, $(w_1, w_2, ..., w_n) = (bmw, car, with, ac)$, and (α_1, α_2) are the labels $(Brand, Category)$. We want to highlight that the target word is also considered as a context word while creating the knowledge graph.

Nodes consisting of pairs of the target word and context words are constructed. We call these nodes as the source nodes. Each source node consists of $(target\_word, context\_word)$ pair. Each attribute/label corresponds to a label node. The source nodes are connected to label nodes if a target word is associated with a label. In this way, we create a knowledge graph comprising the target word, context, and label. The arrows indicate the connection from the source node to the target node. The knowledge graph can also be represented in a triple format shown in Table 4.1.

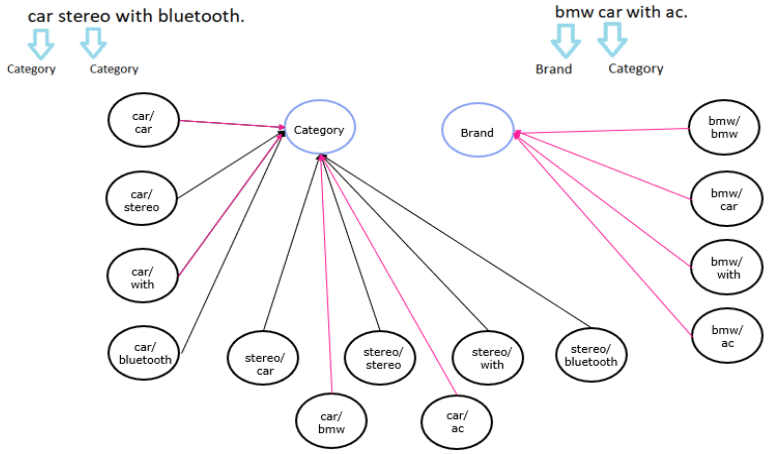


Figure 4.2: Knowledge Graph representation of product descriptions. The connections in black and red are formed from first and second product description respectively.

target word	*context word*	*label*
(car,	car,	Category)
(car,	stereo,	Category)
(car,	with,	Category)
(car,	bluetooth,	Category)
(stereo,	car,	Category)
(...,	...,	...)

Table 4.1: Knowledge Graph representation as a list of triples. The triples are in the format (target_word, context_word, label).

4.2 Constructing the Knowledge Graph Tensor

Section 4.1 helps us understand the process of transforming a product description and its corresponding labels to a knowledge graph and represent it in a triple format. To apply knowledge graph completion/link prediction models, the knowledge graph must be represented in a suitable format for the model to process. We represent the triples in a third order or three-dimensional tensor to apply the Tensor Factorization technique for knowledge graph completion. The tensor has three axes associated with it that correspond to :

- Target word
- Context word
- Label

In the tensor, the position corresponding to *(target_word, context_word, label)* is set as 1 if there is a connection between them in the knowledge graph. The cells containing **?** symbol correspond to the missing/unknown connections in the knowledge graph. The link prediction model will predict the existence of these missing edges. The dimensions of the tensor is

$n_t \times n_c \times n_l$, where n_t, n_c, and n_l are the number of unique target words, context words and labels. Figure 4.3 illustrates the tensor that is constructed using the knowledge graph.

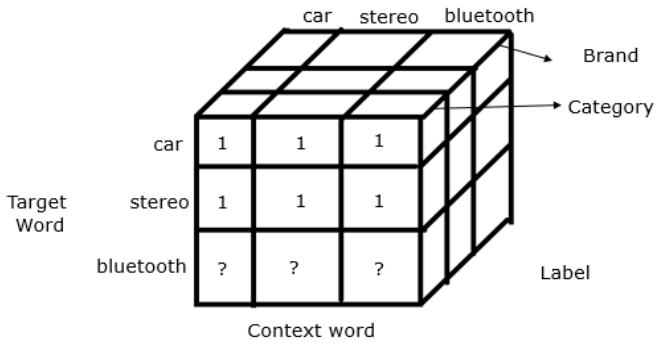


Figure 4.3: Binary Tensor representation of the knowledge graph. The 1's indicate a valid connection between (target_word, context_word, label) and **?** indicate missing/unknown links.

4.3 Tucker Decomposition

The Tucker decomposition [34] factorizes a tensor into a core tensor and separate factor matrices for each mode of the tensor. We want to decompose the binary tensor that is constructed from the knowledge graph mentioned in 4.2. The Tucker decomposition of a third-order tensor $\boldsymbol{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ is defined as:

$$\hat{\boldsymbol{X}} \approx \boldsymbol{G} \times_1 A \times_2 B \times_3 C \tag{4.1}$$

$$= \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \sum_{k=1}^{n_3} g_{ijk} \mathbf{a}_k \otimes \mathbf{b}_k \otimes \mathbf{c}_k \tag{4.2}$$

where $\mathbf{G} \in \mathbb{R}^{r_1 \times r_2 \times r_3}$ is the core tensor of the decomposition and the matrices $A \in \mathbb{R}^{n_1 \times r_1}$, $B \in \mathbb{R}^{n_2 \times r_2}$, $C \in \mathbb{R}^{n_3 \times r_3}$ are the factor matrices for the three modes and where (n_1, n_2, n_3) correspond to the dimensions of the input tensor and (r_1, r_2, r_3) are the ranks of the factorized components. When operating on unfolded tensors, Equation (4.1) can be rewritten as

$$\begin{aligned}
\boldsymbol{X}_{(1)} &\approx A G_{(1)} (C \otimes B)^T, \\
\boldsymbol{X}_{(2)} &\approx B G_{(2)} (C \otimes A)^T, \\
\boldsymbol{X}_{(3)} &\approx C G_{(3)} (B \otimes A)^T.
\end{aligned}$$

which is useful for the computation of the decomposition. Tucker decomposition is also known as Higher-Order SVD (HOSVD), as introduced in [38] and three-mode PCA as introduced in [39].

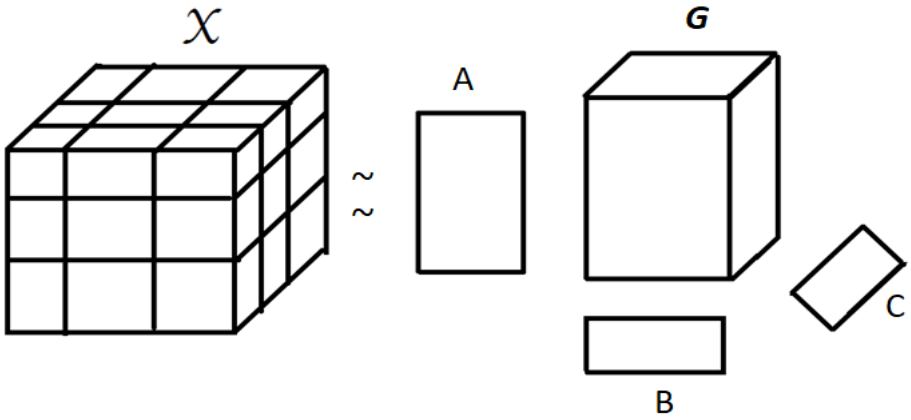


Figure 4.4: Visualization of a Tucker decomposition into its decomposed components. The decomposed components consists of three matrices (A,B,C) each containing the embeddings of the target word, context word and label. The core tensor $\boldsymbol{G}$ contains the level of interaction between the embeddings in A,B,C.

In Link Prediction, we are given a subset of all true triples and the aim is to learn a scoring function ϕ that assigns a score $s = \phi(e_A, e_B, e_C) \in \mathbb{R}$ which indicates whether a triple is true with the ultimate goal of being able to correctly score all missing triples. (e_A, e_B, e_C) are the rows of A, B and C representing the embeddings of target words, context words and label. The scoring function for Tucker is defined as:

$$\phi(e_A, e_B, e_C) = \mathbf{G} \times_1 e_A \times_2 e_B \times_3 e_C \tag{4.3}$$

We apply sigmoid to each score $\phi(e_A, e_B, e_C)$ to obtain the predicted confidence p of a triple. The sigmoid function squishes the confidence score between $[0, 1]$ indicating the probability of a triple being true. The number of parameters increase *linearly* with respect to target words, context words and label embedding dimensions. The number of parameters for $\mathbf{G}$ depends only on embedding dimensionality of A, B and C and not on the number of target words or context words. Following the training procedure introduced by [40] to speed up training, we use 1-N scoring, i.e. we simultaneously score target-context word pairs with all labels in contrast to *1-1* scoring where individual triples are trained one at a time. The model is trained to minimize the Bernoulli negative log-likelihood loss function. A component of the loss for one target-context word pair with all others labels is defined as:

$$L = -\frac{1}{n_l} \sum_{i=1}^{n_l} (y^{(i)} \log(p^{(i)}) + (1 - y^{(i)}) \log(1 - p^{(i)})), \tag{4.4}$$

where $p \in \mathbb{R}^{n_l}$ is the vector of predicted confidence score for all the labels and $y \in \mathbb{R}^{n_l}$ is the

binary label vector.

4.4 Inference Stage

The link prediction model helps in predicting missing links in triples. In our case, we are interested in identifying the right label for a target word given its context. We propose a novel approach for using contextual information while making a prediction. Let S be a sentence in the product description consisting of words $(w_1, w_2, ..., w_n)$ where $w_i \in S$ is the tokenized version of S. To obtain the prediction for a word w_i we perform the following steps:

- Construct tuples of (w_i, w_j) where w_i is the target word, w_j are the context words in the sentence S, where $1 \leq j \leq n$. These tuples will be of the form: (w_i, w_1), (w_i, w_2), ..., (w_i, w_n).

- For each tuple (w_i, w_j) we obtain the predicted confidence score for all the labels. Let the confidence score for each label α_k be C_k, where $1 \leq k \leq m$, and m is the total number of labels.

- This way we have $n \times m$ triples of (w_i, w_j, α_k) where w_i is the target word, $w_1, ..., w_n$ are the context words and α_k is the label. For each triple we have a confidence score between $[0, 1]$.

- We define an aggregation function to sum up the confidence score across all triples for a particular label. These triples would be represented as (w_i, w_j, α_k), with i, k fixed and $1 \leq j \leq n$.

$$\beta_k = \phi(w_i, w_j, \alpha_k) = \sum_{j=1}^{n} C_k \tag{4.5}$$

  This aggregation function gives us an aggregated confidence score for each entity across all the context words for a specific target word. The aggregated confidence score for each label α_k is represented by β_k.

- We calculate a normalization constant for obtaining normalized confidence score across all labels for the triples (w_i, w_j, α_k). The normalization constant is represented by γ.

$$\gamma = \sum_{k=1}^{m} \beta_k \tag{4.6}$$

- Using Equation 4.5 and 4.6 we calculate the normalized confidence score between [0,1]. For each label α_k the normalized confidence score Z_k is calculated as follows:

$$Z_k = \frac{\beta_k}{\gamma} \tag{4.7}$$

This approach provides us with numerous advantages. Firstly, it allows us to provide contextualized predictions for a target word. This is important as now the model can predict an attribute/label for a target word based on the product description. Secondly, this approach tells us the contribution of each context word towards a certain prediction. Equation (4.5) allows us to calculate confidence score for each label across all context words and this feature is used to make predictions interpretable.

Chapter 5

Link Prediction with Unlabeled Data

In this chapter we answer RQ2 with each section tackling the sub-questions mentioned in Section 3.2.

- In Section 5.1, we propose using hypernyms to act as a substitute for attributes. Here we give a brief introduction about Babelnet [41], which acts as an external knowledge base for obtaining hypernyms.
- Section 5.2 provides an approach for creating a knowledge graph from unlabeled data.
- In Section 5.3 and 5.4, we briefly discuss representing the knowledge graph as a tensor and apply Tucker decomposition, which we use as our baseline approach.
- In Section 5.5, we present several approaches for improving the model performance from our baseline approach.

5.1 Hypernyms and Hyponyms

We want to identify the broad or the generic term associated with a word in the product description. Since we deal with unlabeled data, we need a way to find these broader terms that can be used as a proxy for attributes/labels. To do this, we use hypernyms. A hypernym is a word with a broad meaning constituting a category into which words with more specific meanings fall. For example, in Figure 5.1, we see that color is a hypernym of red, and red is the hyponym.

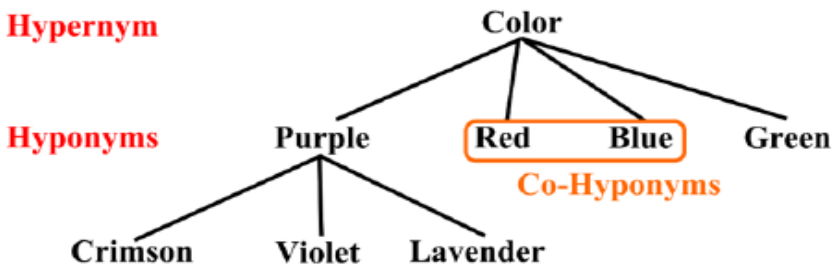


Figure 5.1: Hypernym-hyponym relationship for colors [42]. Color is the hypernym, and purple, red, blue, green are hyponyms.

If we are able to find the right hypernym for our words then we can translate this problem to our previous research question. The question at hand is, how can we obtain these hypernyms?

