Bauhaus-Universität Weimar
Faculty of Media
Degree Programme Computer Science for Digital Media

Measuring the Effectiveness of Word-Embeddings for Facet Completion Tasks

Master's Thesis

Prem Kumar Tiwari
Born 24 July 1995, in Patna

Matriculation Number 119508

1. Referee: Prof. Dr. Benno Stein
2. Referee: Prof. Dr. Andreas Jakoby

Submission date: November 6, 2020

Declaration of Academic Honesty

Unless otherwise indicated in the text or references, this thesis is entirely the product of my own scholarly work.

Furthermore, I confirm that this is the first time that this thesis - in any form is presented to a supervising staff.

Weimar, October 25, 2020

. . . . . .

Prem Kumar Tiwari

"Except our own thoughts, there is nothing absolutely in our power"

- Rene Descartes

To my parents and brother, for their constant love and support.

Abstract

Faceted Search is a powerful paradigm for the exploration of document collections. In order to establish an effective faceted search system, a key component is the existence of relevant facets. Since the relevance of a facet depends on the information needed by the users, this thesis deals with support technology for user-generated facets. More specifically, we consider the task where a user initializes a custom facet with a few terms (up to three) referred to as *support* terms, and the goal is to find terms in a given vocabulary which fit to the facet of the user.

For the purpose of testing facet completion algorithms, we use WordNet - a labouriously hand coded lexical database, which has proven to be very expensive to build and maintain[1] . We analyze how existing state-of-the-art word-embeddings model fare against facets generation tasks. We do this, because word-embeddings model assimilate words that are similar to each other to a closer proximity in its vector space. Building upon this claim, we test the efficiency of word-embedding algorithms to recognize such facets on account of its proximity in its vector space.

Further, we suspect that these word-embedding models assimilate not only similar words but also associated words to a proximate space in word vector representation. For example, are cup and coffee related or are they just associated [1] ? This is so, because in training such word-embedding models are built upon architecture that does not consider the relative position of co-occurrence of words. So in this thesis, we seek to build such a word-embedding model that considers the relative position in the co-occurrence of words in the sequence while training, and check whether this method in training of word-embeddings model can achieve efficiency in the task of facets-generation.

[1] http://ai.stanford.edu/~rion/swn/

Prerequisites

FastText Word-Embeddings: fastText is another word embedding method that is an extension of the word2vec model. Instead of learning vectors for words directly, fastText represents each word as an n-gram of characters. So, for example, take the word, "*artificial*" with n=3, the fastText representation of this word is <*ar, art, rti, tif, ifi, fic, ici, ial, al*>, where the angular brackets indicate the beginning and end of the word. This helps capture the meaning of shorter words and allows the embeddings to understand suffixes and prefixes. Once the word has been represented using character n-grams, a skip-gram model is trained to learn the embeddings. This model is considered to be a bag of words model with a sliding window over a word because no internal structure of the word is taken into account. As long as the characters are within this window, the order of the n-grams doesn't matter[2].

Glove Word-Embeddings: **GloVe**, coined from Global Vectors, is a model for distributed word representation. The model is an unsupervised learning algorithm for obtaining vector representations for words. This is achieved by mapping words into a meaningful space where the distance between words is related to semantic similarity. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space. It is developed as an open-source project at Stanford[3].

Hypernym: A word with a broad meaning constituting a category into which words with more specific meanings fall. For example, colour is a hypernym of blue.

---

[2] "GloVe and fastText — Two Popular Word Vector Models in NLP." 3 Apr. 2019, https://cai.tools.sap/blog/glove-and-fasttext-two-popular-word-vector-models-in-nlp/. Accessed 1 Nov. 2020.

[3] "GloVe (machine learning) - Wikipedia." https://en.wikipedia.org/wiki/GloVe_(machine_learning). Accessed 1 Nov. 2020.

Hyponym: A hyponym is a word or phrase whose semantic field is included within that of another word, its Hypernym. For example, blue is a hyponym of colour.

Test for Hypernym and Hyponym: Hypernym is a broader structure, while hyponym is a specific category of a given hypernym. Hyponymy can be tested by substituting X and Y in the sentence "X is a kind of Y", where X is a hyponym and Y is a hypernym. For example, 'A chair is a kind of furniture' - makes sense, hence chair is a hyponym and furniture a hypernym.

Ontology: Ontology is commonly described as the study of the nature of things, and an *ontology* is a means of organizing and conceptualizing a domain of interest.[2]

Synsets:A set of one or more synonyms that are interchangeable in some context without changing the truth value of the proposition in which they are embedded[4].

Taxonomy: Taxonomy (general**)** is the practice and science of classification of things or concepts, including the principles that underlie such classification[5].

Word2Vec Word-Embeddings: Word2vec is a technique for natural language processing. The word2vec algorithm uses a neural network model to learn word associations from a large corpus of text. Once trained, such a model can detect synonymous words or suggest additional words for a partial sentence. As the name implies, word2vec represents each distinct word with a particular list of numbers called a vector. The vectors are chosen carefully such that a simple mathematical function (the cosine similarity between the vectors) indicates the level of semantic similarity between the words represented by those vectors[6].

WordNet: WordNet is an online lexical database designed for use under program control. English nouns, verbs, adjectives, and adverbs are organized into sets of synonyms, each representing a lexicalized concept. These sets of synonyms are called synsets. WordNet

[4] "synset - Wiktionary." https://en.wiktionary.org/wiki/synset. Accessed 1 Nov. 2020.
[5] "Taxonomy - Wikipedia." https://en.wikipedia.org/wiki/Taxonomy. Accessed 1 Nov. 2020.
[6] "Word2vec - Wikipedia." https://en.wikipedia.org/wiki/Word2vec. Accessed 1 Nov. 2020.

provides a more effective combination of traditional lexicographic and modern computing[7].

Word-Vectors: The word-vectors are assigned specific vectors based on the context in which the words appear. It states that words that co-occur, will be mapped to a proximate vector space. The number of dimensions used to represent a word in terms of vector space determines the word's distributed weight across these dimensions. To put simply, each dimension represents a meaning and the word's numerical value in that dimension captures the closeness of its association to the meaning[represented by that dimension].

[7] "WordNet: a lexical database for English - ACM Digital Library." https://dl.acm.org/doi/abs/10.1145/219717.219748. Accessed 4 Nov. 2020.

Table of Contents

Acknowledgement

First and foremost, I would like to thank my supervisor Tim Gollub for his support during the creation of this thesis. This work would not be possible without his unwavering guidance. A man of great inner strength, composure and character.

I would also like to extend my sincere thanks to Prof. Dr. Benno Stein for accepting my work under his supervision.

I would extend my sincere thanks to Prof. Dr. Andreas Jakoby for his consideration on evaluation of my thesis

Special Thanks to Ms. Anne Peter for her constant help, especially in the month of July. Last but by no means least, my sincere thanks go to my friends Sebastián Laverde Alfonso, Jawad Ghori, Ankit Satapute, Julia Schneider and Annika Wappelhorst for never hesitating to help me in times of need.

1. Introduction

Facet generation is the process of suggesting entities or items which are in the vicinity of a given item. For example, if the given item is “Siberian Husky”, the suggested items or facet term recommendation could be “Eskimo dog”, “Tibetan mastiff”, “Collie”, “Schipperke” etc. The suggested facets based on the given item share that all these dogs belong to the same category or facet i.e. “Working Dogs”. In this manner, facets generation works. Suggestions could be anything, for example, “Pomeranian” could also be suggested but this breed of dog is not closer to the suggested ones. Hence, in facet generation tasks, this closeness serves as an important parameter to govern what is suggested. This aspect of closeness should be automatically inferred to facilitate users navigation. The Feddoul, L. et al. [13] has already pointed out that manual facet predefinition is often inappropriate and apt choosing among facets is virtually impossible without algorithmic support.

Existing state-of-the-art algorithm “word- embeddings” uses the co-occurrence of words as the principle to map them to proximate vector space. This concept has revolutionized many applications that use these representations of words as their core [6]. These effects are very prominent in fields of sentiment analysis [7], information retrieval [8] [9], recommender systems [10] [11] and other Natural Language Processing (NLP) tasks. The promise of these word-embedding algorithms to have similar representation in a vector space for words that are similar to each other[8] could use the algorithmic support of word-embeddings for automatic generation of facets given a *support*, by looking in the proximate vector space of the *support*. So, using this as a core idea, we intend to measure the effectiveness of word-embeddings models for the facets generation tasks.

---

[8] "What Are Word Embeddings for Text?." 11 Oct. 2017, https://machinelearningmastery.com/what-are-word-embeddings/. Accessed 14 Oct. 2020.

For the purpose of evaluating efficiency by word-embedding models we need a dataset that contains a group of lists of similar facets/lemmas. We shall generate such a dataset using the lexical database WordNet for english language. This dataset contains lists of similar facets/lemmas and is generated by traversing the acyclic graph [3] of the WordNet structure.

We shall first experiment, how the existing state-of-the-art word embeddings models can effectively recognize similar facets by pitting these similar facets against random words. The three existing state-of-the-art word-embeddings models that we use in this thesis are:

1. Word-2-Vec Model
2. Glove Model
3. fastText Model

For each of these models, we shall analyze the similarity distribution exhibited to understand their efficiency in the facets generation tasks.

Although, word embeddings give us a way to use an efficient, dense representation in which similar words have a similar encoding[9], word-embeddings themselves are trained on the principle that the words that co-occur, share greater proximity than words that do not co-occur. This underlying logic ascertains that words that co-occur also have a similar encoding. Hence, we suspect that these word-embedding models have similar encoding for not only similar words but also for words that are associated or co-occur. This reasoning accounts for blurring of the boundary between related and associated words. For example, are cup and coffee related or are they just associated [1] ? This reasoning prompts us to draw a stricter boundary that facilitates the purport of word-embeddings for facet generation tasks.

On a closer look in the training of these word-embeddings models, we observe that relevance of the position with respect to the co-occurrence is not considered. For example, consider the following sentence:

[9] "Word embeddings | TensorFlow Core." 24 Sep. 2020, https://www.tensorflow.org/tutorials/text/word_embeddings. Accessed 14 Oct. 2020.

"I eat *apples* after working out."

The relative position of *eat* with respect to *apples* is one before it and anything that can be substituted for apples, probably also belong to the eatable category. Using this reasoning, if we could train a word-embeddings model that accounts for *positional co-occurrence* among words, we could also establish a stricter boundary between related and associated words by allowing similar words to have more similar encoding than the associated words. And using this logic, we can have a word-embeddings model that is more suited to the facet-generation tasks.

This can further be substantiated with reasoning during the training process. For example, if X or Y ($X = Y = W_i$) generate the same positional context $\{W_{i-2}, W_{i-1}, W_{i+1}, W_{i+2}\}$ during training, then X and Y should have similar encoding in the word vector representation. This models our notion in the example presented above- anything that can be substituted for apples(X) also belong to the same category e.g. oranges(Y).

Using this logic as the central concern, we shall train a word-embeddings model by modifying the architecture of the skip-gram Word2Vec model such that it accommodates positional information while training word-embeddings. We shall then test if this notion of using positional co-occurrence while training word-embeddings model facilitates the purport of enriching similar encoding for similar words/facets, which in-turn could propel the algorithmic support of word-embeddings model for the facet-generation tasks.

1.1 Structural Overview

The structure of the thesis is as follows.

Chapter 2 gives more details about the basic foundations of related tasks and methods. It presents challenges with the technologies we shall be working with and also the literature relevant to the thesis.

Chapter 3 mentions the procedure in which we prepare the dataset from traversing the acyclic graph of the WordNet. From this dataset extracted, we further mould the data according to relevant experimentation we shall be working with.

Chapter 4 provides insights about the different evaluation procedures that we shall be doing on various state-of-the-art word-embeddings models and analyzes why such word-embeddings are not good enough to extend their algorithmic support for facets generation tasks. Hence, further shows a need to design word-embeddings models for the tasks of facets generation.

Chapter 5 explains how we try to accommodate positional co-occurrence of words in training a word-embeddings model and then compare this model with a classic skip gram model and analyzes whether this new model fares well for the tasks of facets generation in the comparison.

Chapter 6 concludes our thesis and discusses future possible areas of works.

2. Challenges and Related tasks

For measuring the effectiveness of the word-embeddings model for facet generation tasks, we need some structures that contain facets of similar groups together. Since, there exists no such groups of facets, we shall be constructing one. To construct such a grouping of facets, we shall generate a list of lists, in which every list contains a group of facets that share a similar concept among themselves. To obtain these groups, we shall turn towards existing knowledge structures in the Natural Language Processing (NLP) and Information Retrieval (IR) domains.

Medelyan, Olena, et al. [2] has categorised the representation of knowledge structures into three categories - Term Lists, Term Hierarchies and Semantic Databases. They have not set any boundaries for their distinction and hence their definitions overlap. For the purpose of generating a group of facets in form of list of lists, we shall use WordNet, which lies in the middle of the spectrum Term Hierarchies and Semantic Databases. WordNet is a semantic network represented through an acyclic graph [3]. The relations of hierarchy in WordNet are linked with the help of pointers that convey a semantic network. Traversing such an hierarchy, provides direction of constructing a list of facets. Hence, for generation of facets, we shall be using WordNet.

To understand the challenges, we need to understand the technologies that we are working with and their structure within. In this thesis we shall be working with human annotated lexical database WordNet and three state-of-the-art word-embeddings models- Word2vec, fastText and Glove. We shall then further, try to create a word-embedding model that can accommodate positional co-occurrence in its training to create a word-embedding model apt for facet generation tasks and hence, we shall also mention challenges associated with developing such an architecture.

2.1 WordNet and Associated Challenges

To be able to construct such lists of facets from WordNet, we need to understand the underlying structure of the WordNet. WordNet groups words into synsets, where each synset represents a unique concept. Each synset contains a word or group of words that are instances of the synsets containing them. The word or group of words are also called lemma or lemmas (plural). The categorization among synsets is also done along the lines of part-of-speech (POS) structures and the number of synsets belonging to each POS structures is summarized in table 2.1. WordNet is further arranged in hierarchical structures whereby a semantic relation ensues between connected hierarchical structures. Since, there are four POS structures, we shall briefly mention each of them and understand which ones we shall be working with.

Table 2.1 : Table demonstrating number of synsets for every part-of-speech in WordNet.