5.1.1 Babelnet

Several lexical ontology databases provide a hierarchical relationship between words such as hypernym-hyponym relationships. Wordnet [43] and Babelnet [41] are the two of such databases that we explored during our research. Babelnet is a multilingual encyclopedic dictionary, with lexicographic and encyclopedic coverage of terms and a semantic network. Compared to Wordnet, Babelnet is richer in vocabulary; hence we use it in our experiments. In the following section, we explain how Babelnet is used to obtain the candidate hypernyms, helping us construct the knowledge graph.

5.2 Constructing Knowledge Graph

The previous chapter explained creating a knowledge graph using product descriptions when the label is already known. In this case, we do not have the labels provided to us; hence we create a framework for constructing labeled data using Babelnet.

5.2.1 Data Pre-processing

Using raw data could hamper the quality of the model that we are building and reduce the system's performance. Data cleaning reduces variability in the information to give the model the best chance of working correctly. Below are the pre-processing operations that we apply to the product descriptions:

- **Case standardization:** Converting the listing to lowercase. This is done to reduce variations between words, thereby reducing the size of the vocabulary in the data.
- Tokenization: is the process of segmenting running text into sentences and words. We split the sentences into words based on blank spaces.
- **Remove special characters:** post tokenizing the listing, we identify the tokens which correspond to punctuation or any special characters and remove them from the listing.

We do not remove period ('.') punctuation from the listings. This is because a listing can consist of multiple sentences that can explain different aspects of a product. In such cases, these sentences can have different contexts; hence, keeping them separate was a logical choice.

- **Lemmatization:** it is done to reduce a word into its base form or lemma. This is done to reduce the size of the vocabulary.
- **Standardizing numeric and alphanumeric characters:** Numeric characters can be associated with several broad terms or labels and are not restricted to a limited set of labels; for example, numbers can be associated with model numbers of a mobile phone or even the price of a product. Same goes with alphanumeric characters, hence we consider these words to be out of our scope and replace them with $< num >$ *and* $< alphanum >$ respectively.

Once the above pre-processing steps are done, we combine the tokens and obtain the product description's clean version. Figure 5.2 shows the steps that we undertook for cleansing the data, along with an example showing the transformation of the product description at each step of the operation.

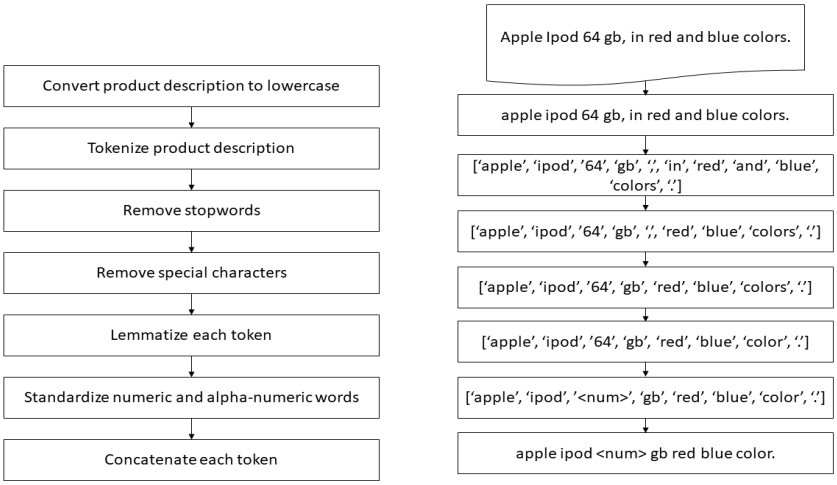


Figure 5.2: Data pre-processing steps. On the left we have the data pre-processing steps, and on the right we provide an example.

5.2.2 Extracting Hypernyms using Babelnet

Now that we have obtained the clean version of the product descriptions, our next step is to obtain the broad or the generic terms that are associated with the words in our corpus. We use Babelnet to extract the hypernyms. We create a knowledge base consisting of a word and its corresponding hypernym. For every word in the product description, we obtain its hypernym in the following manner:

- **Word to Synset ID:** Like WordNet, Babelnet groups words in different languages into sets of synonyms, called Babel synsets. The first step is to find the Babelnet synset id

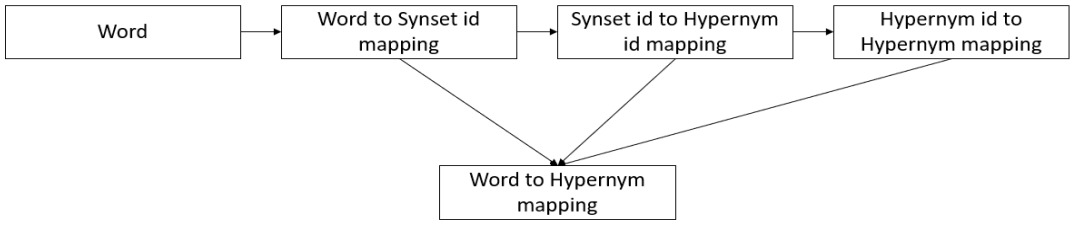


Figure 5.3: Pipeline for creating mapping between word and its hypernym. For each word we obtain its hypernyms using the above process.

for each word. A word can be used in many different ways and can be associated with more than one meaning. Each word has a one to many relationships with synset ID. This can be seen in Figure 5.4 for the word apple.

- **Synset ID to Hypernym ID:** Once we have obtained the synset id, the next step is to find the hypernym associated with the synset id. Each synset id has one to many relationship with hypernym id.

- **Hypernym ID to Hypernym:** For each hypernym id, we have one hypernym associated with it. There is a one to one relationship between hypernym id and hypernym.

- **Word to Hypernym:** Now that we have the synset ID, hypernym ID, and hypernym associated with each word, we map each word to its corresponding set of hypernyms. To do so, we use the mappings, word to synset ID, synset ID to hypernym ID, and hypernym ID to hypernym mapping.

Figure 5.3 is a high-level diagram of the process, and Figure 5.4 we give an example of the above mappings for the word "Apple".

In Figure 5.4, we can see that some of the hypernyms are of multiple words concatenated by '_' character. We flatten these hypernyms by removing the '_' character and treat each word as the hypernym. This decision was taken because of how we construct our knowledge graph, which we explain in the subsequent section.

5.2.3 Knowledge Graph Construction

At this stage, we have standardized product descriptions and word to hypernym mapping that is constructed using Babelnet. Here we present the solution of creating the knowledge graph. This approach is different compared to the approach mentioned in Section 4.1. Below we present the heuristics for constructing the knowledge graph

- Create pairs of words or tuples consisting of every target word with the context in the cleaned product description.

- For every tuple, we want to identify the candidate hypernym, which would be the label/attribute. Below are the set of heuristics that we apply for identifying the hypernym.

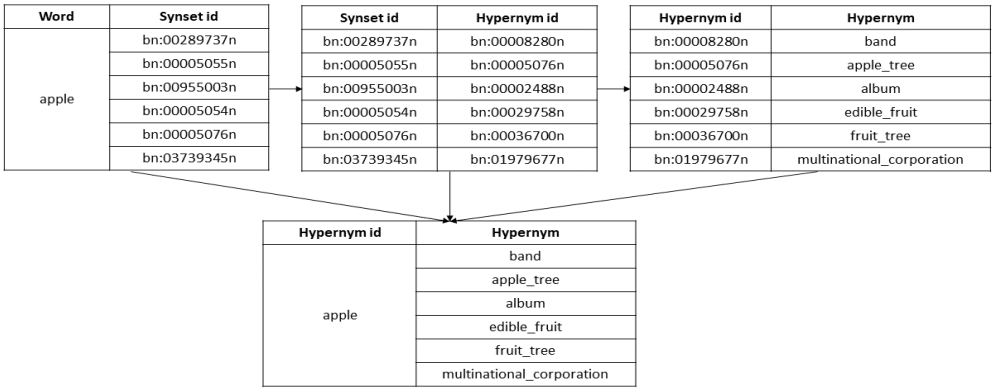


Figure 5.4: Example of hypernym extraction process for the word: apple. Each word can have multiple meanings and for each meaning there is a hypernym. Word to hypernym mapping is a one to many relationship.

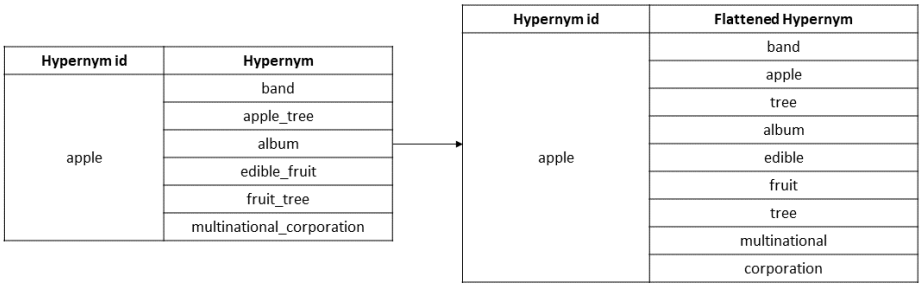


Figure 5.5: Hypernyms consisting of multiple words are flattened for obtaining a better match.

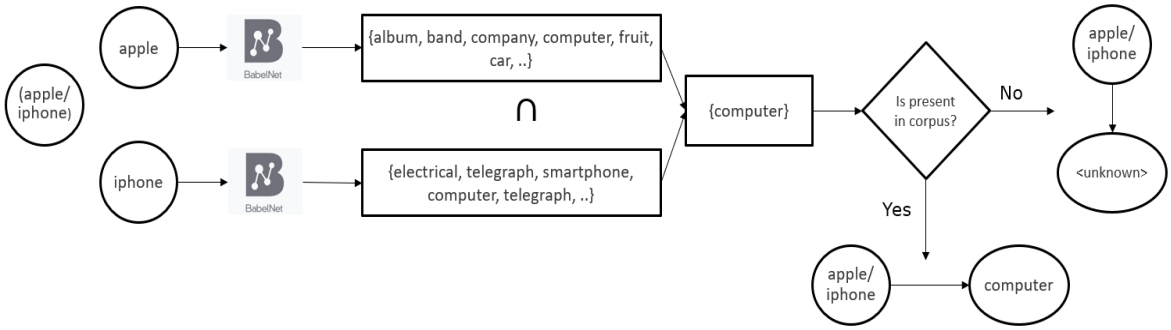


Figure 5.6: Knowledge graph construction for unlabeled data using Babelnet by finding the common hypernym between target and context word.

- Check if the target word and context word have any hypernyms in common.
- If yes, then check if the hypernym exists in our vocabulary of the corpus. Here corpus refers to the data-set that we use for our experiments, and vocabulary refers to the set of unique words in the data-set. This acts as a filtering mechanism as we do not want hypernyms, which have no association with our data-set.
- The hypernyms which pass the above filtering method are considered as the hypernym for the tuple.
- Create triple with $(target\_word, context\_word, hypernym)$
- If we do not find any hypernym in common, we assign the tuple to an "$\langle unknown \rangle$" tag. In this case, the triple will be $(target\_word, context\_word, \langle unknown \rangle)$

In Figure 5.6, we illustrate a knowledge graph construction for a tuple using Babelnet. The figure shows the steps of constructing the triple from the tuple of $(apple, iphone)$.

5.3 Constructing Tensor

From the implementation standpoint, we represent this knowledge graph in the triple format consisting of $(target\_word, context\_word, hypernym)$. The triples connected to the node $\langle unknown \rangle$ are treated as missing links that we want the model to predict, and the remaining links are labeled as 1 in the tensor. The process of creating the tensor is in the same format, as mentioned in Section 4.2.

5.4 Standard Tucker decomposition

We use the Tucker decomposition method that has been mentioned in Section 4.3 as our baseline model. The dimensions of the input tensor are $n_t \times n_c \times n_l$ where $n_t and n_c$ represent the number of unique target words and context words and n_l represent the unique number of labels or hypernyms in this case.

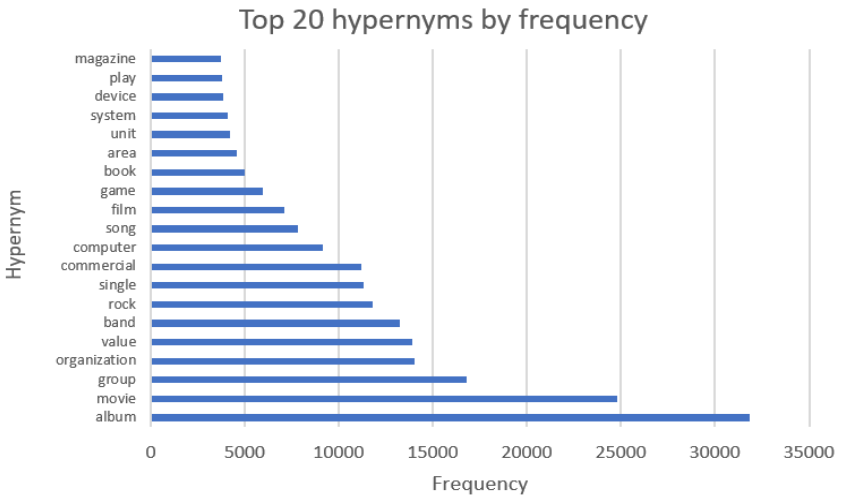


Figure 5.7: Top 20 hypernyms by frequency obtained using the heuristics mentioned in Section 5.2.3. Most of these hypernyms are not aligned with our task.

5.5 Strategies for Improving Model Performance

To improve the baseline approach, we propose multiple strategies for improving the model and improving the input data, i.e., the triples in our case. The triples that were formed using the heuristics mentioned in Section 5.2.3 are noisy in nature. By noisy, we mean that the triples consisted of hypernyms that were not relevant for our use case. In Figure 5.7, we illustrate the most frequent hypernyms that are obtained by our heuristics. We can see that words such as *album, rock, single, song, band* are not relevant labels for electronic products. To improve the performance of the model we suggest techniques for modifying the input data, and the model.

5.5.1 Non-negative Tucker Decomposition

In the standard tucker decomposition, presented in 4.3 gives a confidence score between $[0, 1]$, providing the probability of true relation between the target word, context word, and hypernym.

Tucker decomposition is the summation of the outer product of the vectors of factor matrices (refer Equation (4.2)) which may lead to situations where certain target-context word pair has a low score as certain features contribute negatively in the summation. These negative scores would then hamper the effect of the features that contribute positive scores, which might include the true labels. This leads to having incorrect predictions. By applying non-negative constraints to the factor matrices and core tensor restricts the values between $[0, 1]$ instead of negative values. Restricting the values of Tucker components to [0,1] allows the positive score to dominate over the negative scores.

Also, in the inference stage mentioned in Section 4.4, while predicting the labels for a particular target word, we aggregate the confidence score for each label (hypernym in this case) obtained from the triple $(target\_word, context\_word, hypernym)$. Having non-negative scores

allows the context words that matter to contribute towards predicting the correct hypernym.

5.5.2 Orthogonal Regularizer

In this section, we briefly introduce regularization and provide details about the regularization technique applied to our use-case. Regularization is the process of regularizing a machine learning model's parameters and preventing the model from over-fitting. Figure 5.8 shows two curves incurring zero losses as they both fit the red points perfectly. However, we can say that the green curve is probably over-fitting, and the curve in blue has more generalization capability. We want to avoid a scenario where our model has the nature of the green curve, and to do so; we introduce an orthogonal regularizer.

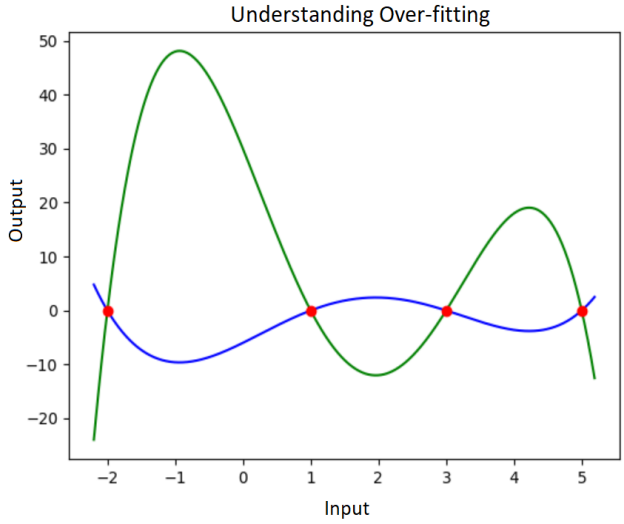


Figure 5.8: Understanding over-fitting: The green and blue functions both incur zero loss on the given data points. The blue function generalises better than green function [44].

In linear algebra, orthogonality refers to the property of perpendicularity between two vectors. Orthogonality is also used for the specific separation of features in a system. One way to express orthogonality in a matrix is

$$XX^T = I$$

where X^T is the transpose of X and I is the identity matrix. On observing the predictions obtained from the baseline model, we observed biased predictions towards certain words such as *album, rock, band.* This is because the embeddings for these hypernyms/labels is dense, and each feature contributes towards the confidence score. The orthogonal regularizer introduces sparsity in the embeddings. We want to penalize the overlap between the columns of XX^T. Hence we apply the orthogonal regularizer to the hypernym factor matrix C. For the factor matrices related to target words (A) and context words (B) we apply L2 regularization. L2 regularization for vector $\boldsymbol{x}$ with n features is defined as the sum of the squares of all the feature weights as shown in Equation (5.1):

$$||\boldsymbol{x}||_2 = x_1^2 + x_2^2 + x_3^2 + .. + x_n^2 \tag{5.1}$$

Considering the ideas mentioned above, below is the regularizer that we apply to our model is presented in Equation (5.2).

$$\lambda_1||A||_2 + \lambda_2||B||_2 + \lambda_3(||CC^T||^2 - ||C||^2), \tag{5.2}$$

where $\lambda_1, \lambda_2,$ *and* λ_3 are the regularization parameters, and A,B,C are the factor matrices from Tucker decomposition.

5.6 Modifying Input Data

One way to improve the performance of the system is by improving the quality of input data. In our case, the input data corresponds to triples of (target_word, context_word, hypernym). This section suggests strategies to improve the quality of input data by obtaining relevant hypernyms.

5.6.1 Using Category Information

Every product description in the OLX data-set has a category associated with it. Figure 5.9 shows an example of a listing with its category information. The category information is used as a feature for identifying the relevant hypernyms for a tuple of $(target\_word, context\_word)$.

Babelnet consists of synsets that are labeled to domains. Each synset in Babelnet is associated with a domain. Figure 5.10 illustrates hypernyms and domains associated with the word apple.

Figure 5.9: OLX listing with the Category Information. The Category Information can be used to find relevant hypernyms.

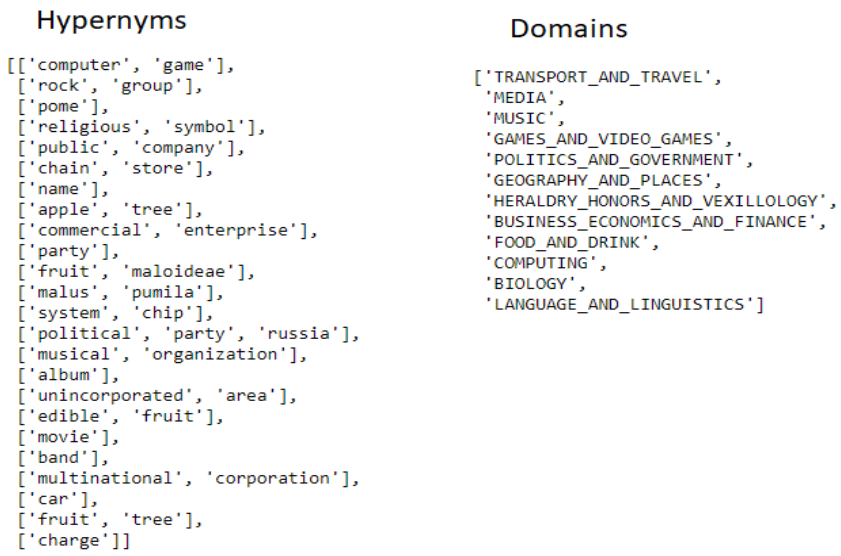


Figure 5.10: Left: Hypernyms of apple, Right: Domains for the word apple. We use the domains to find the relevant hypernym associated to the (target_word, context_word).

Given a tuple of $(target\_word, context\_word)$ we use the technique as mentioned in Section 5.2.3 to find the candidate hypernyms. Category information is tokenized, and domains associated with each token is retrieved. The domains are aggregated based on the frequency, and the hypernyms associated with the top two domains are considered. Figure 5.11 shows the process mentioned above for the tuple $(apple, apple)$. In Section 6.4.4 we present the impact of using category information on model performance.

5.6.2 Using Windowing

In Section 5.2.3, we state that a target word's context is the surrounding words in a sentence. We alter this approach by introducing a window size and consider a fixed number of words as the context and not the entire sentence. We apply this approach to reduce the amount of noise or the number of hypernyms we create using our baseline approach. Reducing the noise would improve the quality of the data, which can help us improve the model's performance.

While constructing tuples for a target word, we consider the context words based on window size. We show the process of constructing tuples for the word *apple* and *iphone* while considering window size as 2. We consider the target word and the set of context words while constructing tuples, as shown in Figure 5.12. In Section 6.4.5, we show the impact of window size on model performance.

5.6.3 Using Pre-trained Embeddings

In Natural Language Processing (NLP), while processing text for tasks such as classification or clustering, we need to represent or encode the text in a vectored format to be consumed by the downstream systems. We hypothesized that using pre-trained embedding models would help find the most semantically close hypernym for the tuples in our input data. We explored

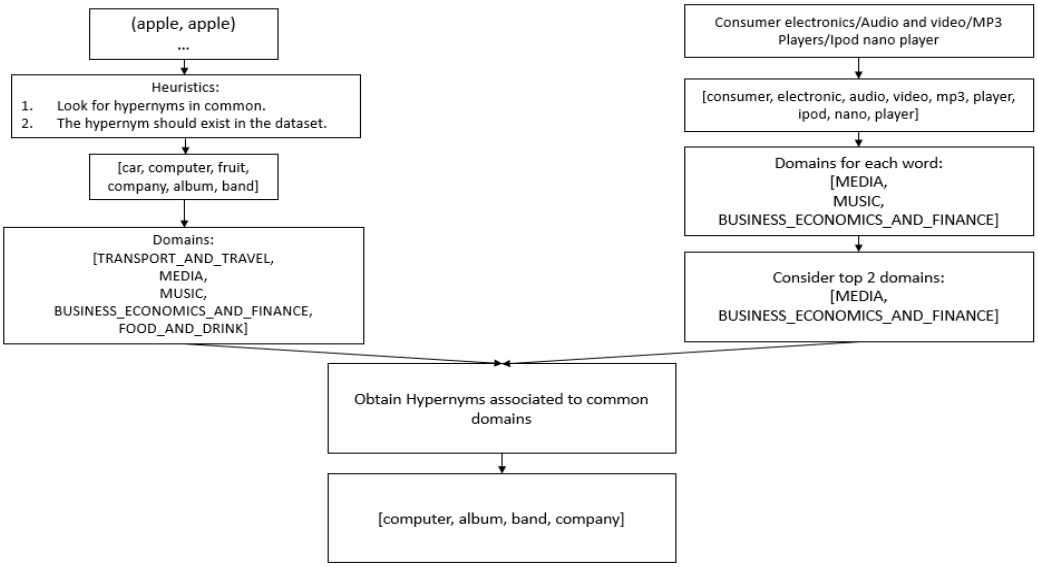


Figure 5.11: Using Category Information to find the relevant hypernyms for the tuple (apple,apple). Here we see process of refining the set of hypernyms obtained for a tuple by using domain information from Babelnet and Category Information from the listing.

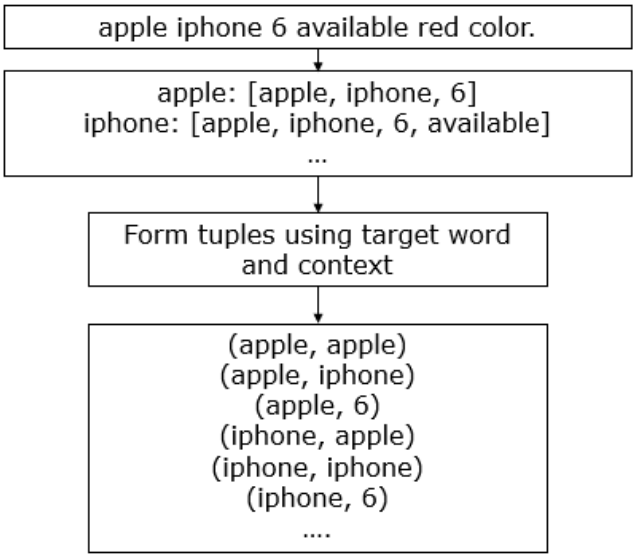


Figure 5.12: Restricting the context words based on window size while constructing tuples.

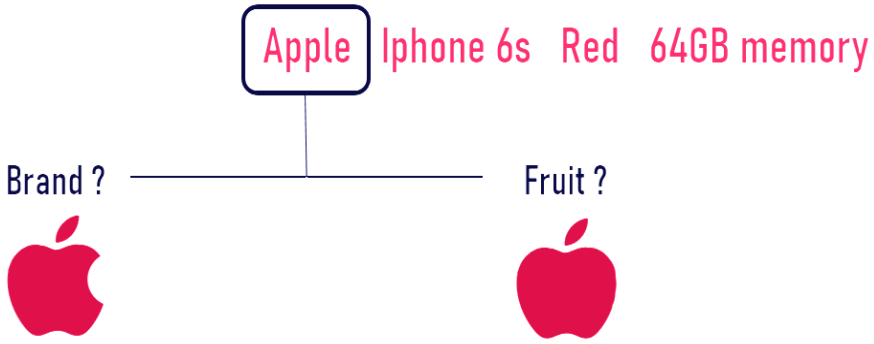


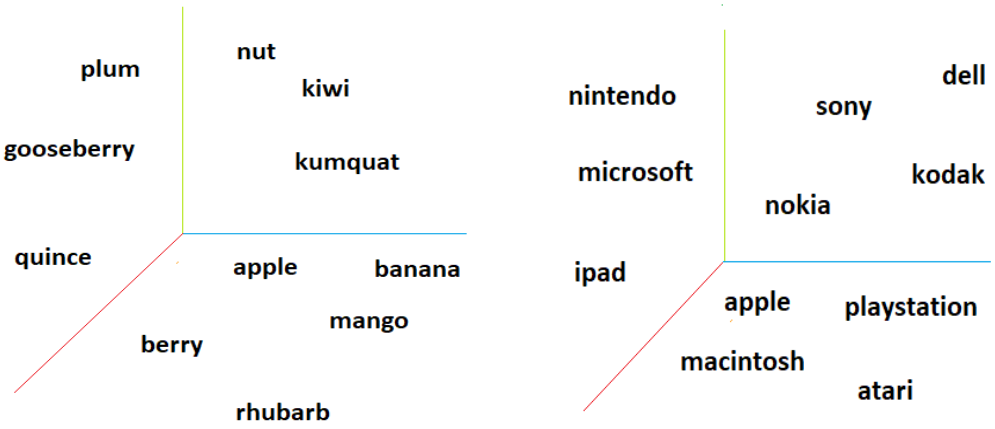
Figure 5.13: Word2Vec and GloVe embedding models cannot capture multiple representations for a single word. Contextualized word embeddings can capture multiple representations for a single word.

some of the popular word embedding algorithms that would be suitable for our use-case and help find the relevant hypernym.