Parts of Speech	Number of Synsets
Noun	82115
Verbs	13767
Adverbs	3621
Adjectives	18156

Noun synsets make up the major chunk of the WordNet. There are 82115 noun synsets in WordNet and are arranged in hierarchical structures. We shall be traversing these synsets and forming a list of lemmas that are instances of a particular synset. We shall also leverage the hierarchical structures of the WordNet for the creation of the datasets. Detailed explanation is provided in chapter 3.

Jing, H. [14] has shown that irrelevant senses could be pruned using verb argument clusters, when such clusters are developed with respect to a domain. On the other hand, when we are dealing with word-embeddings in general, we know that these embeddings are trained on data belonging to more than one domain. So, there exists huge ambiguity that can not be explained for methodologies that we use for noun synsets. Also for the

relatedness of two words, when WordNet relatives are compared, the results are noticeably inferior for verbs than for nouns [15]. Topological anomalies like cycle, rings, dual-inheritance in representation of verbs in the WordNet result in problems like verbs having no hypernym or troponyms [15], further adding to the problems in generating lists of facets. Hence, we shall not be considering the verb POS in WordNet for generation of term lists

The relations of hypernym-hyponym does not exist for representation of adverbs and adjectives [16] in WordNet, and hence use of the hierarchical representation for generating a collection of facets is not possible for these parts-of-speech structures. To remove any bias in deriving conclusion, in comparison to the POS leveraging their hierarchical structures, we shall not be considering adverbs and adjectives for generating lists of similar facets/lemmas.

To sum up, we shall only be considering the "Noun Synsets" of the WordNet [4] for generating the term lists that contain a group of list of facets/ similar lemmas.

2.2 Word Embeddings and Associated Challenges

In this thesis we shall be working with three state of the art word-embedding models - Word2Vec, fastText and Glove. Each of these word-embedding models are developed with different architectures and enough information about them can be found on the internet. Some brief information on these models are mentioned in the prerequisites section of the thesis. These models are offered directly in the "Gensim" library of python. More information on the model and their licenses can be found here[10].

Even though these state-of-the-art word-embeddings models are trained on huge amounts of data, there are words that these models have not seen while training. Hence, there are words that lack representation in the word-embeddings model. There are many words in the WordNet, whose representations are not in these word-embeddings models. Our evaluation procedures can not give results, if these missing representations are not removed in the evaluation procedure. Hence, while we shall be making comparisons, we

[10] "RaRe-Technologies/gensim-data - GitHub." https://github.com/RaRe-Technologies/gensim-data. Accessed 20 Oct. 2020.

shall not take into account those lemmas whose representations are not in the word-embeddings model. For example, “heterotroph”, the representation of this word is not present in the Word2Vec model and hence, we shall not evaluate whether as *support* or *query* when this word appears for evaluation procedure for the Word2vec model.

2.3 Other Related Tasks

Feddoul, L. et al. [13] points out that manual facet predefinition is often inappropriate and apt choosing among facets is virtually impossible without algorithmic support. Hence, researchers have proposed various kinds of algorithmic support for the tasks of facet generation. Significant contributions come from the domain of faceted browsing over knowledge graphs [23] [5]. Prominent approaches use statistically predefined facets for data navigation and do not consider continuously changing data sources e.g. Ontogator [24] or mspace [25]. Other projects and various aspects of facet generation are discussed in [26, 27, 28, 29]. Most recent and promising works involve generating facets dynamically on the fly. This type of facet generation technique relies on building dynamic SPARQL queries and executing them on respective SPARQL endpoints [27].

According to Saedi-et-al [18], semantic networks and semantic spaces have been two prominent approaches to represent lexical semantics. While a unified account of lexical meaning relies on being able to convert to another, the conversions from semantic networks into semantic spaces has started to attract more attention recently. They have presented methodologies as well, for such conversions and their conversions from WordNet to the resulting embeddings has performed better and in fact substantially superior to performance of word embeddings based on very large collections of texts like Word2vec on mainstream semantic similarity tasks.

Various deep learning algorithms have been proposed in developing such graph embeddings techniques notably using node embeddings [12]. Here the goal is to encode graph nodes as low-dimensional vectors that faithfully summarize their graph position and topology of their local neighbourhood[11]. Recent works proposed for node embeddings are - node2vec [19], deepwalk [20]. Some already well established approaches

[11] "ShiroCupz/Embedding-WordNet - GitHub." https://github.com/ShiroCupz/Embedding-WordNet. Accessed 26 Oct. 2020.

for graph embedding techniques include matrix factorization based approaches include Local Linear Embedding (LLE) [21] and laplacian Eigenmaps [22]. However, these transformations have not been analyzed for facet generation tasks, but they present a huge potential for the purpose of facets generation tasks.

2.4 Challenges in Developing Word-Embeddings Model for Facet-Generation Tasks

Feddoul, L. et al. [13] has already pointed out that manual facet predefinition is often inappropriate and apt choosing among facets is virtually impossible without algorithmic support. Hence, we should understand that the outlook of this thesis is to analyze the algorithmic support of word-embeddings for the facet-generation tasks.

Although, word embeddings give us a way to use an efficient, dense representation in which similar words have a similar encoding, word-embeddings themselves are trained on the principle that the words that co-occur, share greater proximity than words that do not co-occur. This implies that both associated and similar words have a similar encoding. And it is because of this, that the tasks of facet generation by word-embeddings models is hindered due to blurring of boundaries between related and associated words. For example, are cup and coffee related or are they just associated [1]? This reasoning prompts us to draw a stricter boundary that facilitates the purport of word-embeddings for facet generation tasks.

On a closer look, we observe that different state-of-the-art word-embeddings models, while training do not consider the relevance of the position with respect to the co-occurrence and we believe if the position of the co-occurrence while training is considered, similar words would have higher proximity than associated words. This in turn could facilitate the use of word-embeddings models for the facets generation tasks.

Consider the following sentence:

"I eat *apples* after working out."

The relative position of eat with respect to *apples* is one before it and anything that can be substituted for *apples*, probably also belong to the eatable category. We believe if we can accommodate this *positional* co-occurrence while training a word embeddings model, similar words will have more similar encoding than associated words and this word-embeddings model would be more suited to facets generation tasks.

If two words X and Y which while training have the same positional context, then they are more similar to each other and belong to a similar category of facets. For example, if X or Y ($X = Y = W_i$) generate the same positional context $\{W_{i-2}, W_{i-1}, W_{i+1}, W_{i+2}\}$ during training, then X and Y should have similar encoding in the word vector representation. This models our notion in the example presented above- anything that can be substituted for apples(X) also belong to the same category e.g. oranges(Y).

Using this logic as the central concern, we shall train a word-embeddings model by modifying the architecture of the skip-gram Word2Vec model such that it accommodates positional information while training word-embeddings. Besides traditional challenges like preprocessing, negative sampling etc., mentioned in training skip-gram architecture [17] for word-embeddings models, we need to accommodate positional information. This presents a more challenging theme and we shall be dealing with this in chapter 5.

This thesis presents one of the initial phases to consider word-embeddings models for the facet generation tasks and hence, we shall not be considering advanced aspects like facet ranking, entity type pivoting or indirect facet generation that are typical in facet generation tasks. Our main consideration would be on the analysis of distribution of facets in the word-embeddings models and how this distribution could facilitate the task of facet generation through word-embeddings models. Hence, based on this aspect we shall approach the development of word-embeddings model for facet generation tasks.

3. Dataset Preparation from WordNet

First, we shall evaluate the efficiency of the three existing state-of-the-art word embeddings models - (Google's) Word2Vec, fastText and Glove for the tasks of facet generation. To do so, we shall feed word/s that belong to the same concept as *support* to the embeddings and calculate their similarity for two queries - one query would be a word that belongs to the same category as *support* and another query would be a random word. We contest the word-embeddings algorithm to effectively recognise words that belong to the same concept over the random-word.

To generate words that belong to the same category, we use WordNet and its hierarchical structure. Through this way, we seek to group similar facets in one list, which shall further be used to generate *support* and *query* pairs that would be contested for similarity against a random word.

3.1 Logic in Dataset Preparation

For evaluation of efficiency of word embeddings models to recognize facets, we need three datasets. These three datasets are as follows:

1. X - a word or group of words that belong to a particular synset/concept.
2. Y - a word that belongs to the same synset/concept as X.
3. Z - a random word

X is our *support* and [Y, Z] are *queries* to the word-embeddings model. We feed [X, Y] to the word-embeddings model and calculate its cosine similarity. We do the same with [X, Z]. Then we compare both the similarity and the greater of the two is considered as facet

recognized by word embeddings model to be more similar to the *support* X. This is then further considered for efficiency. This is explained in chapter 4.

Now that we have established the underlying logic, we shall be leveraging the structures of synset within wordnet itself to extract the words that belong to the same synset/concept.

3.2 Dataset Preparation Process

For the preparation of the dataset, we shall first consider the ways in which such a dataset can be obtained. Since, there is a hierarchy with-in the structure of the wordnet itself, dataset preparation could be done in three ways leveraging the structure of the wordnet. For the reasons mentioned in the section 2.1, we shall only be working with synset that are nouns. In the WordNet database, we have 82,115 synsets that are nouns and hence the number of lists of words that are generated for each of the logic presented below are 82115 one corresponding to every synset. The structure of the wordnet describing the relationship between the synset is presented below in figure 3.1.

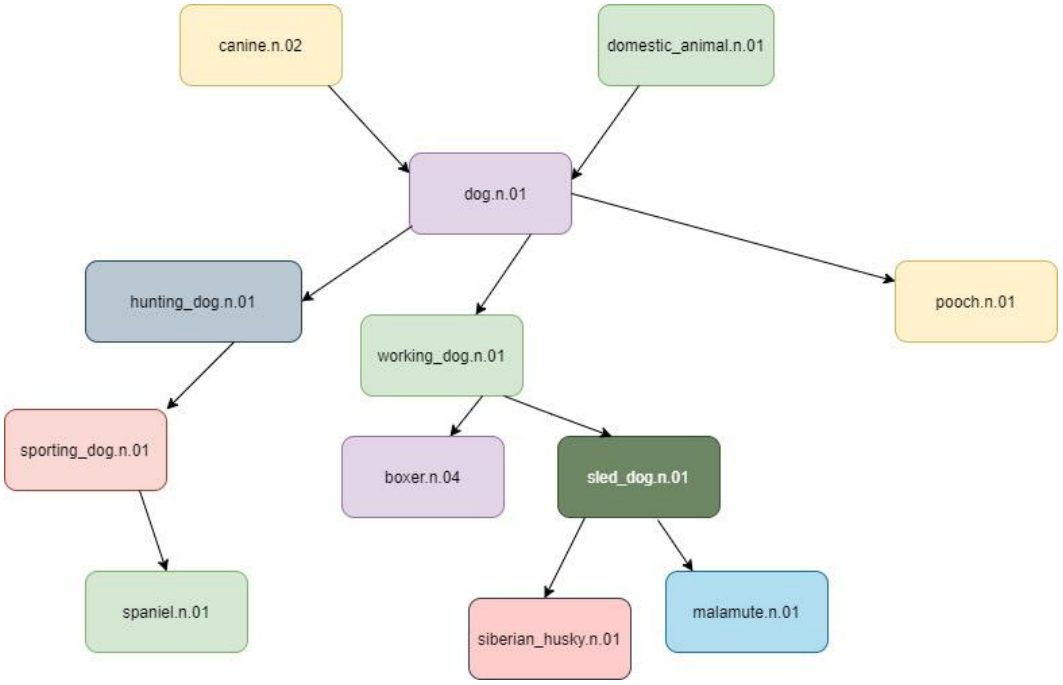

Figure 3.1: Explaining the Hierarchical representation of synsets with-in wordnet.

In the above figure, corresponding to dog.n.01, 'domestic_animal.n.01' and 'canine.n.02' are hypernyms and 'hunting_dog.n.01', 'working_dog.n.01', 'pooch.n.01' are hyponyms of depth 1. If we follow more arrows down or up the structure, we can harness the structure of wordnet for relations involving more depth. But in this thesis, we shall restrict ourselves to depth 1 for both hypernym and hyponym structures. In the figure above, the synset dog has other hyponyms as well than shown. So the hyponym data-set extracted would be even greater than what is shown in the diagram. The diagram is only a representation of the structure of synset with-in wordnet.

First we mention the three ways that could be used in dataset preparation, where each dataset prepared is unique depending on the logic used to derive them. They are:

1. Dataset Preparation from Single Synset.
2. Dataset Preparation by Hypernymy Traversal of the Wordnet Hierarchy Tree.
3. Dataset Preparation by Hyponymy Traversal of the Wordnet Hierarchy Tree.

However, we shall prepare the dataset for only the first two logics. Hyponymy traversal to extract the dataset from wordnet is not considered. Reason for this is explained in section 3.4.3. We shall extract a list of words based on each of the first two logic presented above. From these lists of words prepared per logic, we shall later extract [X, Y] as [*support*, *query*] pair, where these pairs generated share a similar concept among themselves based upon their representation in WordNet.

3.3 Term Lists Preparation Directly from Synset

For this process, we shall extract all the lemma names for a given synset. This gives us two types of entities. On one side, we have synset that have only one lemma name associated with the synset, let us call it *single-lemma-synset*. On the other hand, we have synsets that have more than one lemma name associated with the synset, let us call it *multiple-lemma-synset*.

Now we intend to make comparisons for judging similarity among lemmas from a particular synset. For this purpose, single-lemma-synset can not be used because they have only one lemma in it. However, further in the process for making comparisons to

establish the extent of similarity among *multi-lemma-synset*, we need some random words against whom the similarity would be tested. These groups of random words can be collected at this step itself. This is described in section 3.6.

Figure 3.2 explains how the list is extracted for a particular synset. All the lemmas belonging to the dotted connections are not selected and lemma that belong to the synset itself are only considered. Hence, for synset 'dog.n.01' lemmas - 'dog', 'domestic_dog' and 'Canis_familaris' are used extracted to the list corresponding to this synset.