Word embedding algorithms such as Continuous Bag Of Words (CBOW), Skipgram [45], and GloVe [46] do not capture contextual information; instead, provide a global representation for each word. Word ambiguity or polysemy (the coexistence of many possible meanings for a word or phrase) cannot be captured by these models. In Figure 5.13, we can see that Apple can have more than one meaning; however, word embedding algorithms such as GloVe and Word2Vec can capture only a single representation of each word.

Due to the limitation mentioned above, we look into deep contextualized word representation models such as ELMo [47] and BERT [28] that can handle polysemy. These models are highly sensitive, capturing even the slightest change in the context, resulting in different target words' representations. This would lead to a large number of representations for a single word when, in reality, the word is only used in a handful of different contexts. Hence, fixed embeddings for the words cannot be obtained directly by these models. We want our embeddings to be contextualized and fixed per meaning, which is provided by Adaptive Skip-gram [48]. Adaptive Skip-gram (AdaGram) model (which extends the original Skip-gram [45]) automatically learns the required number of prototypes for each word using a Bayesian non-parametric approach. With adaptive skipgram, we have separate vectors for different meanings of a single word. For example, apple can have one representation referring to apple as a fruit and the other with apple being used in electronics. Figure 5.14 shows the nearest neighbors for the word apple being used in these two contexts.

We use the AdaGram embeddings and combine with our heuristics mentioned in Section 5.2.3 to find the most semantically associated hypernyms. We find the common hypernyms for every tuple and calculate the similarity between the candidate hypernyms and the tuple. We use cosine distance as the similarity metric to find the most semantically associated hypernym. Cosine similarity between two vectors, $\boldsymbol{x}$ *and* $\boldsymbol{y}$ can be defined using the dot product and magnitude of the two vectors:



(a) Nearest Neighbours for Apple fruit.

(b) Nearest Neighbours for Apple brand.

Figure 5.14: Visualization of Nearest neighbours for AdaGram Embeddings. PCA is used to represent high dimension embeddings in three dimensions.

$$Cosine\ Similarity(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}.\boldsymbol{y} = \frac{\boldsymbol{x}.\boldsymbol{y}}{||\boldsymbol{x}||||\boldsymbol{y}||} \tag{5.3}$$

Let $\boldsymbol{x}$ *and*$\boldsymbol{y}$ be the vector representation for the tuple of $(target\_word, context\_word)$ and $\boldsymbol{z}$ be the vector representation of a candidate hypernym. The hypernym would be considered only if the cosine similarity between the tuple and hypernym exceeds a certain threshold which is decided by experiments in Section 6.4.6. The cosine similarity is defined as follows:

$$Similarity = \frac{cos\_sim(\boldsymbol{x}, \boldsymbol{z}) + cos\_sim(\boldsymbol{y}, \boldsymbol{z})}{2}, \tag{5.4}$$

where $cos\_sim(\boldsymbol{x}, \boldsymbol{z})$ and $cos\_sim(\boldsymbol{y}, \boldsymbol{z})$ is the cosine similarity between vector $\boldsymbol{x}, \boldsymbol{z}$ *and* $\boldsymbol{y}, \boldsymbol{z}$ defined in Equation (5.3).

Chapter 6

Evaluation

In this chapter, we explain the experimental setup used to evaluate our model. This includes the data-sets, implementation details, and the evaluation metrics used.

6.1 Evaluation metrics

To evaluate the model for the task of link prediction, we use metrics standard across the link prediction literature. Hits *@k* to measure measures the percentage of times a true triple is ranked within the top k candidate triples.

To evaluate the inference stage method we evaluate precision *@k*, recall *@k* and F1 *@k* for each attribute/label. Precision *@k* is the proportion of recommended attributes/labels in the top-k set that are relevant. Recall *@k* is the proportion of relevant attributes/labels found in the top-k recommendations. For measuring these metrics we need to following quantities:

- True positives *@k* (TP *@k*): True attributes/labels occurs in the top k system generated entities.
- False positives *@k* (FP *@k*): This is the case when true attributes/labels does not occur in the top k system-generated attributes/labels. We consider the system generated entities as false positives in this case.
- False negatives *@k* (FN *@k*): This is the case when the true attributes/labels do not occur in the top k system-generated attributes/labels. We consider the true attributes/labels as a false negative in this case.

Based on the above quantities we can calculate the following metrics for each entity as follows:

$$Precision\ @k = \frac{TP\ @k}{TP\ @k + FP\ @k} \tag{6.1}$$

$$Recall\ @k = \frac{TP\ @k}{TP\ @k + FN\ @k} \tag{6.2}$$

$$F1\ @k = 2 * \frac{Precision\ @k * Recall\ @k}{Precision\ @k + Recall\ @k} \tag{6.3}$$

6.2 Setup

To answer the research questions, we conducted experiments on two data-sets mentioned in Section 3.1.

- To answer RQ1, we use the labeled public data-set. The data-set consists of 941 product descriptions. We compare our model's performance with the baseline provided in [10].
- To answer RQ2, we use the unlabeled data-set. We consider three distinct categories of products consisting of 967 product descriptions. These product descriptions fall under the umbrella of consumer electronics. For testing the performance of the link prediction model, we randomly sample 20 product descriptions from each category and manually construct triples. For evaluating the performance of the inference stage, we manually construct the labels for the words in product descriptions of 50 randomly sampled product descriptions.
- To answer RQ3, we use 4 distinct domains consisting of 9131 product descriptions from the unlabeled data-set. These product descriptions fall under the umbrella of consumer electronics, cars, footwear, and furniture. For evaluating the link prediction performance, we manually construct the labels for the words in product descriptions of 50 randomly sampled product descriptions from each category.
- The standard Tucker decomposition model is considered as our baseline model. We use the codebase provided by [35] and extend it for our use-case. Tucker decomposition is implemented in PyTorch [49].
- The experiments are conducted on NVIDIA DGX station using Tesla V100 32GB/GPU. We use `https://www.comet.ml` for experiment management purposes.

6.3 Results from Labeled Data-set

In this section, we report the results of the experiments conducted on the labeled public data-set and answer RQ1. Figure 6.1 shows the frequency distribution of the labels in the data-set. Due to the imbalanced distribution of the labels, we report results for the most frequently occurring entities, which are *Brand, category, and ModelName.*

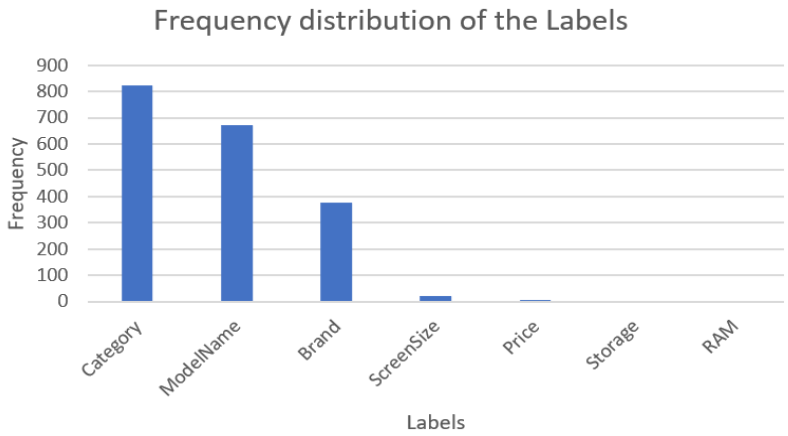


Figure 6.1: Frequency distribution of the labels. Out of seven labels, three are most frequently occurring in the data-set. We report the results for the labels Category, ModelName, and Brand.

We split the data-set into 800 product descriptions for training and 141 for testing. The hyper-parameters are chosen by the random search based on performance on the training set. The target word, context, and label/attribute embedding dimensionality or the rank of the Tucker components are experimented with two values: $\{100, 300\}$. We use batch normalization [50] to speed up the training phase of the model. Batch Normalization is a technique that mitigates the effect of unstable gradients within deep neural networks. Batch normalization introduces an additional layer to the neural network that performs operations on the inputs from the previous layer. The learning rate is set to $\{0.0005\}$. Unlike [35], we do not use dropout in our training process because tensor factorization decomposes the input tensor into its latent factors, and using dropout would mean not using these latent factors, which could hamper the learning process for the model. We also train the model by applying non-negative constraints, as mentioned in Section 5.5.1. We refer to this model as *NTD* in the tables below and compare its performance with the baseline model. To evaluate model for the task of link prediction we use metrics Hits *@k*, where $k \in \{1, 3, 5\}$. Table 6.1 provides the results on the test set for the models trained under different settings of embedding dimensions and baseline vs. non-negative constraints on the factor components of Tucker decomposition.

Model	**Embedding Dimensions**	**Hits @1**	**Hits @3**	**Hits @5**
Baseline	100	0.9776	1	1
Baseline	300	0.9708	1	1
NTD	100	**0.9799**	1	1
NTD	300	0.9754	1	1

Table 6.1: Link Prediction Performance on Test set. Baseline model refers to standard Tucker decomposition and NTD refers to Tucker decomposition with non-negative constraints. It is apparent from this table that Tucker decomposition model is good at predicting true triples, with NTD model having embedding dimension as 100 giving the best results.

To evaluate the inference stage method presented in Section 4.4 we calculate precision *@k*,

recall @k and F1 @k, ($k = 1$) for the most frequently occurring labels: *Category, ModelName, and Brand*. Based on the performance shown in Table 6.1 we choose model trained on non-negative constraints (NTD) with embedding dimension: 100 as the candidate model for calculating classification metrics, Precision @1, Recall @1, F1 @1 for the labels *Category, ModelName, and Brand*. In Table 6.2 we compare the results of our model with [10].

Model	Label	Precision	Recall	F1 score
SVM	Brand	0.895	0.486	0.630
	Category	0.667	0.531	0.591
	ModelName	0.800	0.431	0.560
GBT	Brand	0.793	0.697	0.742
	Category	0.603	0.620	0.611
	ModelName	0.727	0.480	0.578
CRF	Brand	0.818	0.614	0.701
	Category	0.621	0.711	0.663
	ModelName	0.631	0.746	0.684
NTD	Brand	**0.8947**↑	**0.8095**↑	**0.85**↑
	Category	**0.9594**↑	**0.8068**↑	**0.8765**↑
	ModelName	**0.9594**↑	**0.8068**↑	**0.8765**↑

Table 6.2: Classification metrics on test set for most frequently occurring labels. What stands out in the table is the performance of the non-negative Tucker decomposition model (NTD). It is significantly better than the approaches mentioned in [10].

6.4 Results from Unlabeled Data-set

In this section, we report the results of the experiments conducted on the unlabeled data-set for answering RQ2. Since the data-set is unlabelled, we construct the knowledge graph using the heuristics mentioned in Section 5.2. We use the link prediction metrics (hits @k) to identify the best model.

6.4.1 Tucker Decomposition

To establish baseline performance, we apply the Tucker decomposition mentioned in Section 4.3. Table 6.3 shows the link prediction results obtained from the test set. Interestingly, the results indicate that the model has a hard time predicting the true triple. The next section presents the results that are obtained by implementing the strategies mentioned in Section 5.5.

Model	Hits @15	Hits @10	Hits @5	Hits @3
Baseline	0.4146	0.3219	0.2487	0.1756

Table 6.3: Baseline Tucker decomposition performance on unlabeled data. Interestingly, the results indicate that the model is having a hard time to predict the true triple.

6.4.2 Non-negative Tucker Decomposition

Based on the results obtained from the previous section, we proposed strategies for modifications of the model and input data in Section 5.5. In Section 5.5.1, we indicate that by introducing non-negative constraints in the factor matrices and core tensor, we allow the true/correct relations to dominate over incorrect/false relations. In Table 6.4, we present the results of non-negative Tucker decomposition (NTD) and compare it with the baseline model performance on the test set. We observe an improvement in the performance from the baseline approach by applying non-negative constraints. Hence, our justification for introducing non-negative constraints was corroborated by the results. At this point, we decide to use non-negative constraints on the model and use it as a baseline while adding further modifications.

Model	Hits @15	Hits @10	Hits @5	Hits @3
Baseline	0.4146	0.3219	0.2487	0.1756
NTD	**0.4780**↑	**0.3707**↑	**0.2634**↑	**0.2048**↑

Table 6.4: Non-negative Tucker decomposition (NTD) vs Baseline performance. There is an improvement in the performance of the model by applying non-negative constraints.

6.4.3 Using Orthogonal Regularizer

In Section 5.5.2 we provide the motivation for adding a regularizer to the model with the primary purpose being to enhance the generalizing capability of the model and reduce bias towards certain hypernyms by introducing sparsity into the hypernym factor matrix. Table 6.5 presents the results on the test set obtained by adding regularizer to the pre-existing model with non-negative constraints (NTD). Results indicate an improvement in the metrics over the previous approaches. We believe this is due to the reduction in bias, causing the true hypernyms to have a higher confidence score over the hypernyms, which were being introduced due to the bias of the model. Considering the results, we decide to use the regularizer along with the non-negative constraints approach while conducting further experiments. These are the results that we obtain by providing strategies for improving the model. In the next sections, we provide the results of the model by implementing strategies for improving input data.

Model	Hits @15	Hits @10	Hits @5	Hits @3
Baseline	0.4146	0.3219	0.2487	0.1756
NTD	0.4780	0.3707	0.2634	0.2048
NTD + regularizer	**0.5365**↑	**0.4439**↑	**0.2975**↑	**0.2243**↑

Table 6.5: Link prediction results by using orthogonal regularizer. The results are better compared to the previous approaches.

6.4.4 Using Category Information

Here we provide the model results by using the category information associated with product descriptions and finding relevant hypernyms. In Section 5.6.1, we present the approach for using category information as a filtering mechanism while obtaining hypernyms. To analyze

the impact of using the category information on the input data, we show the number of triples before and after using category information as a filtering mechanism. In Figure 6.2, we observe the drop in the number of triples by 30% by using Category Information. In Figure 6.3, we show the impact of using Category Information on the number of distinct hypernyms. We observe a drop of 45% in the number of distinct hypernyms by using category information. This is because the hypernyms are being limited to the domains decided by the category information. In Table 6.6, we present the link prediction results by using category information and compare it with the results from our previous approaches. We want to highlight that we are also applying the non-negative constraints and orthogonal regularizer to the model while considering category information to train the model. We observe an increase in the model's performance by using category information for Hits @{3,5,10}, which indicates our success in improving the quality of input data by focusing on the relevant hypernyms, thereby reducing noise. Surprisingly, we observe a slight drop in Hits @15 by 0.38% compared to the previous approach of using non-negative constraints and orthogonal regularizer; however, the performance has improved for the lower values of Hits @k, which is of greater significance, as we want the true triple to have a higher confidence score. Overall, based on the positive results obtained, we decide to use category information while constructing triples.

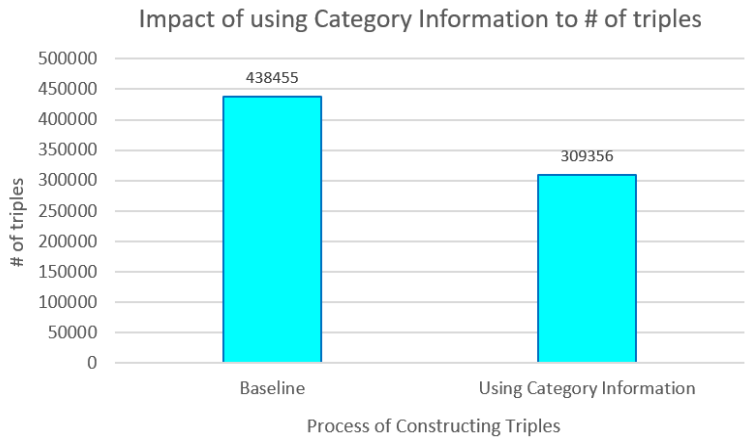


Figure 6.2: Number of triples in Baseline approach and using category information. Baseline refers the heuristics mentioned in Section 5.2.3 for creating triples. We observe a drop in the number of triples by 30% by Using Category Information.

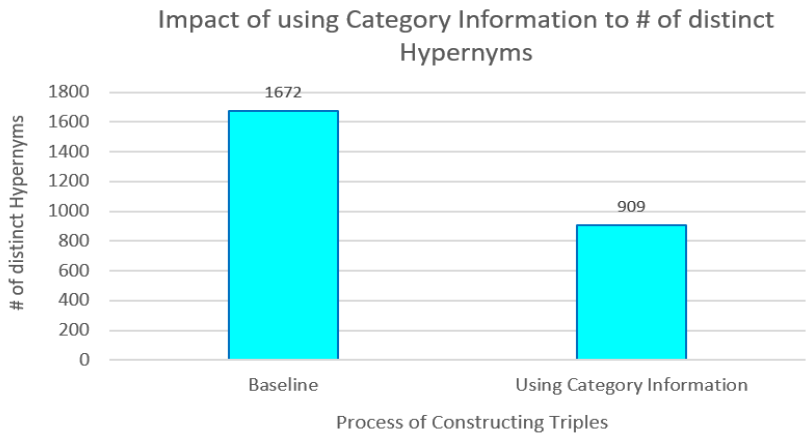


Figure 6.3: Number of distinct hypernyms in Baseline approach and Using Category Information. We observe a 45% drop in the number of distinct hypernyms. This clearly shows that by using Category Information we are limiting the number of hypernyms while constructing triples.

Model	**Hits @15**	**Hits @10**	**Hits @5**	**Hits @3**
Baseline	0.4146	0.3219	0.2487	0.1756
NTD	0.4780	0.3707	0.2634	0.2048
NTD + regularizer	0.5365	0.4439	0.2975	0.2243
Using category information	0.5327↓	**0.4785**↑	**0.3216**↑	**0.2447**↑

Table 6.6: Link prediction results by using Category Information. We observe an improvement in performance of the model by using category information for Hits @{3,5,10} which indicates that we have been successful in improving the quality of input data. Note: Using category information model uses the non-negative constraints and orthogonal regularizer.

6.4.5 Using Windowing

In Section 5.6.2, we present an approach for constructing triples by restricting the number of context words for a target word. To decide the optimal window size, we conduct experiments with different window sizes like {1,2,3,5,10,15}. In Figure 6.4 we present the link prediction results for different window sizes. We observe window sizes {2,5} provide similar results compared to the remaining window sizes. In Table 6.7, we present the results of the windowing approach in comparison with the previous approaches. There is no improvement in the performance of the model by applying the windowing approach. However, we also do not observe a drastic drop in performance when compared to the previous approach of using category information.

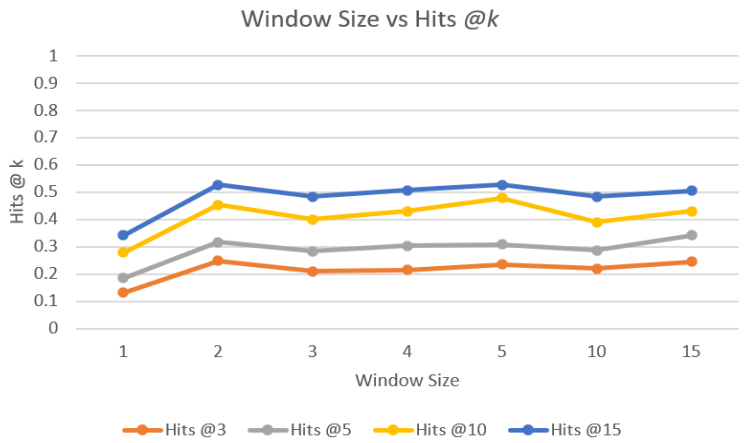


Figure 6.4: Choosing the right window size based on Hits @k. Window size {2,5} provide similar performance compared to remaining window sizes and are the best candidate models.

Model	Hits @15	Hits @10	Hits @5	Hits @3
Baseline	0.4146	0.3219	0.2487	0.1756
NTD	0.4780	0.3707	0.2634	0.2048
NTD + regularizer	0.5365	0.4439	0.2975	0.2243
Using category information	**0.5327**	**0.4785**	**0.3216**	**0.2447**
Using windowing-size 2	0.5268↓	0.4536↓	0.3170↓	**0.2447**
Using windowing-size 5	0.5268↓	0.4780↓	0.3073↓	0.2341↓

Table 6.7: Link prediction results by using windowing. Window size {2,5} provide similar performance, however the performance is not better than the approach using category information.

6.4.6 Using Pre-trained Embeddings

In Section 5.6.3, we present an approach for using existing pre-trained models to find the most semantically associated hypernym for a target word and context word. We use the embeddings of pre-trained Adaptive Skip-gram model [51] for our experiments. We consider {0.3,0.4,0.5} as the similarity threshold between the target word, context word, and the hypernyms. Figure 6.5 shows the link prediction model's performance by varying the similarity threshold. We observe that the 0.3 similarity threshold provides the highest hits @k score; hence we choose the 0.3 threshold while constructing input data.

In Figure 6.6, we compare the results of using pre-trained embeddings with previous approaches. Comparing the results, it can be seen that using pre-trained embeddings does not yield a performance higher than *Using Category Information*. Overall, using pre-trained embeddings did not improve the performance compared to the previous approaches; hence we do not consider this approach while constructing input data.

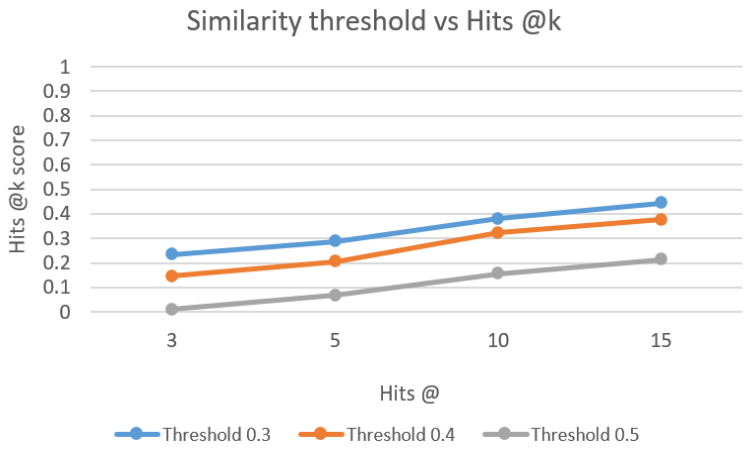


Figure 6.5: Identifying the optimal similarity threshold. On increasing the similarity threshold we observe a drop in the performance of the model. Similarity threshold of 0.3 gives the best performance.

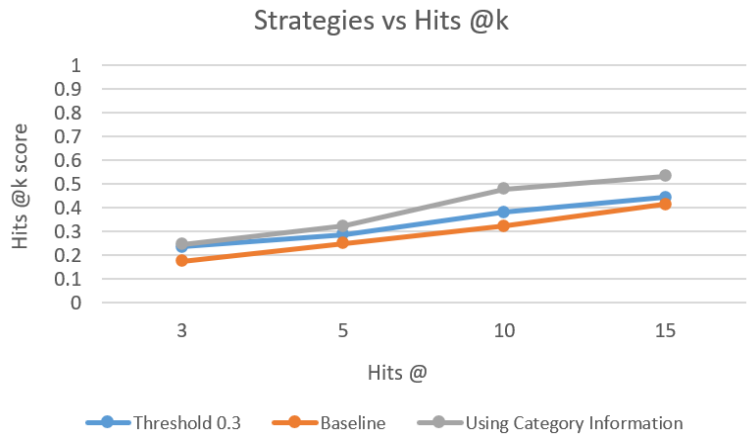


Figure 6.6: Comparing results of using pre-trained embeddings to previous approaches. Using pre-trained embeddings gives better performance than the baseline approach, however it does not perform better than the approach *Using Category Information*.

6.4.7 Hyper-parameter Optimisation

A hyper-parameter is a parameter that is set before the learning process of a model begins. These parameters can be tuned and can directly affect how well a model trains. A set of candidate values were chosen for each hyper-parameter, and experiments were conducted to select the best value for each parameter.

- **Rank of factor matrices:** This hyper-parameter is concerned with the size of the embeddings for each of the factor matrices. For simplicity purpose we consider $d_e = d_r = d_o$. In Figure 6.7, we choose five different values for the rank and calculate the hits @10 score for each model and look for the rank, which gives the highest hits @10

score. We observe that at rank=100, we get the highest hits @10 score and reach the ceiling. Increasing the rank beyond 100 does not yield a higher hits @10 score.

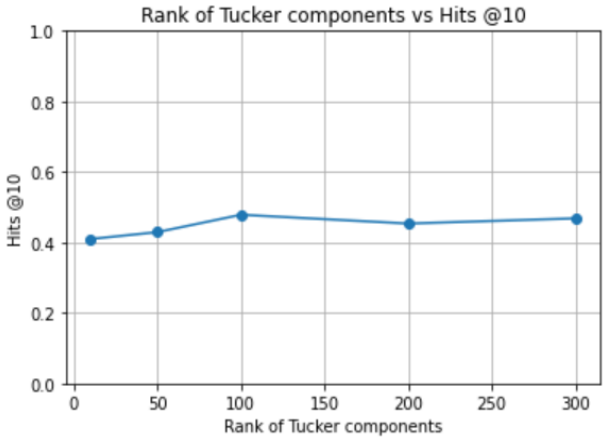


Figure 6.7: Choosing the rank of the factor matrices. The highest Hits @10 score is obtained for rank=100.

- **Number of epochs:** The number of epochs is a hyper-parameter that defines the number of times that the learning algorithm will work through the entire training data-set. One epoch means that each sample in the training data-set has had an opportunity to update the internal model parameters. An epoch is comprised of one or more batches. For example, as above, an epoch that has one batch is called the batch gradient descent learning algorithm. To determine the optimal number of epochs, we conducted experiments with different values and looked for points where the model reached convergence. Figure 6.8 shows that choosing the number of epochs as 200 is ideal for our experiments.