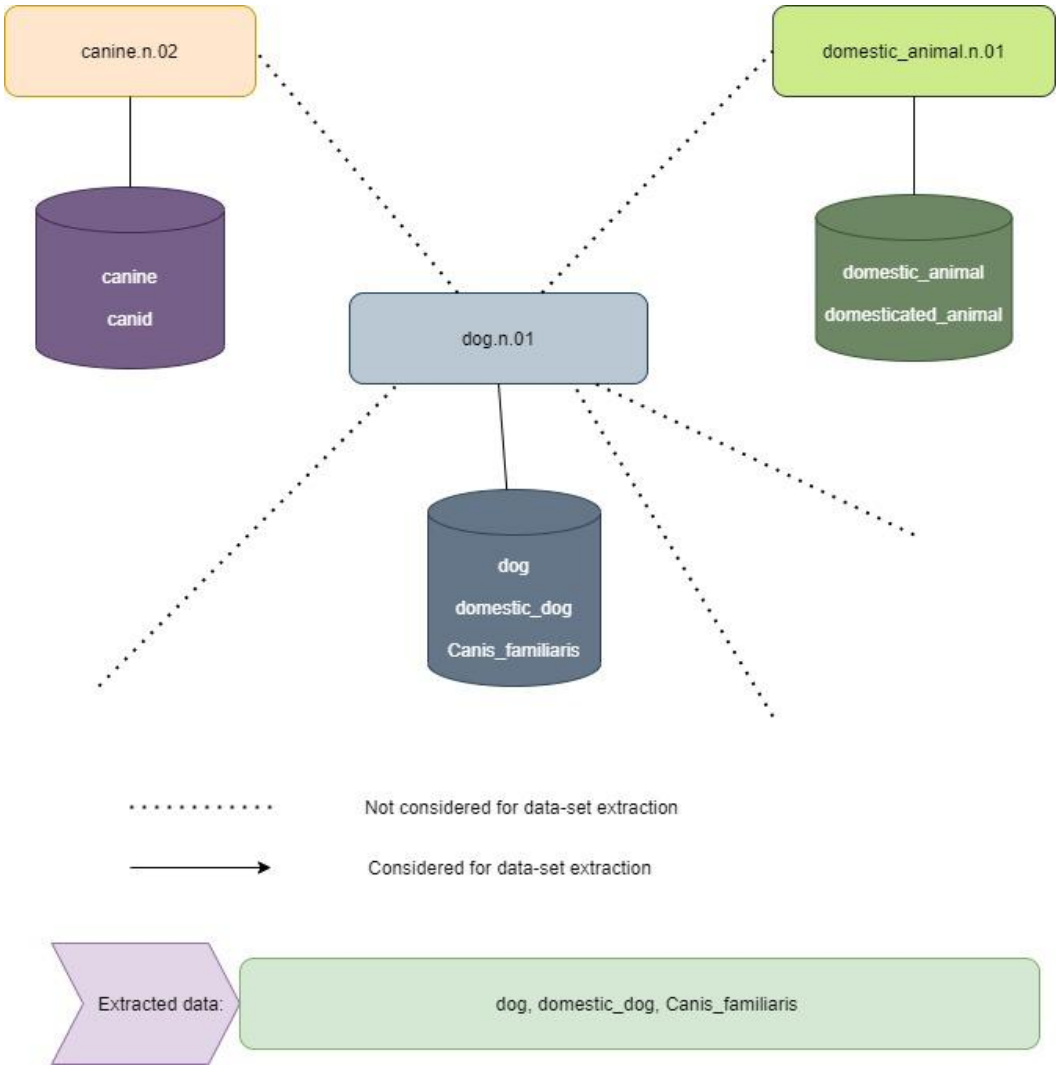

Figure 3.2: List of words generated from the synset itself. Boxes contain the name of the synset and associated cylinders represent the lemma names that belong to that synset.

3.4 Term Lists Preparation by Leveraging one depth Hypernymy- Hyponymy structural relations in WordNet.

Before we proceed to prepare the dataset leveraging the hierarchical structure of the WordNet, we should ask why are we doing so? In any facet generation tasks, few facets belong to the exact same category as *support*, but few other facets share some form of relation with the *support*, although they do not belong to the exact same category as the *support* itself. This aspect of the relation is governed by the extent of specificity. This is presented in the next sub-section.

3.4.1 Specificity for facet Generation Schema

While we shall extract pairs of words leveraging the Hyponym-Hypernym structure of WordNet, it is important to understand what are the concerns of specificity in facet generation tasks and how we shall address them. Facets are suggested based on their similarity with the 'object', which is provided. The higher the ranking in similarity, the higher probability is, with which the facet would be suggested. And similarly in this way, less similar entities are ranked lower and are suggested later. This similarity is highly correlated to the concept of specificity in WordNet. The farther the distance between two synsets in wordnet, the less specific is the relation among them as compared to the synsets in the vicinity. And this is proportionately reflected on the postulation of similarity.

We present an example to demonstrate this. Consider figure 3.1. We shall take into consideration following synsets namely - 'dog.n.01' and 'pooch.n.01'. The lemmas that belong to the synset 'pooch_n.01' are 'barker', 'pooch' and 'bow-wow'. These lemmas are very similar to each other as they belong to the same category- 'pooch.n.01'. This higher similarity arises from the specificity of the synset itself. If we use a lemma from synset 'dog.n.01', this lemma is a bit general for lemmas that belong to the 'pooch.n.01'. Nevertheless, they share some good similarity. So, in this manner, leveraging the hierarchical structure of wordnet, we can create lists of lemma that induce some generality in it. We should however note here that, this sense of similarity and generality is easy to apprehend with-in smaller levels of depth (here depth is 1). But if we traverse higher depths in WordNet seeking such generality, this sense of similarity will start to

appear vague because the specificity of synsets which are at the end and have almost no hyponym cannot not be held with much accountability with those that are higher up and have many hyponyms. For this reason, in this thesis, we are leveraging the hierarchical structure of WordNet only for depth of level 1.

So, in the end this method of data generation harnesses the structure of WordNet itself for the purpose of facet generation. Using this schema, on one hand we shall measure the efficiency of word embedding models to recognize specific facets and on the other hand this efficiency would be calculated for recognizing some generality among facets. These two ideas are modelled in preparation of a dataset by considering the inclusion of hierarchy of WordNet or not. We are leveraging the hypernym-hyponym structure of WordNet to create a dataset that seeks to introduce generality among the data without compromising much of the specificity. This way the extent of specificity is leveraged using the structure of WordNet itself. And maintaining this as a central concern, we shall proceed with the generation of lists that encompasses the hierarchical structure of the WordNet.

Now that we have defined the specificity parameter to observe while traversing the tree, we need to understand how we are going to traverse the tree. As the tree is a hierarchical structure, we can traverse it in two ways:

1. Hypernymy Traversal Approach (Bottom Up Traversal)
2. Hyponymy Traversal Approach (Top Down Traversal)

3.4.2 Term Lists Preparation through Hypernymy Traversal of the WordNet Hierarchy Tree

In this process, we shall traverse the WordNet in a bottom to top manner for the preparation of the dataset. This way, we seek to leverage the hypernym relations in the wordnet for the extraction of the dataset. Because we want to use the notion of generality while also maintaining specificity in the preparation of the dataset, we shall restrict ourselves to depth one while traversing the WordNet for the preparation of the dataset. If higher depths are considered the notion of specificity becomes a bit vague and hence similarity is not much prominent for the purpose of facets generation. Of-course more depth such as depth of two to five could be considered, but a depth of more than five is not a good measure for such a dataset preparation considering the specificity. To present

findings from the perspective of some benchmark results, we shall only be considering dataset preparation by hypernymy traversal of depth one.

We shall consider figure 3.3 to understand the logic behind dataset preparation by hypernymy traversal in the WordNet. In the diagram, all the synsets present above the level of the given synset are hypernyms of it. A synset can have more than one hypernym or no hypernym at all. Table 3.1 describes the number of hypernyms different synsets have in wordnet for depth one. We can observe that the range of hypernyms a synset can have vary from zero to a maximum of five. An interesting thing to notice here is that more than 98% of synsets have a maximum of one hypernym.

Table 3.1: Number of Hyper Synset for a given synset up for depth of one level.

Number of Hypernyms	Number of Synsets
0	7726
1	72967
2	1388
3	30
4	3
5	1

From the figure 3.3, we can see that the synset 'dog.n.01' has two hypernyms namely- 'canine.n.02' and 'domestic_animal.n.01'. The lemmas from the hypernyms are- 'canine', 'canid', 'domestic_animal' and 'domesticated animal', whereas the lemmas from synset itself are- 'dog', 'domestic_dog' and 'canis_familaris'. So, the dataset prepared for this synset using the hypernymy traversal of depth one contains both the lemmas- lemmas from hypernyms and lemmas of the synset itself. Hence the final list that is extracted is - 'dog', 'domestic_dog, 'canis_familaris', 'canine', 'canid', 'domestic_animal' and 'domesticated animal'.

Hence, in this manner we shall create a list for each of the 82115 synsets from their lemmas and the lemmas of all their hypernyms. It has also been made sure that our data

does not have duplicates, because duplication of data would account for higher accuracy, which would make bad inference.

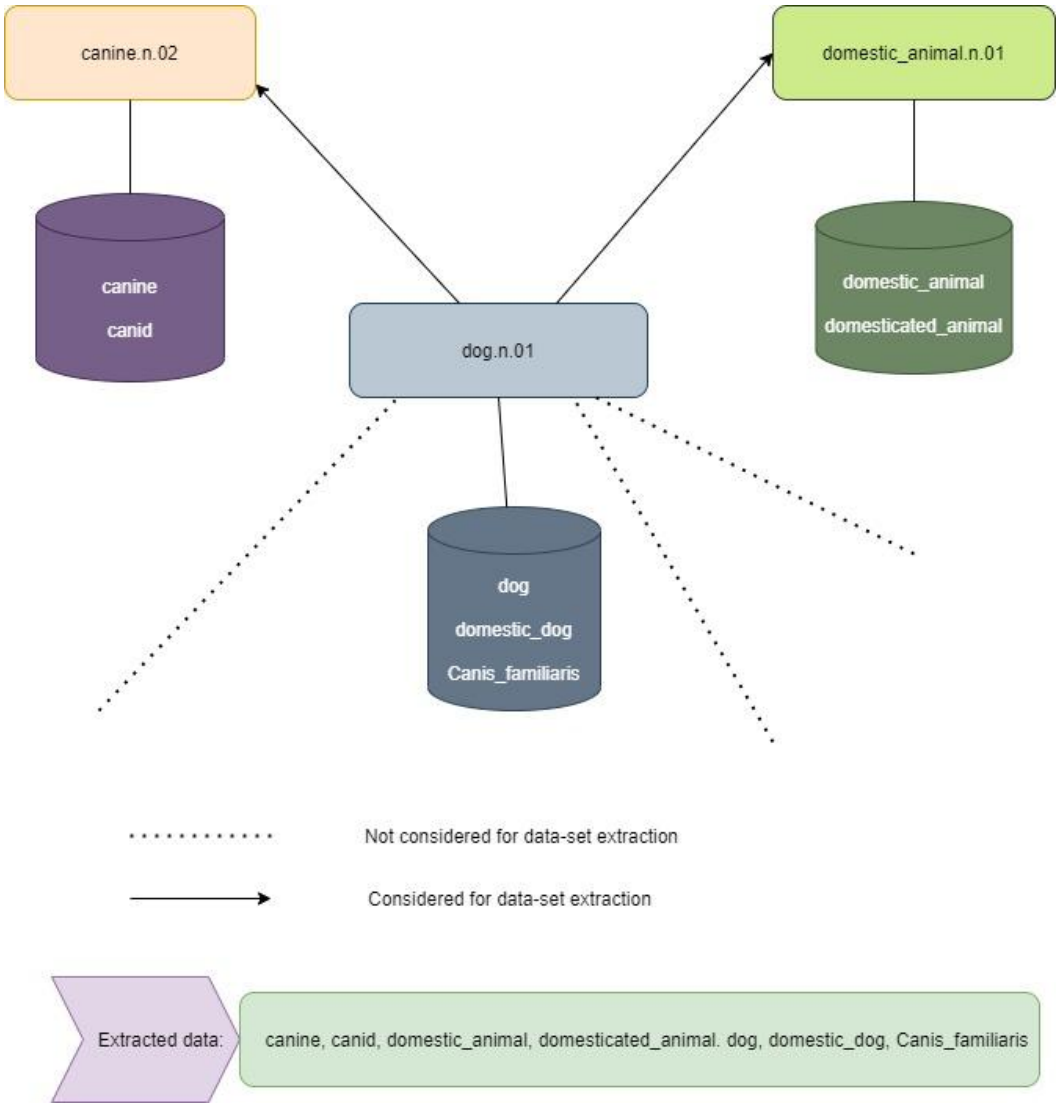

Figure 3.3: Demonstrating data-set extraction leveraging the hypernymy relation in the wordnet. Boxes contain the name of the synset and associated cylinders represent the lemma names that belong to that synset.

3.4.3 Hyponymy Traversal of the WordNet Hierarchy Tree

There exists another way of traversal in a hierarchical structure - the top down approach. With respect to the WordNet, this top down approach - we call it "Hyponymy traversal". We shall not be considering hyponymy traversal for preparation of the dataset. This is

because, while doing the hyponymy walk on the WordNet graph, we risk grouping unrelated things together. This especially happens at the very top of WordNet, where there are many children from a parent.