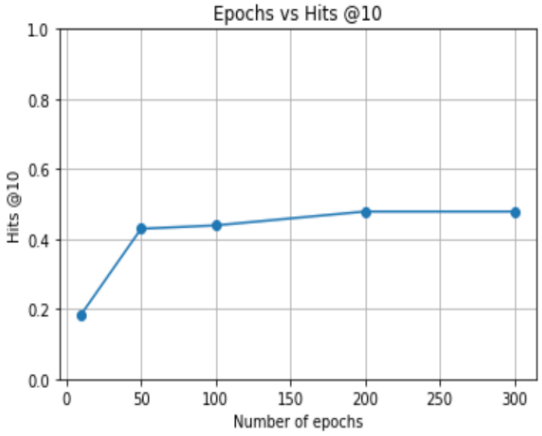


Figure 6.8: Choosing number of epochs. Training the model till 200 epochs leads to having the highest Hits@10 score, beyond 200 epochs we observe a plateau.

- **Batch size:** The batch size is a hyper-parameter that defines the number of samples to

work through before updating the internal model parameters. A training data-set can be divided into one or more batches. In [52], the authors state that that smaller batch sizes yield lower loss function values, which is a common belief among the researchers. We choose a batch size of 64 for our experiments.

- **Learning rate:** The learning rate is a hyper-parameter that controls how much to change the model in response to the estimated error each time the model weights are updated. Choosing the learning rate is challenging as a value too small may result in a long training process that could get stuck, whereas a value too large may result in learning a sub-optimal set of weights too fast or an unstable training process. We consider the learning rates mentioned in [35] for our experiments. Although, we settled for the learning rate $\{5 - e3\}$.

- **Regularization parameter λ_i:** Since we consider L2 loss for regularization, we keep the λ values on the lower side to avoid the norms to be very high. The λ value selected was $\{5e - 5\}$.

To summarize the above section, Table 6.8 shows the chosen hyper-parameters for our experiments.

$d_e = d_r = d_o$	# of epochs	batch size	lr	λ_i
100	200	64	5e-3	5e-5

Table 6.8: Model Hyper-parameters selected based on the experiments.

6.4.8 Inference stage: Classification metrics

The link prediction model helps us identify the right label, given a target word and context word. In Section 4.4, we present an approach for predicting the label for a target word by considering context words in the sentence. With our experiments on the unlabeled data-set, we not only consider all the context words but also restrict the number of context words based on a window size while making predictions. For our experiments we consider window size: $\{1, 2, 5, None\}$ where *1* refers to considering 1 word to the left and right of a target word and *None* refers to considering all the context words in a sentence. For evaluation, create a test set consisting of 50 randomly sampled listings from the *Consumer Electronics* category. We choose the model which provides the best link prediction results; hence the model *Using Category information* is used for evaluation. For each product description, we construct the labels manually. We calculate precision @k and recall @k which helps us calculate F1 @k, ($k \in 3, 5, 10, 15$) for the labels. Table 8.1 contains the results for the labels for different window sizes used in the inference stage.

In Figure 6.9, 6.10, 6.11 and 6.12 we present the F1 score @k for the labels for k values 3,5,10, and 15 respectively. Interestingly, we observe across all the results that considering 1 or 2 surrounding words around the target words provides better results than considering all the surrounding words in the sentence. We also observe that some of the labels like *color, device, and condition* do not show up in the top 3 predictions for the model. Overall, we want to improve the F1 score @k ($k \in 3, 5$) as we want our model to make the right predictions with a high confidence score.

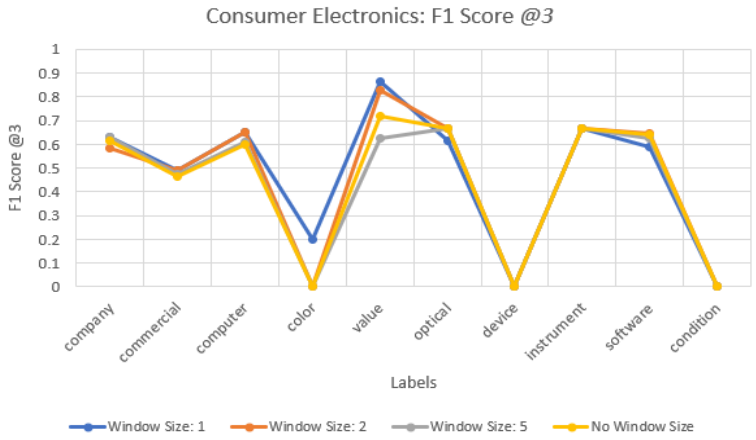


Figure 6.9: F1 Score @3 for the labels. We observe certain labels like *color, device and condition* are not showing up in the top 3 predictions. Overall we see window size 1 and 2 are providing the highest F1 score.

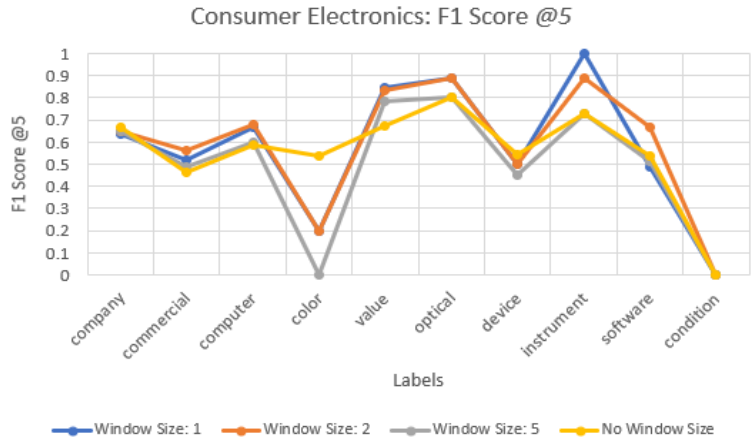


Figure 6.10: F1 Score @5 for the labels. We observe certain labels like *color, condition* are not showing up in the top 5 predictions. Overall we see window size 1 and 2 are providing the highest F1 score.

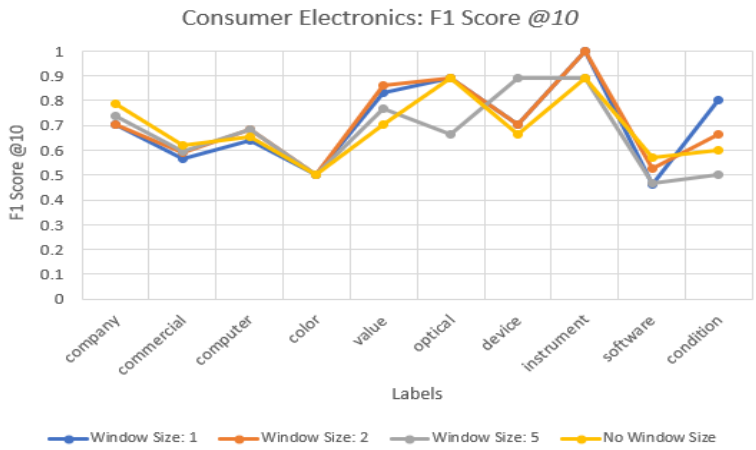


Figure 6.11: F1 Score @10 for the labels. Restricting the window size to 1 and 2 is showing the best results.

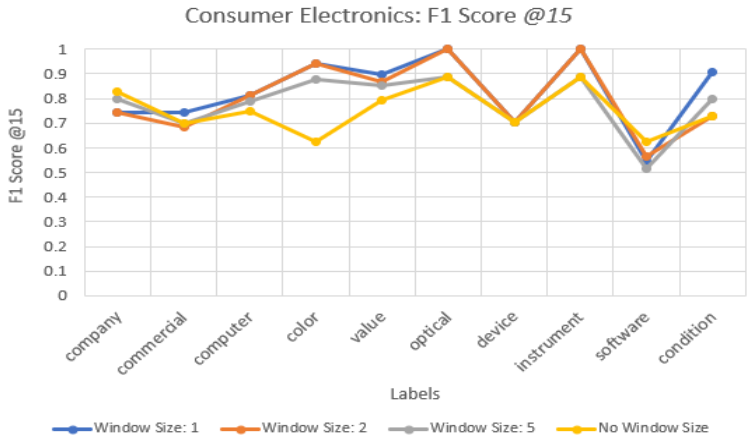


Figure 6.12: F1 Score @15 for the labels. Clearly, restricting the window size 1 and 2 is providing the highest F1 score for most labels.

6.5 Impact on Performance by Scaling Data

In this section, we answer RQ3 by analyzing the model's behavior when data volume is scaled up. Input data for the link prediction model is constructed using heuristics, which introduces certain irrelevant hypernyms while constructing triples. We refer to these irrelevant hypernyms as the noise in our data-set. We analyze the impact on the model's performance by increasing the volume of input data, thereby adding noise as well. To conduct this experiment, we consider product descriptions from the domains of consumer electronics, furniture, footwear, and cars. Table 6.9 shows the categories and the number of product descriptions that we considered for this experiment.

For this experiment, the volume of data is increased by adding product descriptions, one category at a time. To avoid bias in the order in which the product descriptions are added,

Category information	# of product descriptions
Consumer Electronics	967
Furniture	937
Footwear	1246
Cars	5981
Total	**9131**

Table 6.9: Categories and the number of product description. For answering RQ3 we consider product descriptions from 4 different domains.

we create combinations of {1,2,3,4} categories of product descriptions and train the model. By doing so, we provide an unbiased result of models that are trained. The test set for each category is manually constructed by randomly sampling 50 product descriptions. In Figure 6.13, we present the results obtained from our experiments. In the figure, the x-axis corresponds to the number of categories of product descriptions we consider in the model training process, and the y-axis refers to hits @10 metric for link prediction. Each data point corresponds to the model's performance for a certain number of distinct categories of product descriptions. For example, we consider four different categories of product descriptions; hence we see 4 data points where the number of categories is equal to 1; we create combinations of 2 categories for the number of categories = 2. Hence we see 6 data points (${}^4C_2 = 6$) and so on.

We observe a linear decay in the model's performance as more data is added for training. We can see the effect of more noise on the performance of the model. However, we do not observe a drastic drop in the performance of the model, which tells us that the Tucker decomposition model is able to learn and predict true triples despite the presence of noise. This result is relevant for answering the question of whether to use a single model for all the product descriptions or one model per category of product descriptions.

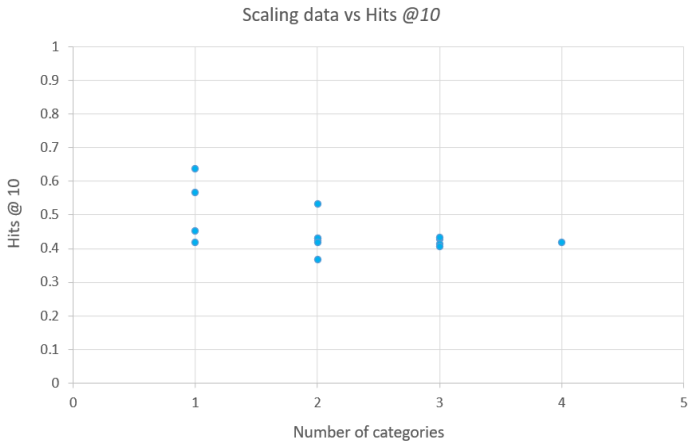


Figure 6.13: Impact on performance of the model by scaling up data. We observe a linear decay in the performance of model as the volume of training data is scaled up.

6.6 Discussion

The experiments set out with the aim of answering the research questions and understanding the behavior of the Tucker decomposition model. RQ1 sought to determine if the Tensor factorization model is suitable for extracting attributes/labels from product descriptions. The results in Table 6.1 indicate that the Tucker decomposition model with non-negative constraints performs better than the classification models mentioned in [10]. This makes our approach achieve state-of-the-art results for the labeled data-set. We observe that our model has higher precision than recall. This means that our model is picky when it comes to classifying attributes as high precision indicates that when the model predicts a particular label, it is usually right; however, it also misses true labels. In summary, these results indicate that knowledge graph completion is a suitable model for predicting attributes from product descriptions, thereby answering RQ1.

The main challenge in RQ2 was to achieve a similar quality of data as the labeled data-set used to answer RQ1. In Table 6.10, we summarize the results obtained from all the experiments conducted on the unlabeled data-set while answering RQ2. We presented several strategies for modifying the input data and the model to improve the system's performance. From the table, we see that using non-negative constraints, orthogonal regularizer, and category information provides the best link prediction results.

One unanticipated finding was the model performance by using pre-trained embeddings to construct triples. We hoped to find more semantically relevant hypernyms; however, we could not achieve results better than using category information. Since we used embeddings trained on Wikipedia corpus, which is also what Babelnet is built on, using pre-trained embeddings led us to identify a similar set of hypernyms that were being constructed through the heuristics. Hence, we could not see an improvement in the performance by using pre-trained embeddings.

The results obtained by using windowing while constructing the triples differ from the results of using category information by less than 1%. Interestingly, we observe that the number of triples for training reduces by one-third when using windowing. The effect of the different approaches on the number of triples can be seen in Figure 6.14. We see that by using category information, and windowing approach with window size 5, and 2, we see the number of triples reduce by a factor of $\sim$ 29%, 47%, and 58% respectively compared to the baseline approach.

The number of triples for training the model is directly proportional to the time taken for a model to train. In Figure 6.15, we see the time taken for a model to train based on the approach taken for constructing input data. We see that by applying each strategy, the training time reduces as compared to the baseline approach, with window size 2 having the least training time. This makes sense as the number of triples for window size 2 is also the least amongst all the candidates in Figure 6.14. From Figure 6.14, 6.15, we can conclude that the windowing approach reduces the number of triples, which in turn reduces the time taken to train a model while maintaining the link prediction performance. These results will be useful to decide which approach to consider while constructing the input data when constructing triples for a large volume of data.

In Table 6.10 and Figure 6.16 we have summarized the link prediction results obtained from experiments on unlabeled data-set. We have improved from our baseline approach by 7%, 8%, 15%, 12% in hits @{3,5,10,15} respectively. From all of the suggested approaches using category information along with non-negative Tucker decomposition and orthogonal regularizer has provided us with the best results. However, if we consider large volumes of data, then the windowing approach should be considered a candidate.

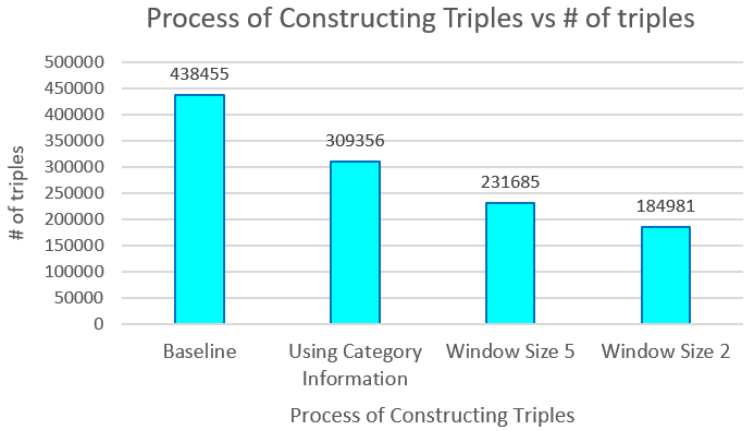


Figure 6.14: Process of Constructing Triples vs Number of triples. We see that by using category information, windowing approach with window size 5 and 2 we see the number of triples reduce by a factor of $\sim$ 29%, 47% and 58% respectively compared to baseline approach.

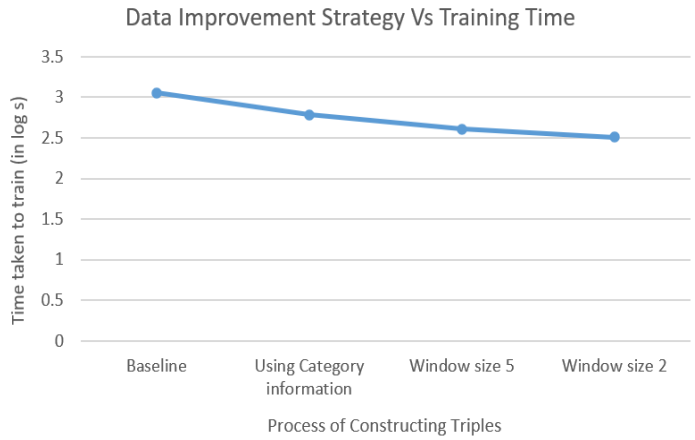


Figure 6.15: Impact on model training time by improving input data. We see that by applying each strategy the training time reduces as compared to the baseline approach.

Approach						Results			
Baseline	NTD	Orthogonal Regularizer	Category Information	Windowing	Pre-trained Embeddings	Hits @15	Hits @10	Hits @5	Hits @3
✓	-	-	-	-	-	0.4146	0.3219	0.2487	0.1756
-	✓	-	-	-	-	0.4780	0.3707	0.2634	0.2048
-	✓	✓	-	-	-	0.5365	0.4439	0.2975	0.2243
-	✓	✓	✓	-	-	**0.5327**	**0.4785**	**0.3216**	**0.2447**
-	✓	✓	✓	✓	-	0.5268	0.4536	0.3171	0.2447
-	✓	✓	✓	✓	✓	0.4439	0.3804	0.2878	0.2341

Table 6.10: Summary of Link Prediction results. Using category information along with the modifications on the model yield the best results.

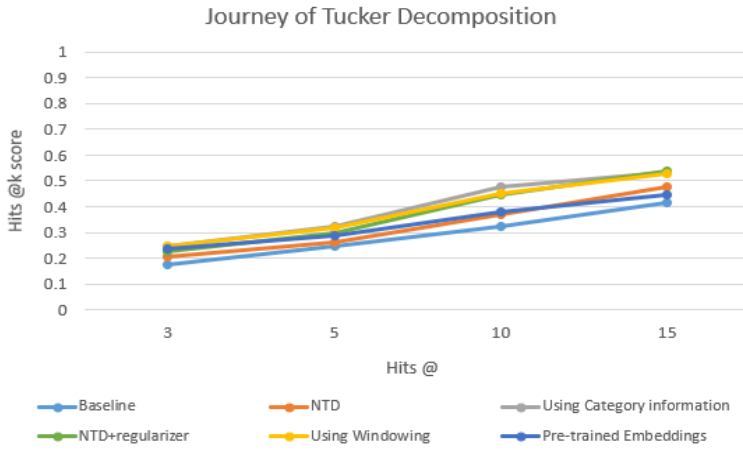


Figure 6.16: Journey of Tucker decomposition. Using category information provides the best results.

Chapter 7

Conclusion

In this thesis, we proposed a semi-supervised solution for converting unstructured classifieds product descriptions to a structured format by understanding the semantic meaning of words by extracting the relationship between them.

We proposed a novel approach for constructing a knowledge graph from product descriptions and using the context words for predicting attribute/label for target words. We demonstrated that Tensor Factorization is a suitable approach for predicting missing attributes/labels. Our approach leads to achieving state-of-the-art performance on the public data-set that has labeled data.

We proposed a novel approach for constructing a knowledge graph for unlabeled product descriptions by using Babelnet to assign attribute/label to words. When using the Tensor Factorization technique, the Tucker decomposition model was considered as the baseline approach. We proposed strategies for improving the performance of the system by modifying input data and the model. On applying the proposed strategies, we achieved an increase in the performance by 7%,8%,15%,12% in hits @{3,5,10,15} respectively compared to the baseline approach. When constructing the knowledge graph, we evaluated the influence of window size for considering context words on solution performance. We found that a window size of {1,2} achieves the best results for predicting an attribute from product descriptions, and we do not need to consider all the surrounding words in the sentence. However, the window size may vary across different languages as it depends on the structure for that particular language.

Finally, we demonstrated that the solution performs the best when trained on a product description for a single category of products. When training data is extended with multiple product categories, we observed a linear decay in the model's performance.

7.1 Future Work

The evaluation of the developed solution demonstrated that the results of the experiments on the unlabeled data-set are not at par with the results obtained on the labeled data-set.

That is likely caused by the insufficient quality of labels that we create when constructing the knowledge graph. The future work shall investigate alternative approaches for creating the knowledge graph.

In the scope of our thesis, we have considered product descriptions in the English language. Further research should be undertaken to investigate if our proposed approach for constructing the knowledge graph and inference stage pipeline is suitable for other languages. The authors in [53] and [54] state that knowledge graph completion models are uncalibrated. It means that the confidence score obtained for a triple being true or false does not represent the true confidence score. The authors propose techniques like Platt Scaling [55] and Isotonic regression [56] to calibrate knowledge graph completion models. Further studies and research to make our knowledge graph completion model approach reliable, it is recommended to pursue the topic of probability calibration.

Bibliography

[1] Leonard Kaufman and Peter Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Sept. 2009. ISBN: 9780470317488.

[2] David Blei, Andrew Ng, and Michael Jordan. "Latent Dirichlet Allocation". In: *Journal of Machine Learning Research* 3 (May 2003), pp. 993–1022. DOI: 10.1162/jmlr.2003.3.4-5.993.

[3] Dan Jurafsky. *Information Extraction*. https://web.stanford.edu/~jurafsky/slp3/17.pdf. 2019.

[4] *Named Entity Recognition*. URL: https://www.paperswithcode.com/task/named-entity-recognition-ner.

[5] *NLP's ImageNet moment has arrived*. URL: https://ruder.io/nlp-imagenet/.

[6] Maximilian Nickel et al. "A Review of Relational Machine Learning for Knowledge Graphs". In: *Proceedings of the IEEE* 104 (Dec. 2015). DOI: 10.1109/JPROC.2015.2483592.

[7] Tamara Kolda and Brett Bader. "Tensor Decompositions and Applications". In: *SIAM Review* 51 (Aug. 2009), pp. 455–500. DOI: 10.1137/07070111X.

[8] R Ghani et al. "Text mining for product attribute extraction". In: *SIGKDD* (2006), pp. 41–48.

[9] David Hand and Keming Yu. "Idiot's Bayes: Not So Stupid after All?" In: *International Statistical Review* 69 (May 2007), pp. 385–398. DOI: 10.1111/j.1751-5823.2001.tb00465.x.

[10] Sidorov Mikhail. "Attribute extraction from eCommerce product descriptions". In: (2018).

[11] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: (1995).

[12] Alexey Natekin1 and Alois Knoll. "Gradient boosting machines, a tutorial". In: (2013).

[13] John Lafferty D., Andrew McCallum, and C. Fernando Pereira N. "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data". In: *Proceedings of the Eighteenth International Conference on Machine Learning* (2001), pp. 282–289.

[14] Ajinkya More. "Attribute Extraction from Product Titles in eCommerce". In: (2016).

[15] Lidong Bing, Tak-Lam Wong, and Wai Lam. "Unsupervised Extraction of Popular Product Attributes from Web Sites". In: (2016).

[16] Abdi-Hakin Dirie A. "Extracting Diverse Attribute-Value Information from Product Catalog Text via Transfer Learning". In: (2016).

[17] Guineng Zheng et al. "OpenTag: Open Attribute Value Extraction from Product Profiles". In: *KDD* (2018).

[18] Burr Settles. *Active Learning Literature Survey*. http://burrsettles.com/pub/settles.activelearning.pdf. 2010.
[19] Santosh Raju and Praneeth Shishtla. "A Graph Clustering Approach to Product Attribute Extraction". In: *4th Indian International Conference on Artificial Intelligence* (2009).
[20] Santosh Raju, Prasad Pingali, and Vasudeva Varma. "An Unsupervised Approach to Product Attribute Extraction". In: *ECIR* (2009).
[21] Thorvald Sørensen and Lee Dice Raymond. *Dice Coefficient*. https://enacademic.com/dic.nsf/enwiki/5165495.
[22] *Group Average Hierarchical Clustering*. https://nlp.stanford.edu/IR-book/html/htmledition/group-average-agglomerative-clustering-1.html.
[23] Duangmanee Putthividhya and Junling Hu. Boot. "Bootstrapped named entity recognition for product attribute extraction". In: *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing* (2011), pp. 1557–1567.
[24] B. Taskar et al. "Link Prediction in Relational Data". In: *NIPS*. 2003.
[25] Lise Getoor and Christopher P. Diehl. "Link Mining: A Survey". In: *SIGKDD Explor. Newsl.* 7.2 (Dec. 2005), pp. 3–12. ISSN: 1931-0145. DOI: 10.1145/1117454.1117456. URL: https://doi.org/10.1145/1117454.1117456.
[26] Liang Yao, Chengsheng Mao, and Yuan Luo. *KG-BERT: BERT for Knowledge Graph Completion*. Sept. 2019.
[27] Quan Wang et al. *CoKE: Contextualized Knowledge Graph Embedding*. 2020. arXiv: 1911.02168 [cs.AI].
[28] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: (2018).
[29] Zhengzheng Xing, Jian Pei, and Eamonn Keogh. "A Brief Survey on Sequence Classification". In: 12.1 (Nov. 2010), pp. 40–48. ISSN: 1931-0145. DOI: 10.1145/1882471.1882478. URL: https://doi.org/10.1145/1882471.1882478.
[30] Yehuda Koren, Robert Bell, and Chris Volinsky. *MATRIX FACTORIZATION TECHNIQUES FOR RECOMMENDER SYSTEMS*. 2009.
[31] R. Zhang et al. "Collaborative Filtering for Recommender Systems". In: *2014 Second International Conference on Advanced Cloud and Big Data*. 2014, pp. 301–308.
[32] Maximilian Nickel and Volker Tresp. "Tensor Factorization for Multi-relational Learning". In: Sept. 2013, pp. 617–621. DOI: 10.1007/978-3-642-40994-3_40.
[33] Mehran Kazemi and David Poole. "SimplE Embedding for Link Prediction in Knowledge Graphs". In: (Feb. 2018).
[34] Ledyard Tucker. "Some Mathematical Notes on Three-Mode Factor Analysis". In: *Psychometrika* 31 (Feb. 1966), pp. 279–311. DOI: 10.1007/BF02289464.
[35] Ivana Balažević, Carl Allen, and Timothy Hospedales. *TuckER: Tensor Factorization for Knowledge Graph Completion*. Jan. 2019.
[36] *Best Buy E-Commerce NER Dataset*. URL: https://www.kaggle.com/dataturks/best-buy-ecommerce-ner-dataset/home.
[37] *OLX website*. URL: https://www.olx.com/.
[38] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. "On the Best Rank-1 and Rank Approximation of Higher-Order Tensors". In: *SIAM J. Matrix Anal. Appl.* 21.4 (Mar. 2000), pp. 1324–1342. ISSN: 0895-4798. DOI: 10.1137/S0895479898346995. URL: https://doi.org/10.1137/S0895479898346995.