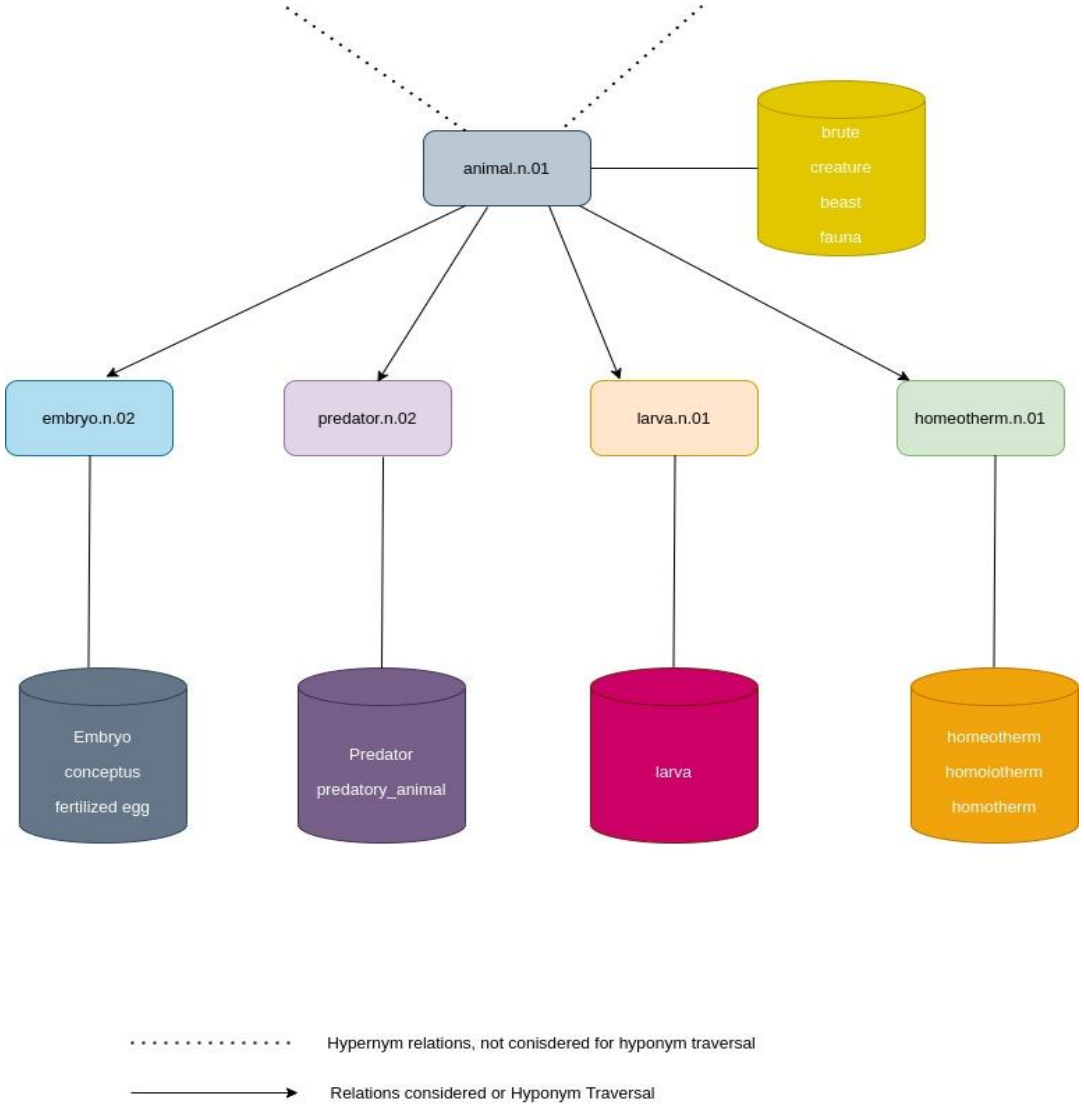

Figure 3.4: Demonstrating hyponymy relation in the WordNet. Boxes contain the name of the synset and associated cylinders represent the lemma names that belong to that synset.

This becomes very clear from figure 3.4. The synset 'animal.n.01' has 47 direct children but only 4 have been shown in the figure for illustrative purposes. As we can observe, if we group lemmas from four children together for the given synset into a list, the extent of similarity is very vague to define even for human annotators. Hence, we shall not be considering the hyponyms traversal for dataset preparation.

3.5 Support and Query Generation

Now that we have prepared a list that contains lemmas belonging from a single concept, we shall extract *support* and *query* from these term lists [2]. The query extracted from this list per *support*, would represent a true query and one that word embeddings model should correctly identify over the random word. This *support-query* generation from the lists extracted above, will be done in two ways and they are mentioned in the following subsections.

3.5.1 Support-query as combination of a pair of words

In this technique of support-query generation, we shall have one word as *support* and other word as *query*. Both of these words are extracted from the list that we have generated using two logics discussed in 3.3 and 3.4.2.

For each of the 82115 lists of words corresponding to 82115 synsets generated as per underlying different logic presented above, we shall generate pairs of words from each list. Since, each list has a different number of words in it, the number of pairs generated per list differs. However, we can calculate the total pairs generated by summing the combination of all the pairs generated per list. If a list has n lemmas, then the number of pairs generated per list can be calculated using the equation 3.1. This is demonstrated in the figure 3.5.

$$C(n,2) = \frac{factorial(n)}{factorial(n-2) * factorial(2)} \qquad [3.1]$$

This is done for all three types of dataset generated. Our intent for doing this is that later in evaluation one word from this pair would act as *support* and the other word would be *queried* for similarity analysis along with a random word. This is explained in more detail later in chapter 4. Table 3.2 summarizes the number of *support-query* pairs generated for the lists generated through logic discussed in section 3.3 and 3.4.2. A special procedure is done for term lists extracted from section 3.4.2. In the hypernymy traversal we have *support-true query* extracted from the term lists that were also present in *support-true query* for term lists extracted from synset directly. So, first we remove all such elements and the support-query pair for section 3.4.2 now only contains the hypernymy relations among the pair.

Table3.2: Number of Support-Query pairs generated for different WordNet Traversal

Type of WordNet Traversal	Number of (Support-Query)Pairs generated
Preparation from Direct Synset	107469
Preparation from Hypernymy traversal of WordNet	366712

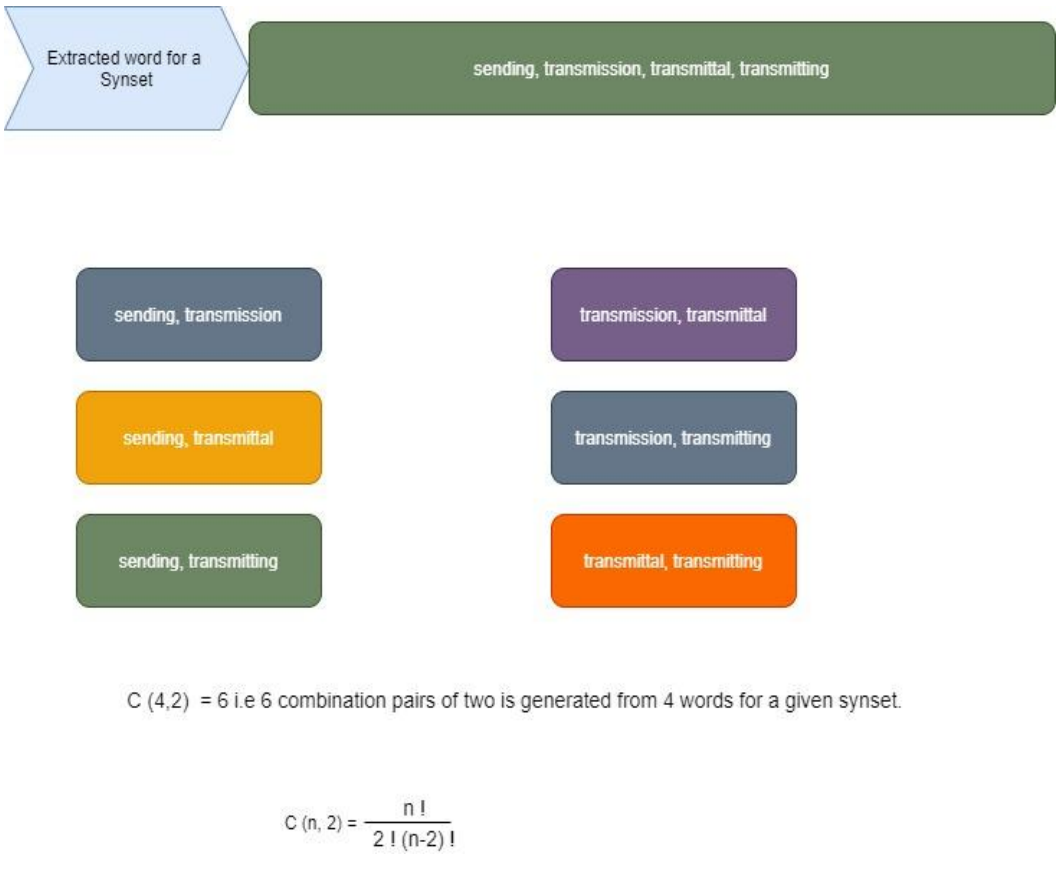

Figure 3.5: Combination pair generated for list of words. This method of generating a combination pair from a list is done for both types of the dataset.

3.5.2 Multiple Support and Single True Query Generation

The context being sought becomes clearer when there is more than one support explaining it. With this expectation that effectiveness of word embeddings models for the facet generation tasks improve when provided with more than one word, we shall generate a dataset that contains two words that will act as *support*. Along with this, we shall a query which is similar to the support. This will be done for both types of term lists [2] that are generated using the logics as discussed in section 3.3 and 3.4.2. This is better explained from figure 3.6.

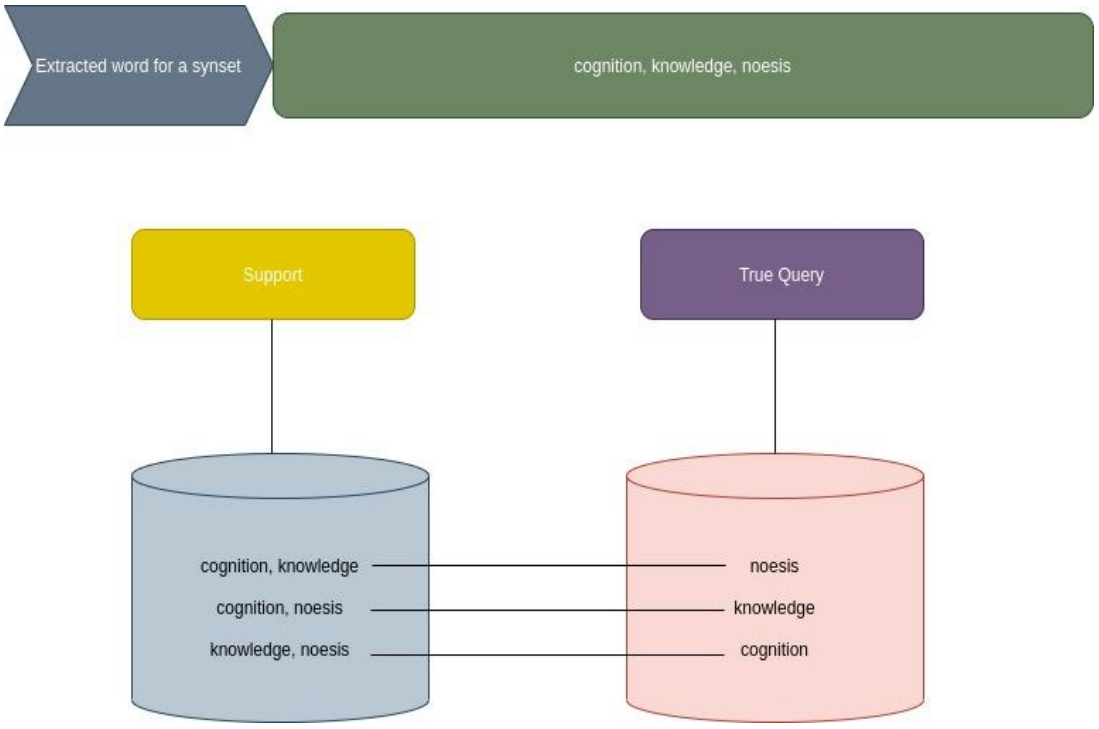

Figure 3.6 : Figure demonstrating multiple support and a single true query generation from list representing a concept.

For preparation of the dataset through this technique, for every pair of words in support, we have a word in true query, that belongs to the same concept/synset as words in support. Table 3.3 summarizes the number of multiple-support and single true query generated for different traversals of WordNet according to logic discussed in section 3.3 and 3.4.2. A special procedure is done for term lists extracted from section 3.4.2. In the hypernymy traversal we have *support-true query* extracted from the term lists that were also

present in *support-true query* for term lists extracted from synset directly. So, first we remove all such elements and the support-query pair for section 3.4.2 only contains the hypernymy relations among the pair.

Total number of *support-query* pairs generated from a list can be obtained using the equation 3.2. Here *n* is the number of items in the lists.

$$Q(n, 2) = (n - 2) * \frac{factorial(n)}{factorial(n - 2) * factorial(2)} \quad [3.2]$$

Table3.3: Number of Multiple Support-Single Query generated for different WordNet Traversal

Type of WordNet Traversal	Number of (Multiple Support- Single Query) generated
Preparation from Direct Synset	258408
Preparation from Hypernymy traversal of WordNet	1380524

3.6 Random word Generation

Now we need to have a random word that has to be given as a query along with one word from the generated pair against the support. For this, we take the *single-lemma-synset* that was presented in section 3.3. Out of these random words selected, we take only those words into further processes whose representations are present in the word embeddings model. This is done to ensure that our process of measuring similarity measures does not fail on account of its missing representation of the random word in word embeddings model.

For example, one of the *single-lemma-synset* has one lemma 'Heterotroph', whose representation is not present in the word-embeddings model. Because we do not want our algorithm to fail on account of absence of representation of such random words in our word-embeddings model, we need to remove it.

After filtering out such words and only taking into consideration such random words, whose representations are present in the word-embeddings model, we do not have enough words per pair of words for each logic under which these pair of words were generated. To overcome this problem, the number of random words are generated on the fly by duplicating the random words extracted above.

4. Effectiveness Evaluation

After we have prepared the dataset, we shall now evaluate the efficiency of word embeddings model to correctly identify the similarity among the words that belong to the same synset/concept. We are using three state-of-the-art word-embeddings models - (Google's) Word2Vec, (Facebook) FastText, (Stanford's) Glove for this purpose. These models are in open-source domain and are used subject to permissions of various licenses [12]. The effectiveness of word-embeddings model to correctly identify words that belong to the same concept/facet group is contested on the grounds of similarity against a random word. For similarity comparisons, we shall be using cosine similarity. This is predefined by an inbuilt function in Gensim called 'n_similarity' about which we mention further in the section 4.2 . The evaluation procedure for similarity comparison is described below.

4.1 Evaluation Procedure

For the purpose of evaluation, we have three entities namely -

1. X - a word or group of words that belong to the same facet group/concept.
2. Y - a word that belongs to the same facet group as X.
3. Z - a random word.

We shall make two similarity comparisons namely -

1. [X, Y] - similarity comparison between *Support* and *True Query*.
2. [X, Z] - similarity comparison between *Support* and *Random Word*.

---

[12] "RaRe-Technologies/gensim-data - GitHub." https://github.com/RaRe-Technologies/gensim-data. Accessed 7 Oct. 2020.

We then compare, the similarity value between [X, Y] and [X, Z] and if the similarity value between [X, Y] has a greater value than [X, Z], we mark this event as correct identification of facets by word-embeddings models and append this similarity value to "correct predictions list" and when this not the case, we mark the event as wrong identification of facets by word-embeddings models and append this similarity value to "incorrect predictions list". This evaluation procedure is well illustrated in figure 4.1.

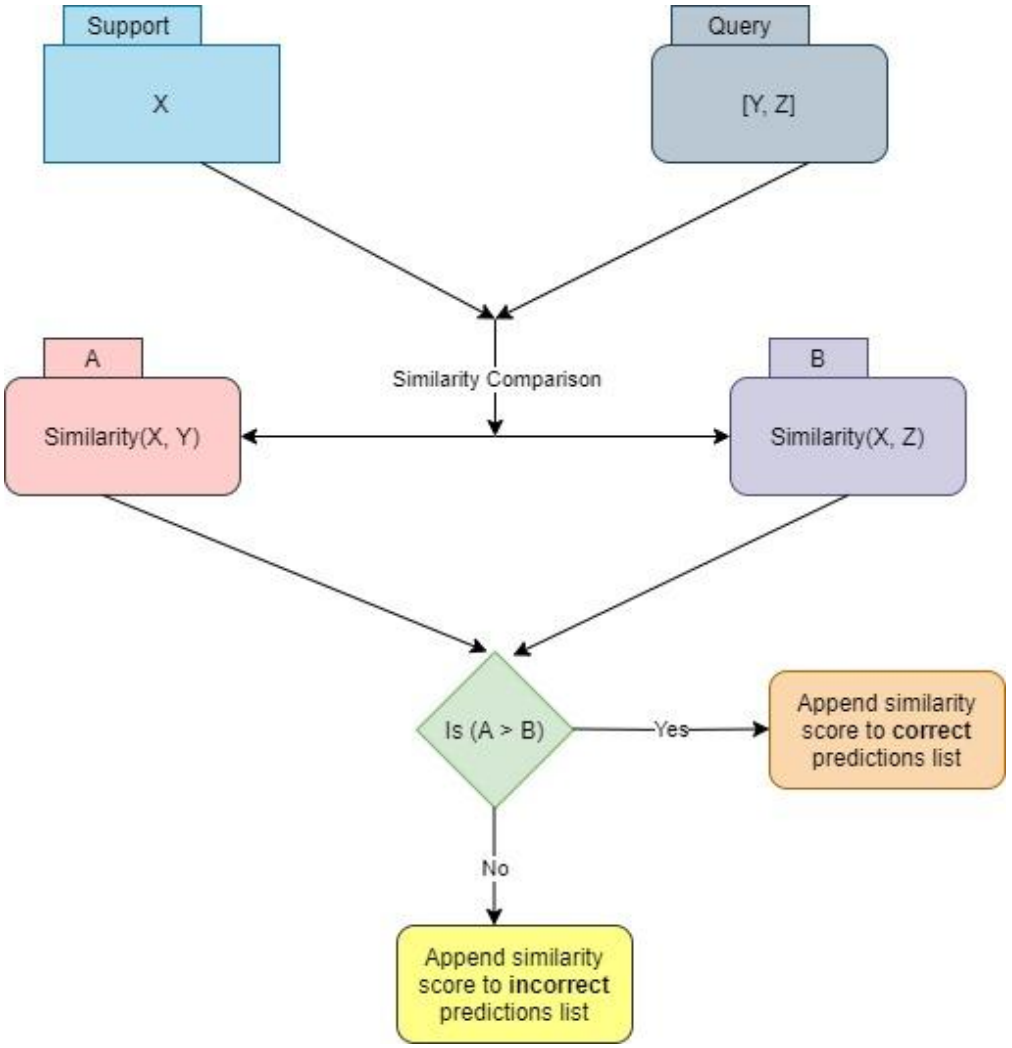

Figure 4.1: Demonstrating the evaluation procedure. X is *Support*, Y is *True Query* and Z is the random word. Word embedding models are queried for similarity comparisons between [X, Y] and [X, Z].

It is worthwhile to bring to notice that there are many lemmas in the WordNet that have no representation in the existing Word2Vec, fastText or Glove word embeddings model and according to section 3.6, we are only considering random words that are in the word embeddings model.

However, our procedure could fail on account if either lemmas from *support* or *true query* are not present in the word embeddings model. This happens because the word-embeddings algorithm has not seen these lemmas while training. Hence, when such a scenario arises, we remove such a pair of *support* and *true query* from consideration by our procedure and a separate count of them is maintained. These results are summarized in corresponding tables.

We shall perform the above procedure, using two methodologies namely -

1. Single Support and single True query.
2. Multiple Support and single True query.

These methodologies are run for the datasets derived using the logic 3.3 and 3.4.2 on the three state-of-the-art word-embeddings models - Word2Vec, fastText and Glove. Before, we mention our analysis on these two methodologies, we shall briefly understand the function “n_similarity” under the Gensim package of python for better understanding of the analysis.

4.2 Gensim’s *n_similarity* function

Gensim’s *n_similarity* functionality[13] facilitates all the comparisons being made. This functionality of this module provides enough flexibility for both our methodologies. It takes input as two sets of *docvecs*, then calculates the mean of each of the docvecs independently and then compares the cosine similarity between the two means. This can be better inferred from the figure 4.2.

[13] "gensim: models.keyedvectors – Store and ...." 1 Nov. 2019, https://radimrehurek.com/gensim/models/keyedvectors.html. Accessed 19 Oct. 2020.

For our purposes, let us reserve the block P for *support* and block Q for *query*. Both blocks can take any number of arguments and then only the mean is calculated. Since we are using two different methodologies differing in number of values provided in the *support*, block P can accommodate both types of methodologies - *single support* and *multiple support*. The *docvec* of block Q is similar in both the methodologies and takes only a single *query* as its argument. Hence, in this manner, we shall be conducting the effectiveness evaluation for different word-embeddings models.

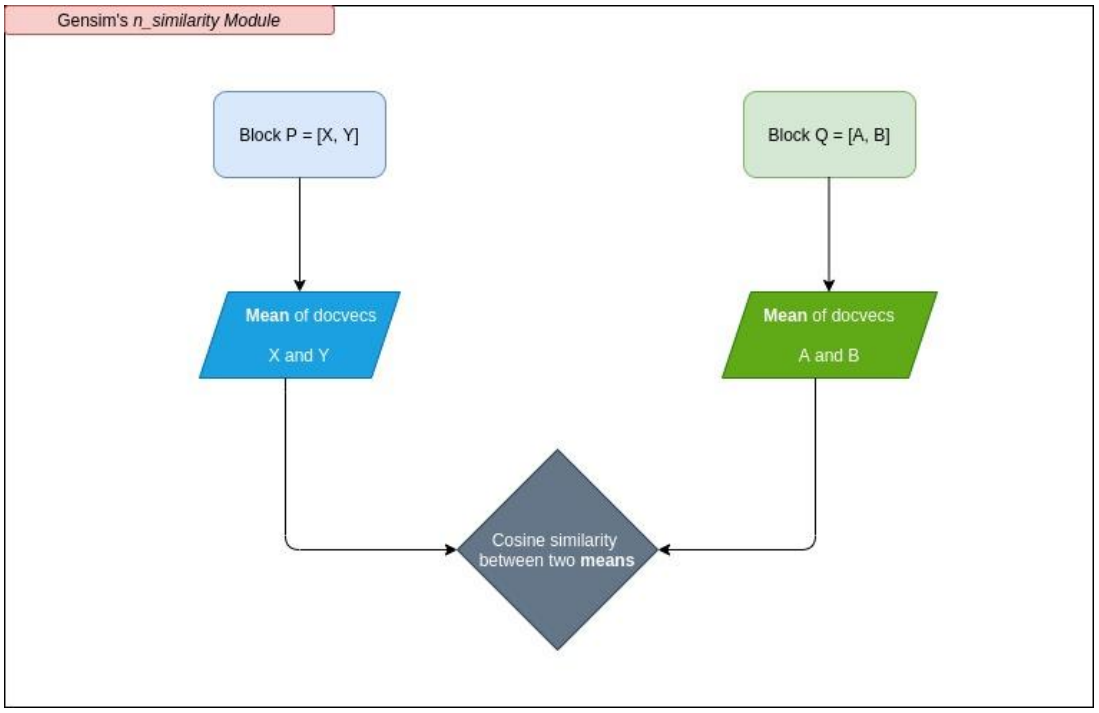

Figure 4.2: Demonstrating the inner workings of Gensim's *n_similarity* module.

4.3 Analysis on Single Support and Single True Query

In chapter 3, we created a dataset by extracting a single lemma as *support* and another single lemma as *query* from two types of term lists. These term lists are:

1. Term lists generated from the Synset directly.
2. Term lists generated through hypernym traversal of WordNet

Since we have a single lemma as support, Block P in figure 4.2 would get only one argument as we use a single *support* for this methodology. Block Q also gets only one argument.

4.3.1 Term Lists Generated from Synset directly

In section 3.3 we created term lists directly from a synset. From each term list, we extracted pairs of lemmas, where in each pair one lemma would act as *support* and the other lemma would be our *true query*. After this process, we run the evaluation procedure as described in section 4.1. We do this for three state-of-the-art word-embeddings models - Word2vec, fastText and Glove. We make comparisons among each of these models and results are summarized in table 4.1. We can see that the fastText model performed best out of all three models.

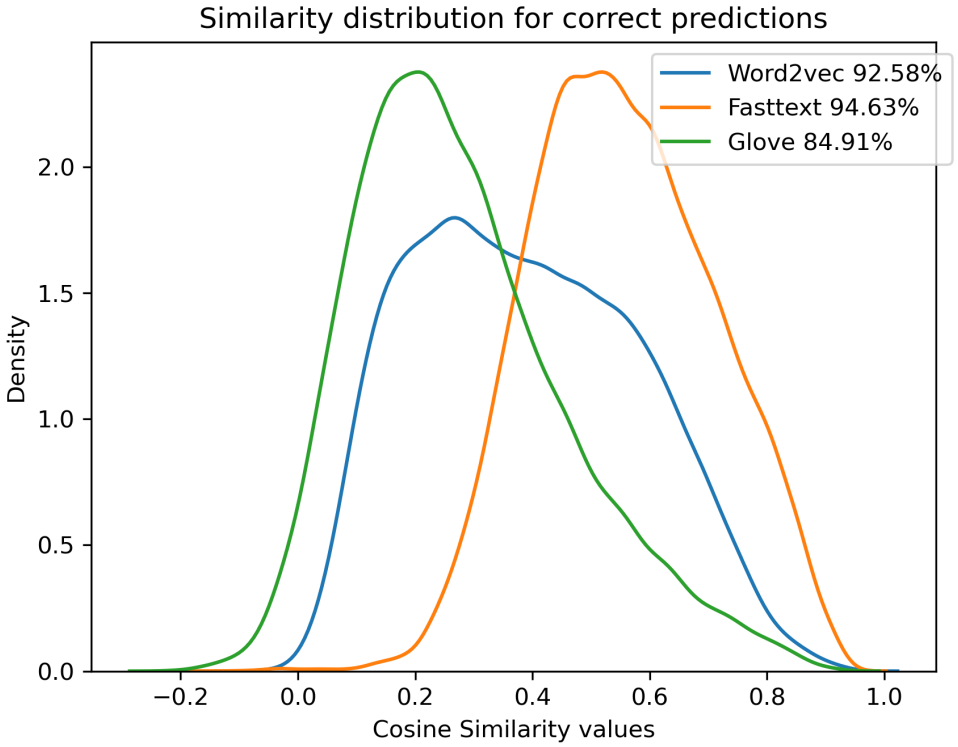

Figure 4.3: Figure demonstrating similarity distribution for correct predictions for term lists directly extracted from the synset. Here the word-embeddings models have correctly identified *support-true query* as being more similar than *support-random lemma*. Here, support is a single lemma.

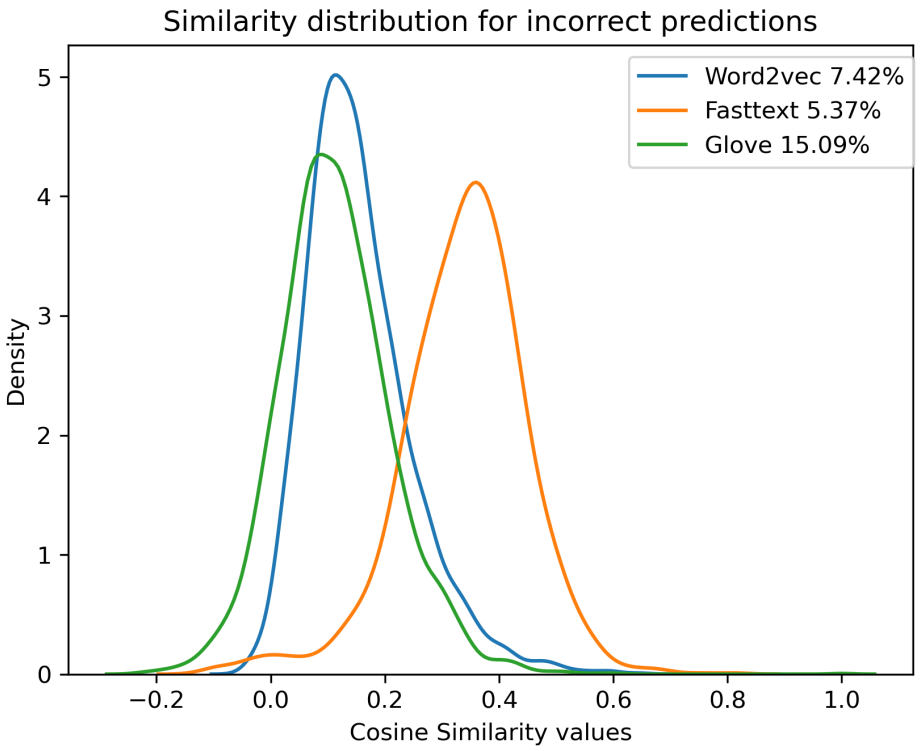

Figure 4.4: Figure demonstrating similarity distribution for incorrect predictions for term lists directly extracted from the synset. Here the word-embeddings models have incorrectly identified, *support-random lemma* as being more similar than *support- true query*. Here, support is a single lemma.

However, these results alone do not tell the complete picture and hence, we shall further analyze the similarity distribution for both correct and incorrect predictions. In figure 4.3 and 4.4 , we have created a density plot of similarity distribution for correct and incorrect predictions respectively.