[39] Pieter Kroonenberg and Jan De Leeuw. "Principal Component Analysis of Three-mode Data by Means of Alternating Least Squares Algorithm". In: *Psychometrika* 45 (Mar. 1980), pp. 69–97. DOI: 10.1007/BF02293599.
[40] T Dettmers et al. "Convolutional 2D knowledge graph embeddings". In: Feb. 2018.
[41] Roberto Navigli and Simone Ponzetto. "BabelNet: Building a Very Large Multilingual Semantic Network". In: Sept. 2010, pp. 216–225.
[42] *Hypernymy and Hyponymy*. https://en.wikipedia.org/wiki/Hyponymy_and_hypernymy. Accessed: 2020-02-17.
[43] C. Fellbaum. "WordNet : an electronic lexical database". In: *Language* 76 (2000), p. 706.
[44] *Regularization*. URL: https://chunml.github.io/ChunML.github.io/tutorial/Regularization/.
[45] Tomas Mikolov et al. "Distributed Representations of Words and Phrases and their Compositionality". In: *NIPS* (2013).
[46] Jeffrey Pennington, Richard Socher, and Christopher Manning D. "GloVe: Global Vectors for Word Representation". In: *EMNLP* (2014).
[47] Matthew Peters E. et al. "Deep contextualized word representations". In: (2018).
[48] Sergey Bartunov et al. "Breaking Sticks and Ambiguities with Adaptive Skip-gram". In: (2015).
[49] Adam Paszke et al. "Automatic differentiation in PyTorch". In: (2017).
[50] S. Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *ArXiv* abs/1502.03167 (2015).
[51] Alexander Panchenko. "Best of Both Worlds: Making Word Sense Embeddings Interpretable". In: May 2016.
[52] Xin Qian and Diego Klabjan. *The Impact of the Mini-batch Size on the Variance of Gradients in Stochastic Gradient Descent*. 2020. arXiv: 2004.13146 [math.OC].
[53] Pedro Tabacof and Luca Costabello. *Probability Calibration for Knowledge Graph Embedding Models*. Dec. 2019.
[54] Tara Safavi, Danai Koutra, and Edgar Meij. *Improving the Utility of Knowledge Graph Embeddings with Calibration*. Apr. 2020.
[55] John Platt. "Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods". In: *Adv. Large Margin Classif.* 10 (June 2000).
[56] Wei Wu, Michael Woodroofe, and Graciela Mentz. "Isotonic regression: Another look at the changepoint problem". In: *Biometrika* 88 (Sept. 2001), pp. 793–804. DOI: 10.1093/biomet/88.3.793.

Chapter 8

Appendix

8.1 Inference Stage Results

Label	k value	Inference window size	TP	FP	FN	Precision @k	Recall @k	F1 @k
company	3	1	12	8	6	0.6	0.6667	0.6316
commercial	3	1	12	20	5	0.375	0.7059	0.4898
computer	3	1	15	13	3	0.5357	0.8333	0.6521
color	3	1	1	0	8	1	0.1111	0.2
value	3	1	22	0	7	1	0.7587	0.8627
optical	3	1	4	5	0	0.4444	1	0.6153
device	3	1	0	0	24	-	0	-
instrument	3	1	4	4	0	0.5	1	0.6667
software	3	1	10	9	5	0.5263	0.6667	0.5882
condition	3	1	0	2	6	0	0	-
company	5	1	13	10	5	0.5652	0.7222	0.6341
commercial	5	1	12	17	5	0.4138	0.7059	0.5217
computer	5	1	15	12	3	0.5556	0.8333	0.6667
color	5	1	1	0	8	1	0.1111	0.2
value	5	1	22	1	7	0.9565	0.7586	0.8461
optical	5	1	4	1	0	0.8	1	0.8889
device	5	1	8	0	16	1	0.3333	0.5
instrument	5	1	4	0	0	1	1	1
software	5	1	11	19	4	0.3667	0.7333	0.4889
condition	5	1	0	1	6	0	0	-
company	10	1	13	6	5	0.6842	0.7222	0.7027
commercial	10	1	13	16	4	0.4483	0.7647	0.5652
computer	10	1	15	14	3	0.5172	0.8333	0.6383
color	10	1	3	0	6	1	0.3333333333	0.5
value	10	1	22	2	7	0.9167	0.7586	0.8301

Label	k value	Inference window size	TP	FP	FN	Precision @k	Recall @k	F1 @k
optical	10	1	4	1	0	0.8	1	0.8889
device	10	1	13	0	11	1	0.5417	0.7027
instrument	10	1	4	0	0	1	1	1
software	10	1	15	35	0	0.3	1	0.4615
condition	10	1	4	0	2	1	0.6667	0.8
company	15	1	16	9	2	0.64	0.8889	0.7441
commercial	15	1	13	5	4	0.7222	0.7647	0.7429
computer	15	1	15	4	3	0.7895	0.8333	0.8108
color	15	1	8	0	1	1	0.8889	0.9412
value	15	1	26	3	3	0.8966	0.8966	0.8966
optical	15	1	4	0	0	1	1	1
device	15	1	13	0	11	1	0.5417	0.7027
instrument	15	1	4	0	0	1	1	1
software	15	1	15	25	0	0.375	1	0.5455
condition	15	1	5	0	1	1	0.8333	0.9091
company	3	2	12	11	6	0.5217	0.6667	0.5853
commercial	3	2	12	20	5	0.375	0.7059	0.4898
computer	3	2	15	13	3	0.5357	0.8333	0.6522
color	3	2	0	0	9	-	0	-
value	3	2	22	2	7	0.9167	0.7586	0.8302
optical	3	2	4	4	0	0.5	1	0.6667
device	3	2	0	0	24	-	0	-
instrument	3	2	4	4	0	0.5	1	0.6667
software	3	2	10	6	5	0.625	0.6667	0.6452
condition	3	2	0	2	6	0	0	-
company	5	2	13	9	5	0.5909	0.7222	0.65
commercial	5	2	13	16	4	0.4483	0.7647	0.5652
computer	5	2	15	11	3	0.5769	0.8333	0.6818
color	5	2	1	0	8	1	0.1111	0.2
value	5	2	23	3	6	0.8846	0.7931	0.8363
optical	5	2	4	1	0	0.8	1	0.8889
device	5	2	8	0	16	1	0.333333333	0.5
instrument	5	2	4	1	0	0.8	1	0.8889
software	5	2	11	7	4	0.6111	0.7333	0.6667
condition	5	2	0	1	6	0	0	-
company	10	2	13	6	5	0.6842	0.7222	0.7027
commercial	10	2	13	14	4	0.4815	0.7647	0.5909
computer	10	2	15	11	3	0.5769	0.8333	0.6818
color	10	2	3	0	6	1	0.3333	0.5
value	10	2	25	4	4	0.8621	0.8621	0.8621
optical	10	2	4	1	0	0.8	1	0.888888889
device	10	2	13	0	11	1	0.5417	0.7027
instrument	10	2	4	0	0	1	1	1

Label	k value	Inference window size	TP	FP	FN	Precision @k	Recall @k	F1 @k
software	10	2	15	27	0	0.3571	1	0.5263
condition	10	2	3	0	3	1	0.5	0.6667
company	15	2	16	9	2	0.64	0.8889	0.7442
commercial	15	2	13	8	4	0.6190	0.7647	0.6842
computer	15	2	15	4	3	0.7894	0.8333	0.8108
color	15	2	8	0	1	1	0.8889	0.9412
value	15	2	26	5	3	0.8387	0.8966	0.8667
optical	15	2	4	0	0	1	1	1
device	15	2	13	0	11	1	0.5417	0.7027
instrument	15	2	4	0	0	1	1	1
software	15	2	15	23	0	0.3947	1	0.5660
condition	15	2	4	1	2	0.8	0.6667	0.7272
company	3	5	12	8	6	0.6	0.6667	0.6316
commercial	3	5	10	15	7	0.4	0.5882	0.4761
computer	3	5	15	16	3	0.4839	0.8333	0.6122
color	3	5	0	0	9	-	0	-
value	3	5	10	7	5	0.5882	0.6667	0.625
optical	3	5	4	4	0	0.5	1	0.6667
device	3	5	0	0	24	-	0	-
instrument	3	5	4	4	0	0.5	1	0.6667
software	3	5	10	7	5	0.5882	0.6667	0.625
condition	3	5	0	1	6	0	0	-
company	5	5	13	9	5	0.5909	0.7222	0.65
commercial	5	5	10	14	7	0.4167	0.5882	0.4878
computer	5	5	15	17	3	0.46875	0.8333	0.6
color	5	5	0	0	9	#DIV/0!	0	-
value	5	5	22	5	7	0.8148	0.7586	0.7857
optical	5	5	4	2	0	0.6667	1	0.8
device	5	5	7	0	17	1	0.2917	0.4516
instrument	5	5	4	3	0	0.5714	1	0.7272
software	5	5	11	17	4	0.3929	0.7333	0.5116
condition	5	5	0	1	6	0	0	-
company	10	5	14	6	4	0.7	0.7778	0.7368
commercial	10	5	14	16	3	0.4667	0.8235	0.5957
computer	10	5	15	11	3	0.5769	0.8333	0.6818
color	10	5	3	0	6	1	0.3333	0.5
value	10	5	23	8	6	0.7419	0.7931	0.7667
optical	10	5	12	0	12	1	0.5	0.6667
device	10	5	4	1	0	0.8	1	0.8889
instrument	10	5	4	1	0	0.8	1	0.8889
software	10	5	14	31	1	0.3111	0.9333	0.4667
condition	10	5	2	0	4	1	0.3333	0.5
company	15	5	16	6	2	0.7272	0.8889	0.8

Label	k value	Inference window size	TP	FP	FN	Precision @k	Recall @k	F1 @k
commercial	15	5	14	9	3	0.6087	0.8235	0.7
computer	15	5	15	5	3	0.75	0.8333	0.7895
color	15	5	7	0	2	1	0.7778	0.875
value	15	5	26	6	3	0.8125	0.8966	0.8525
optical	15	5	4	1	0	0.8	1	0.8889
device	15	5	13	0	11	1	0.5417	0.7027
instrument	15	5	4	1	0	0.8	1	0.8889
software	15	5	15	28	0	0.3488	1	0.5172
condition	15	5	4	0	2	1	0.6667	0.8
company	3	None	12	9	6	0.5714	0.6667	0.6154
commercial	3	None	10	16	7	0.3846	0.5882	0.4651
computer	3	None	15	17	3	0.4687	0.8333	0.6
color	3	None	0	0	9	-	0	-
value	3	None	23	12	6	0.6571	0.7931	0.7188
optical	3	None	4	4	0	0.5	1	0.6667
device	3	None	0	0	24	-	0	-
instrument	3	None	4	4	0	0.5	1	0.6667
software	3	None	9	4	6	0.6923	0.6	0.6429
condition	3	None	0	1	7	0	0	-
company	5	None	14	10	4	0.5833	0.7778	0.6667
commercial	5	None	10	16	7	0.3846	0.5882	0.4651
computer	5	None	15	18	3	0.4545	0.8333	0.5882
color	5	None	0	0	9	0.4545	0.6667	0.5405
value	5	None	23	16	6	0.5897	0.7931	0.6765
optical	5	None	4	2	0	0.6667	1	0.8
device	5	None	9	0	15	1	0.375	0.5454
instrument	5	None	4	3	0	0.5714	1	0.7272
software	5	None	10	12	5	0.4545	0.6667	0.5405
condition	5	None	0	0	7	-	0	-
company	10	None	15	5	3	0.75	0.8333	0.7895
commercial	10	None	13	12	4	0.52	0.7647	0.6190
computer	10	None	15	13	3	0.5357	0.8333	0.6522
color	10	None	3	0	6	1	0.3333	0.5
value	10	None	24	15	5	0.6154	0.8276	0.7059
optical	10	None	4	1	0	0.8	1	0.8889
device	10	None	12	0	12	1	0.5	0.6667
instrument	10	None	4	1	0	0.8	1	0.8889
software	10	None	14	20	1	0.4118	0.9333	0.5714
condition	10	None	3	0	4	1	0.4286	0.6
company	15	None	17	6	1	0.7391	0.9444	0.8292
commercial	15	None	14	9	3	0.6087	0.8235	0.7
computer	15	None	15	7	3	0.6818	0.8333	0.75
color	15	None	7	0	2	0.4545	1	0.625

Label	k value	Inference window size	TP	FP	FN	Precision @k	Recall @k	F1 @k
value	15	None	27	12	2	0.6923	0.9310	0.7941
optical	15	None	4	1	0	0.8	1	0.8889
device	15	None	13	0	11	1	0.5417	0.7027
instrument	15	None	4	1	0	0.8	1	0.8889
software	15	None	15	18	0	0.4545	1	0.625
condition	15	None	4	0	3	1	0.5714	0.7272

Table 8.1: Inference Stage results: Classification Metrics. TP,FP,FN refers to True Positive, False Positive and False Negative respectively. The hyphen (-) indicates that specific metric cannot be calculated because of division by 0.