4.3.2 Term Lists Generated through Hypernym Traversal of WordNet

In section 3.4.2 we created term lists by traversing the hierarchical structures leveraging the hypernym relations. From each term list, we extracted pairs of lemmas, where in each pair one lemma would act as *support* and the other lemma would be our *true query*. A special procedure is done for term lists extracted from section 3.4.2. In the hypernymy traversal we have *support-true query* extracted from the term lists that were also present in

support-true query for term lists extracted from synset directly. So, first we remove all such elements and the support-query pair for section 3.4.2 only contains the hypernymy relations among the pair. After this process, we run the evaluation procedure as described in section 4.1. We do this for three state-of-the-art word-embeddings models - Word2vec, fastText and Glove. We make comparisons among each of these models and results are summarized in table 4.1. We can see that the fastText model performed best out of all three models.

Since, we induced some generality among the lemmas using the hypernymy traversal, we expect the accuracy to be lesser than in section 4.3.1. Similar to above we shall analyze the similarity distribution for both correct and incorrect predictions. In figure 4.5 and 4.6, we have created a density plot of similarity distribution for correct and incorrect predictions respectively.

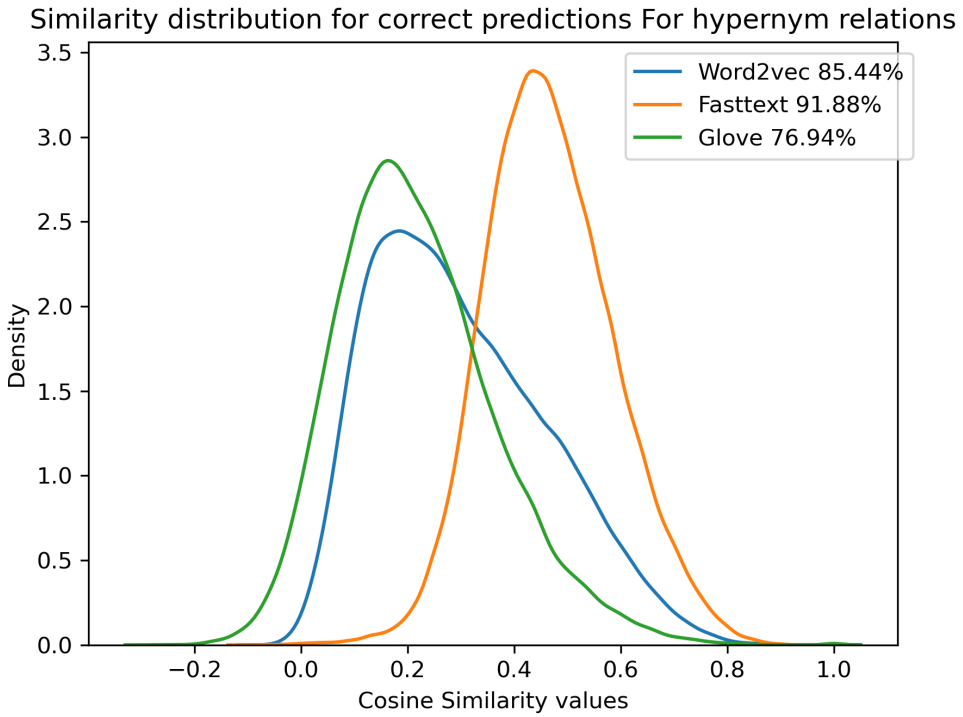

Figure 4.5: Figure demonstrating similarity distribution for correct predictions for term lists containing hypernym relations. Here the word-embeddings models have correctly identified *support-true query* as being more similar than *support-random lemma*. Here, *support* is a single lemma.

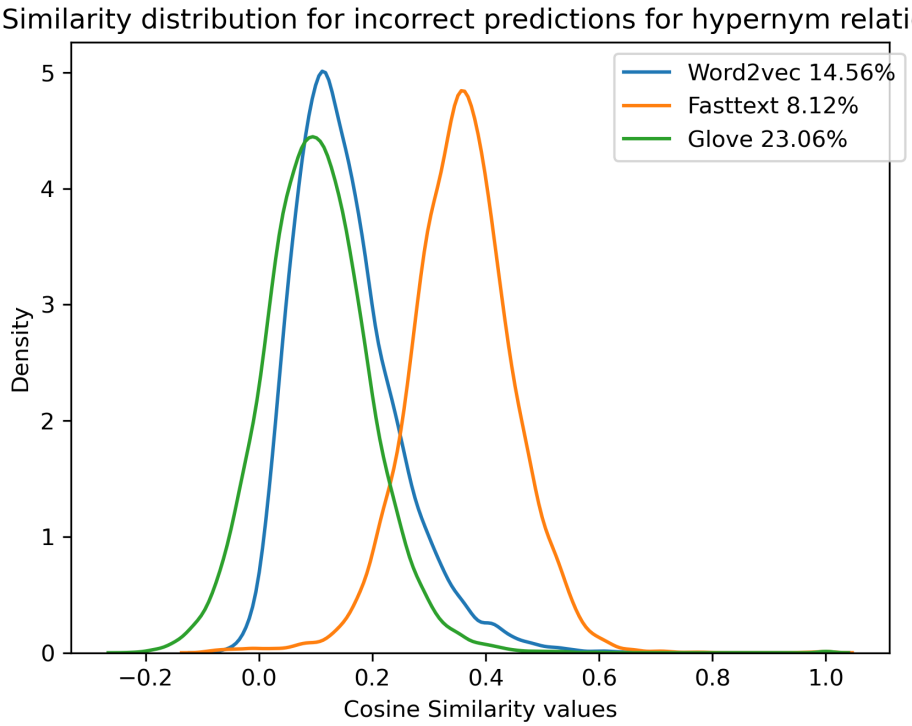

Figure 4.6: Figure demonstrating similarity distribution for incorrect predictions for term lists containing hypernym relations.. Here the word-embeddings models have incorrectly identified, *support-random lemma* as being more similar than *support-true query*. Here, *support* is a single lemma.

Table4.1: Table summarizing results by evaluation procedure for similarity comparisons on analysis of Single Support and single True Query.

Dataset prepared only from synset directly	Word2Vec	fastText	Glove
Correct Predictions: Support and True query with higher similarity and number of correct predictions	92.58 % 29763	94.62 % 28962	84.90 % 20583
Incorrect Predictions: Support and Random word with higher similarity and number of incorrect predictions	7.42 % 2384	5.38 % 1644	15.10 % 3658
Support or True Query missing representation in word embeddings Model: Not considered for analysis	75322	76863	83228

Dataset prepared from Hypernym Traversal on WordNet	Word2Vec	fastText	Glove
Correct Predictions: Support and True query with higher similarity and number of correct predictions	85.43 % 81818	91.87 % 86689	76.93 % 60590
Incorrect Predictions: Support and Random word with higher similarity and number of incorrect predictions	14.57 % 13944	8.13 % 7662	23.07 % 18164
Support or True Query missing representation in word embeddings Model: Not considered for analysis	165453	166864	182461

4.4 Analysis on Multiple Support and single True Query

We expect that when more queries are provided as support, the word-sense disambiguation increases and the word-embeddings models are more effectively able to recognize the facets that belong to the same concept. Why does this happen? What is the purport of this expectation? When multiple lemmas are provided as support, the word sense disambiguation increases and as meaning becomes more clear, the intended facet dimensionality can be inferred. Based on this reasoning, we can use the algorithmic support of word-embeddings models for facets generation tasks.

And using the logic above, we expect that the similarity curve for all the word-embeddings models, would shift towards right for correct predictions and towards left for incorrect predictions.

In chapter 3, we created a dataset by extracting multiple lemmas as *support* and another single lemma as *query* from two types of term lists. These term lists are:

1. Term lists generated from the Synset directly.
2. Term lists generated through hypernym traversal of WordNet

Since we have multiple lemmas as support, Block P in figure 4.2 would get two arguments as we use two lemmas in the multiple *support* for this methodology. Block Q gets only one argument.

4.4.1 Term Lists Generated from Synset directly

In section 3.3 we created term lists directly from a synset. From each term list, we used two lemmas to generate a *support* and one lemma as a true *query*. After this process, we run the evaluation procedure described in section 4.1. We do this for three state-of-the-art word-embeddings models - Word2vec, fastText and Glove. We make comparisons among each of these models and results are summarized in table 4.2. We can see that the fastText model performed best out of all three models.

We shall also further analyze the similarity distribution for both correct and incorrect predictions. In figure 4.7 and 4.8, we have created a density plot of similarity distribution for correct and incorrect predictions.

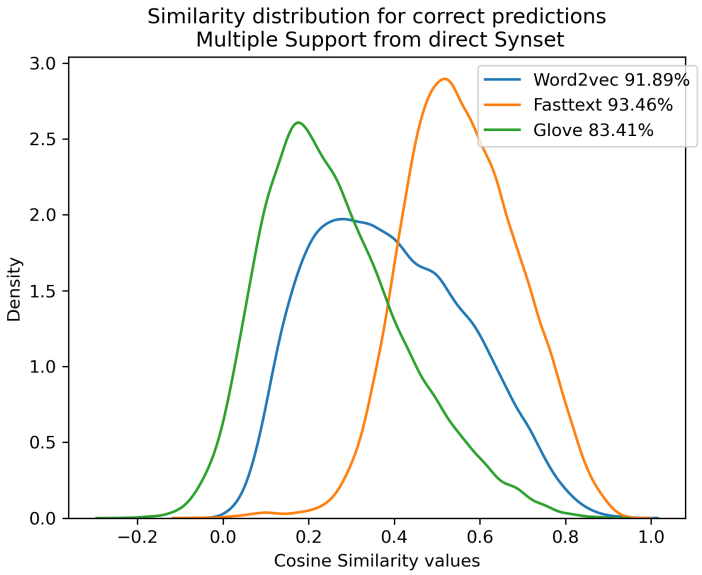

Figure 4.7: Figure demonstrating similarity distribution for correct predictions for term lists directly extracted from the synset. Here the word-embeddings models have correctly identified *support-true query* as being more similar than *support-random lemma*. Here, *support* has two lemmas.

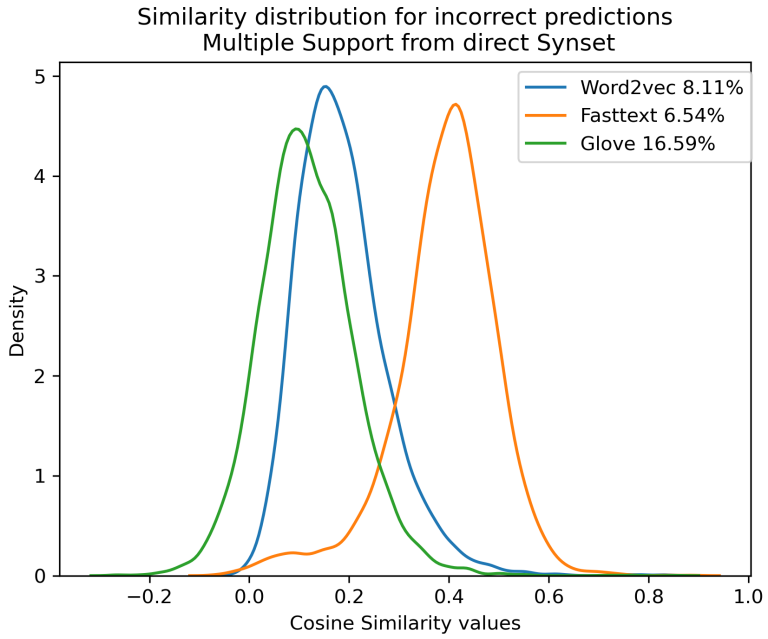

Figure 4.8: Figure demonstrating similarity distribution for incorrect predictions for term lists directly extracted from the synset. Here the word-embeddings models have incorrectly identified, *support-random lemma* as being more similar than *support- true query*. Here, *support* has two lemmas.

Table 4.2: Table summarizing results by evaluation procedure for similarity comparisons on analysis of Multiple support Support and single True Query.

Dataset prepared only from synset directly	Word2Vec	fastText	Glove
Correct Predictions: Support and True query with higher similarity and number of correct predictions	91.89 % 68242	93.46 % 73003	83.41 % 52741
Incorrect Predictions: Support and Random word with higher similarity and number of incorrect predictions	8.11 % 6026	6.54 % 5105	16.59 % 10493
Support or True Query missing representation in word embeddings Model: Not considered for analysis	184140	180300	195174

Dataset prepared from Hypernym Traversal on WordNet	Word2Vec	fastText	Glove
Correct Predictions: Support and True query with higher similarity and number of correct predictions	91.17 % 364398	94 % 388502	85 % 280704
Incorrect Predictions: Support and Random word with higher similarity and number of incorrect predictions	8.83 % 35271	6 % 24882	15 % 49388
Support or True Query missing representation in word embeddings Model: Not considered for analysis	980855	967140	1050432

4.4.2 Term Lists Generated through Hypernym Traversal of WordNet

In section 3.4.2 we created term lists by traversing the hierarchical structures leveraging the hypernym relations. From each term list, we extracted two lemmas as *support* and one lemma as *true query*. A special procedure is done for term lists extracted from section 3.4.2. In the hypernymy traversal we have [*multiple support, true query*] extracted from the term lists that were also present in [*multi support, true query*] for term lists extracted from synset directly. So, first we remove all such elements and the support-query pair for section 3.4.2 now only contains the hypernymy relations among the pair. After this process, we run the evaluation procedure as described in section 4.1. We do this for three state-of-the-art word-embeddings models - Word2vec, fastText and Glove. We make comparisons among each of these models and results are summarized in table 4.2. We can see that the fastText model performed best out of all three models.

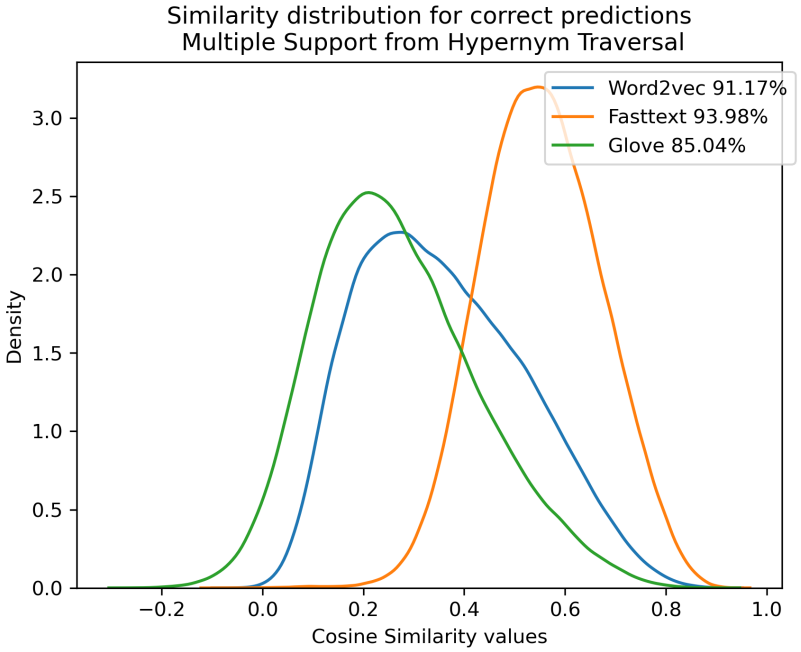

Figure 4.9: Figure demonstrating similarity distribution for correct predictions for term lists containing hypernym relations. Here the word-embeddings models have correctly identified *support-true query* as being more similar than *support-random lemma*. Here, *support* has two lemmas.

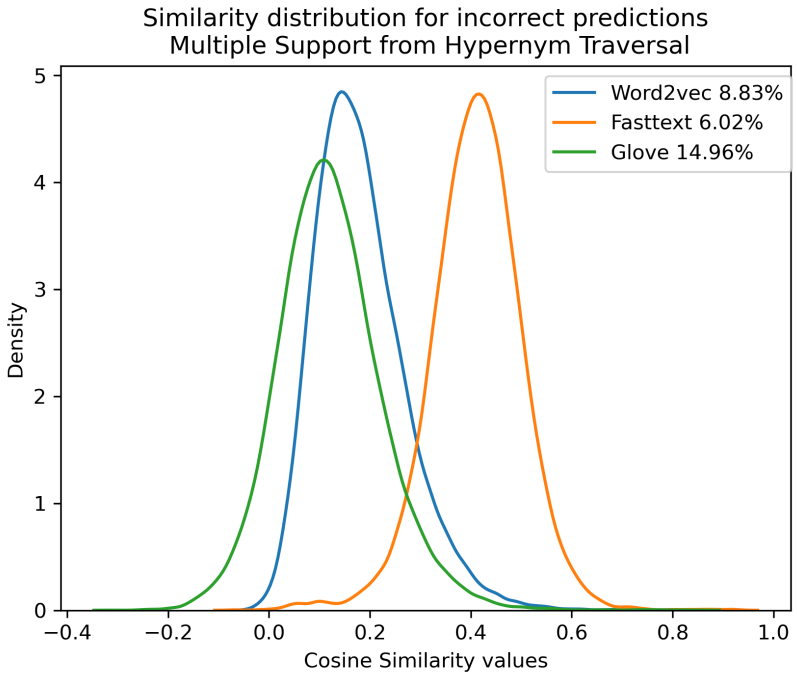

Figure 4.10: Figure demonstrating similarity distribution for incorrect predictions for term lists containing hypernym relations.. Here the word-embeddings models have incorrectly identified, *support-random lemma* as being more similar than *support-true query*. Here, *support* has two lemmas.

We have also created a density plot of similarity distribution for correct predictions and incorrect predictions for multi-support hypernym traversal term lists in figure 4.9 and 4.10 respectively.

4.5 Evaluation Analysis

We have summarized the results for both our methodologies -

1. Single Support and Single True Query
2. Multiple Support and Single True Query

in table 4.3 and 4.4 for correct predictions and incorrect predictions. They provide the relevant information for figures- 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9 and 4.10 and also allow

side-by-side comparisons. Now, we shall try to make sense of what these pictures and aforementioned table are conveying.

Table 4.3: Table summarizing mean and standard deviation for different evaluation procedures by different state-of-the-art word-embeddings models for correct predictions.

Dataset prepared only from synset directly	**Single Support Mean and Standard Deviation(SD)**	**Multiple Support Mean and Standard Deviation(SD)**
Word2Vec	Mean = 0.38 SD = 0.19	Mean = 0.39 SD = 0.18
fastText	Mean = 0.55 SD = 0.15	Mean = 0.55 SD = 0.13
Glove	Mean = 0.27 SD = 0.18	Mean = 0.26 SD = 0.16

Dataset prepared from Hypernym Traversal on WordNet	**Single Support Mean and Standard Deviation(SD)**	**Multiple Support Mean and Standard Deviation(SD)**
Word2Vec	Mean = 0.30 SD = 0.16	Mean = 0.36 SD = 0.16
fastText	Mean = 0.46 SD = 0.12	Mean = 0.55 SD = 0.12
Glove	Mean = 0.21 SD = 0.15	Mean = 0.27 SD = 0.15

Table 4.4: Table summarizing mean and standard deviation for different evaluation procedures by different state-of-the-art word-embeddings models for incorrect predictions.

Dataset prepared only from synset directly	**Single Support Mean and Standard Deviation(SD)**	**Multiple Support Mean and Standard Deviation(SD)**
Word2Vec	Mean = 0.15 SD = 0.09	Mean = 0.18 SD = 0.09
fastText	Mean = 0.33 SD = 0.11	Mean = 0.39 SD = 0.10
Glove	Mean = 0.11 SD = 0.099	Mean = 0.11 SD = 0.094

Dataset prepared from Hypernym Traversal on WordNet	**Single Support Mean and Standard Deviation(SD)**	**Multiple Support Mean and Standard Deviation(SD)**
Word2Vec	Mean = 0.15 SD = 0.09	Mean = 0.18 SD = 0.09
fastText	Mean = 0.36 SD = 0.09	Mean = 0.41 SD = 0.09
Glove	Mean = 0.10 SD = 0.094	Mean = 0.12 SD = 0.10

Since, models like fastText, Word2Vec have very high accuracy of more than 90% , we expect that these models could be used for facets generation tasks. But this is not the case. These word-embeddings models while training provide similar encoding to two types of entities - related and associated words. Hence, when asked for suggestions of facets, they

provide both kinds of words. This is inherent in word-embeddings models trained on large collections of texts. Hence, in these word-embeddings models there is a blurring of boundaries between related and associated words. For example, are cup and coffee related or are they just associated [1]? When provided *coffee* as support, *cup* is also suggested by the word-embeddings model and we can not accept such word-embeddings models for the tasks of facets generation. This reasoning prompts us to draw a stricter boundary among associated and related words that facilitates the purport of word-embeddings for facet generation tasks. And for this purpose, we shall further analyze the similarity distributions among the facets.

First observation that we get is that all the similarity scores when compiled and analyzed together through the pictorial representation, shows a gaussian distribution. Hence, much information can be achieved through mean and standard deviation of these structures.

As we can see from table 4.3, that for dataset prepared from hypernym traversal from WordNet has lesser mean than the dataset prepared from synset directly. This was expected as the similarity for lemmas obtained from synset directly are more similar in WordNet than for lemmas sharing hypernym relations. This is what has been reflected in similarity scores with higher mean for dataset prepared from synset directly. Also as we move from single support to multiple support in table 4.3, we observe that mean increases and hence, there is a form of sense disambiguation, as meaning becomes more clear. This is a very important observation, which prompts the ability of word-embeddings to extend its algorithmic support for facets generation tasks. On the other hand, there is a significant difference among the mean of correct predictions and incorrect predictions, which further accentuates the ability of word-embeddings models to differentiate unrelated words and their ability for facets generation tasks.

However, we also observe that from these Gaussian distributions, even for the best performing model - fastText, the mean for similarity score is 0.55. We know that this should be closer to 1 as they belong to the same category and if it were so, their algorithmic support could have been extended for the task of facets generation. We also expect the standard deviation to be smaller. However, when we query these word embeddings models for suggestions both associated and related words are suggested.

Since, the mean is not closer to 1, the state-of-the-art word-embeddings models cannot be used for facets generation tasks despite providing such a high accuracy. However, increase in the mean when moving from single-support to multiple support, we observe that there is a sense disambiguation and indeed, if improved word-embeddings models can be used for the tasks of facets generation.

On closer notice in the training of these word-embeddings models, we observe that relevance of the position is not considered. We believe that, if the relevance of the position could be considered for the tasks of facets generation, we can have word-embeddings models that can be used for the tasks of facets generation. This consideration would result in drawing a stricter boundary between related and associated words and this in-turn would produce such a word-embeddings model that can be used for the tasks of facets generation.

Consider following example to illustrate the reasoning presented above:

"I eat *apples* after working out."

The relative position of eat with respect to *apples* is one before it and anything that can be substituted for *apples*, probably also belong to the eatable category. We believe if we can accommodate this *positional* co-occurrence while training a word embeddings model, similar words will have more similar encoding than associated words and this word-embeddings model would be more suited to facets generation tasks.

If two words X and Y which while training have the same positional context, then they are more similar to each other and belong to a similar category of facets. For example, if X or Y ($X = Y = W_i$) generate the same positional context $\{W_{i-2}, W_{i-1}, W_{i+1}, W_{i+2}\}$ during training, then X and Y should have similar encoding in the word vector representation. This models our notion in the example presented above- anything that can be substituted for apples(X) also belong to the same category e.g. oranges(Y).

Using this logic as the central concern, we shall train a word-embeddings model by modifying the architecture of the skip-gram Word2Vec model such that it accommodates positional information while training word-embeddings and check whether such a model

would be suitable for facets-generation tasks. The entire procedure for accommodating positional co-occurrence while training word-embeddings models and the analysis is presented in chapter 5. We expect such a model to produce a Gaussian distribution with mean closer to 1 and a narrower standard deviation.

5. Developing Word-Embeddings Model for Facet Generation Tasks

As proposed in the last chapter, we would like to create a word-embeddings model that takes into consideration the relevance of position while training. We call this *positional-co occurrence.* We shall train a skip gram model that takes into consideration the position of the words that co-occur. If two words X and Y have the same contextual predictions while training, then they can be used in place of each other as a substitute which prompts us to believe that they belong to the same category. And in training this model, these words that have similar contextual predictions, in the embeddings model would have similar encoding.

We shall train two models - *classic skip gram* and *modified skip gram* and compare their performance on previous datasets that we have extracted and check if the *modified skip gram* model is able to meet our expectations, and whether encoding positional information while training a word-embeddings model can give us a word-embeddings model that is suitable fo the tasks of facets generation. The *modified skip gram* would be able to accommodate the positional information while training. We train both the models for 50 epochs and measure the relative performance of both the models to test the ability of *modified skip gram model* for the task of facets generation.

5.1 Preprocessing

We first extract the 2 percent of the wikipedia data and then preprocess it according to standard procedures. These procedures include cleaning the data-set, removing numbers, punctuation, stop-words, converting to lower strings etc. I used the resources provided

through the webis repository for some extraction and preprocessing. Others were done on my own. Most efficient word-embeddings models use procedures like subsampling, negative sampling, hierarchical softmax etc. This thesis is the first step in developing a word-embeddings model that accommodates positional information while training, I am unsure about how to accommodate these advanced procedures in the training. So, for this thesis I have restricted to use only negative sampling as one of the procedures to be used while training both the word-embeddings models - *classic skip gram* and *modified skip gram*.

After doing all the extracting and cleaning the data, we prepare our n- grams from the extracted wikipedia dataset. In this thesis, we stick to the parameters used by the original paper [17]. So in our n-grams, we have n = 5 and we use a window size of 2. Since, we are training a skip gram model, we need an input vector and a context vector. Consider following n-gram(with n=5):

"history germany east berlin gate"

For the above n-gram, *east* is the input vector and [history, germany, berlin, gate] these words are our context vectors. The context vectors are derived because of our setting of the hyper-parameter of the window size of two. These two words are words before and after the input vector *east*.

Another important thing to consider is the size of word-embeddings. This is dependent on two factors:

1. Vocabulary size
2. Number of dimensions.

Although, we would have different vocabulary sizes for each of our word-embeddings models, the number of dimensions are 100 for both the word-embeddings models.

5.1.1 Preparing original skip gram model

We can first generate our vocabulary from the n-grams itself for which we train the embeddings. We then train the classic *skip-gram* model using pytorch library in python.

This training we make use of negative sampling to speed up the process and improve the performance of the word-embeddings model. The concept of negative sampling and its intended purpose can be studied in the original paper by Mikolov et. al. [17]. We train the embeddings for 50 epochs for the n-grams that we have generated from 2 % of our wikipedia dataset. During the training, we also use the process of negative sampling. The code for generating the original embeddings can be found on the webis gitlab project "pytorch WS-19".

After training the word-embeddings model for 50 epochs, we take the embeddings from the model and save it for comparison later.

5.2 Modified Skip Gram model

In this skip gram, we want to accommodate positional information while training our word-embeddings models. To do this, we modify our n-grams as shown in the figure 5.1. The n-grams for the modified skip-gram are prepared from n-grams for classic skip-grams by adding positions of the context with respect to the input vector. This addition is done at the end of the every word in the context vector of the n-grams. For example, in the figure 5.1, we consider *east* as the input vector and we treat its position as zero. Now, since we have a window-size of two, we consider two words before and after the input word *east*. Since, *history* is at second position before the input vector it is represented as *history-2* in the n-gram for the modified skip-gram. Similar transformation is applied to all the words of context vectors based upon their positional co-occurrence. Hence, in this manner, we accommodate positional information while training the word-embeddings model.

However, we notice that after adding position to words in the context vector, we could have 4 other words for a given word depending upon the position in which they occur. For example, the word *history* could also have *history-2, history-1, history+1, history+2* in training. Hence, we observe that by doing so, the parameter space has increased, because of increase in the vocabulary size. We shall now train the skip-gram models with the modified n-grams that encodes positional information in itself. This training is done as well for 50 epochs and after training is finished, we save the word-embeddings for comparison later.

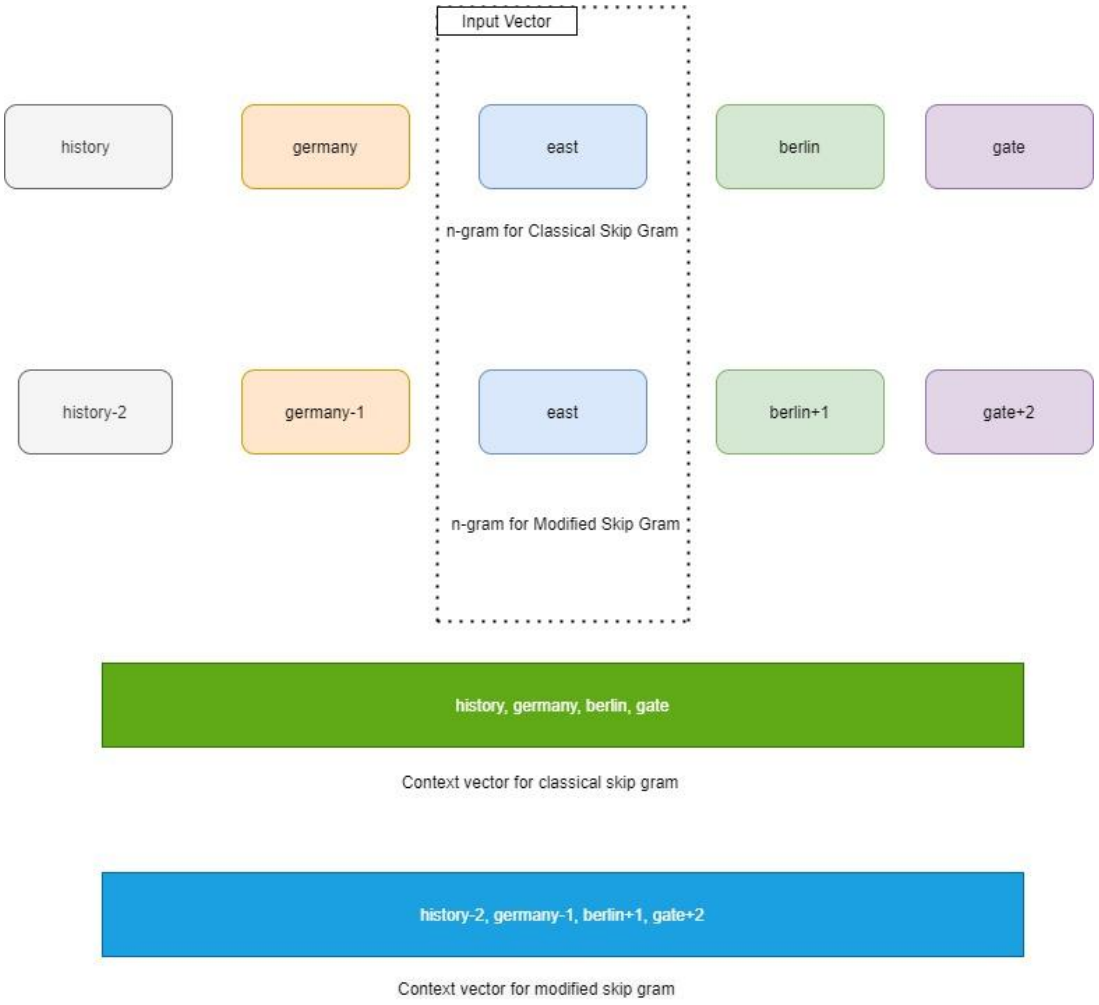

Figure 5.1: Figure demonstrating preparation of n-grams for modified skip-gram. The n-grams for the modified skip-gram are prepared from n-grams for classic skip-grams by adding positions of the context with respect to the input vector.

5.3 Evaluation and analysis: Checking suitability for facets generation tasks

After, we have trained both the models, we shall pit them against each other for various procedures as discussed in chapter 4. We first run the procedure as defined in figure 4.1 for the single *support* as we did in chapter 4 and then again for the multiple *support*.

5.3.1 Analysis on Single support and single True Query

We run the evaluation procedure as discussed in section 4.3.1 and 4.3.2 for both our embeddings models. Hence, we evaluate on term lists generated from synset directly and term lists generated from hypernym traversal. From each term list, we extracted a pair of lemmas, where in each pair one lemma would act as *support* and the other lemmas as *true query*. After this process, we run the evaluation procedure as described in section 4.1. In this step we have run the evaluation procedure for classic skip-gram and modified skip-gram models.

After running the evaluation procedure, we create the density plot of the similarity distributions for both types of term lists and these density plots have been shown in figure 5.2 and figure 5.3. We further summarize the results of our evaluation in table 5.1.

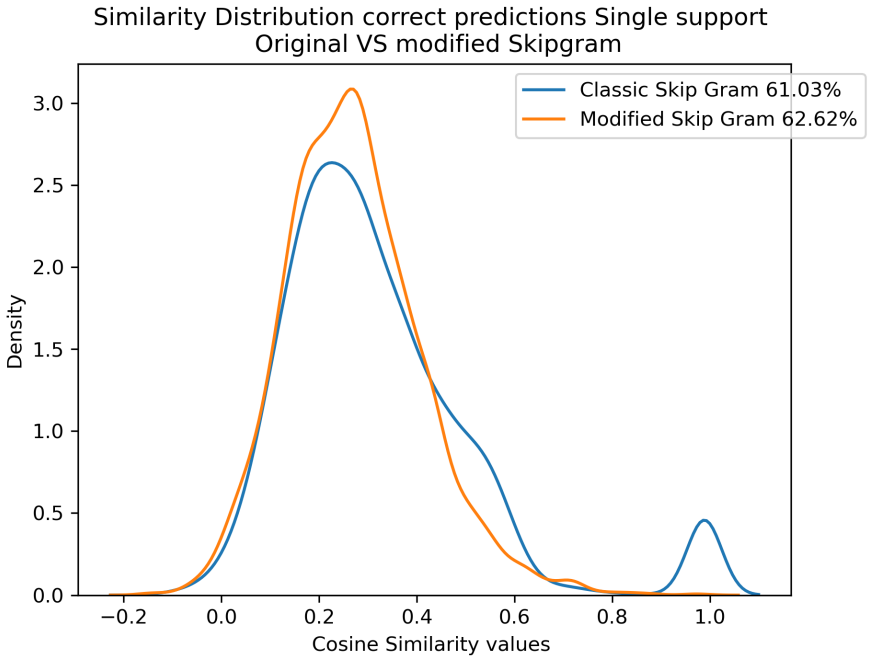

Figure 5.2: Figure demonstrating similarity distribution for correct predictions for term lists directly extracted from the synset. Here the word-embeddings models have correctly identified s*upport-true query* as being more similar than *support-random* lemma. Here, support is a single lemma. This density plot has been developed for the original and modified skip gram model.

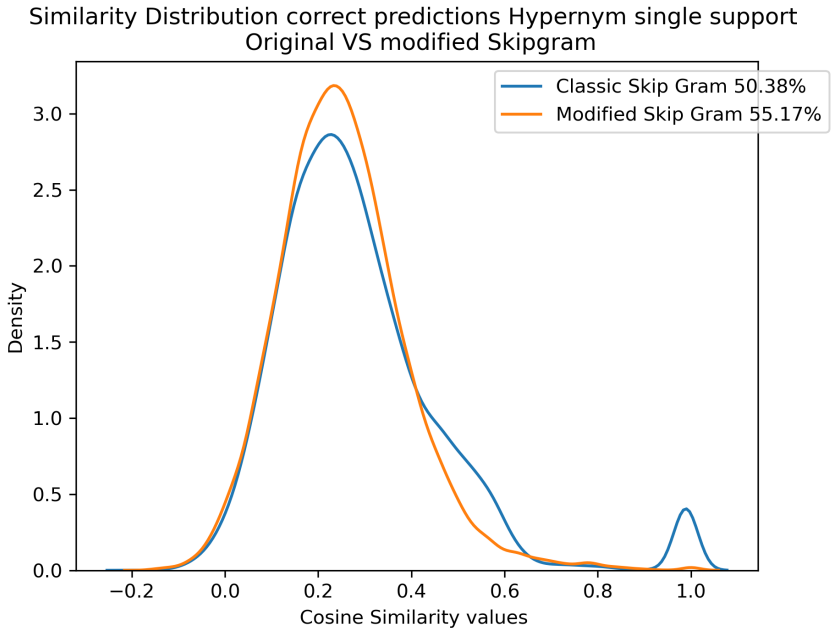

Figure 5.3: Figure demonstrating similarity distribution for correct predictions for term lists containing hypernym relations. Here the word-embeddings models have correctly identified *support-true query* as being more similar than *support-random* lemma. Here, support is a single lemma. This density plot has been developed for the original and modified skip gram model.

Although the models are developed very rudimentarily, we can observe from the above figures that there is no change in the mean of gaussian distribution for the modified skip-gram models over the classic skip-gram model. Further summary of the gaussian distribution for correct and incorrect predictions is provided in table 5.3 and 5.4 respectively.

Table 5.1: Table summarizing results by evaluation procedure for similarity comparisons on analysis of Single Support and single True Query for Original and Modified Skip-gram

Dataset prepared only from synset directly	**Original Skip-Gram**	**Modified Skip-Gram**
Correct Predictions: Support and True query with higher similarity and number of correct predictions	61.03 % 5513	62.62 % 4278
Incorrect Predictions: Support and Random word with higher similarity and number of incorrect predictions	38.97 % 3521	37.38 % 2554

Dataset prepared from Hypernym Traversal on WordNet	**Original Skip-Gram**	**Modified Skip-Gram**
Correct Predictions: Support and True query with higher similarity and number of correct predictions	50.38 % 18159	55.17 % 15585
Incorrect Predictions: Support and Random word with higher similarity and number of incorrect predictions	49.52 % 17887	44.83 % 12666

5.3.2 Analysis on Multiple Support and single True Query

We run the evaluation procedure as discussed in section 4.4.1 and 4.4.2 for both our embeddings models. Hence, we evaluate on term lists generated from synset directly and term lists generated from hypernym traversal. From each term list, we used two lemmas to generate a *support* and one lemma as a *true query*. After this process, we run the evaluation procedure as described in section 4.1. In this step we have run the evaluation procedure for classic skip gram and modified skip-gram models.

After running the evaluation procedure, we create the density plot of the similarity distributions for both types of term lists and these density plots have been shown in figure 5.4 and figure 5.5 . We further summarize the results of our evaluation in table 5.2.

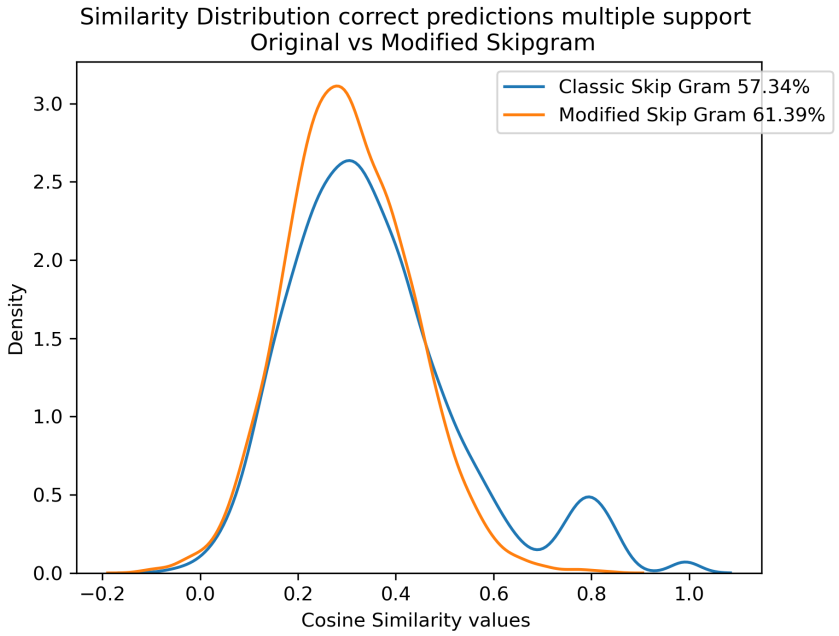

Figure 5.4: Figure demonstrating similarity distribution for correct predictions for term lists directly extracted from the synset. Here the word-embeddings models have correctly identified *support-true query* as being more similar than *support-random lemma*. Here, *support* has two lemmas. This density plot has been developed for the original and modified skip gram model.

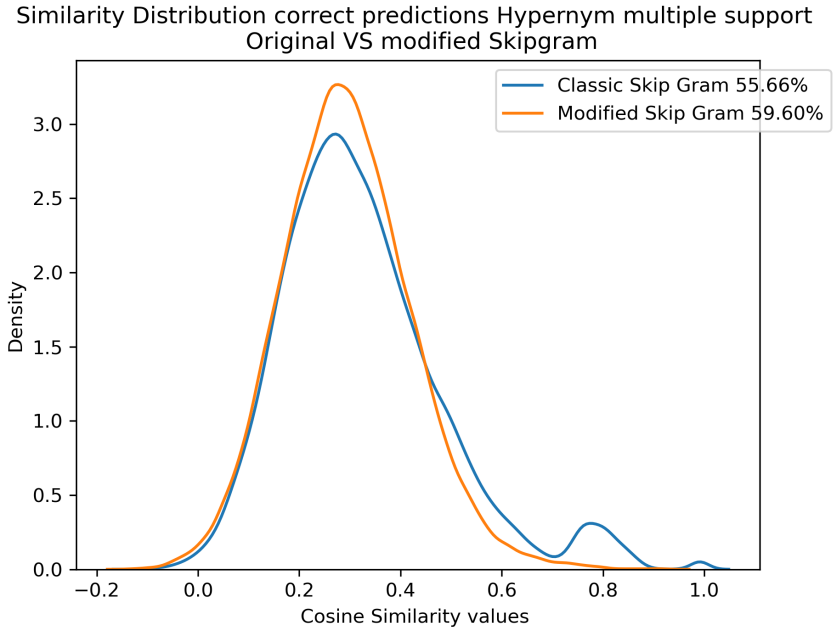

Figure 5.5: Figure demonstrating similarity distribution for correct predictions for term lists containing hypernym relations. Here the word-embeddings models have correctly identified *support-true query* as being more similar than *support-random lemma*. Here, *support* has two lemmas. This density plot has been developed for the original and modified skip gram model.

When multiple support is used, the word-sense disambiguation increases and hence, the mean of the gaussian distribution increases when moving from single *support* to multiple *support.* This is true for both the models that we developed. However, when we compare classic skip-gram model with modified skip-gram model, we observe no significant change in the mean of the gaussian distribution for modified skip-gram model. Further summary of the gaussian distribution for correct and incorrect predictions is provided in table 5.3 and 5.4 respectively.

Table 5.2: Table summarizing results by evaluation procedure for similarity comparisons on analysis of Multi Support and single True Query for Original and Modified Skip-gram

Dataset prepared only from synset directly	**Original Skip-Gram**	**Modified Skip-Gram**
Correct Predictions: Support and True query with higher similarity and number of correct predictions	57.34 % 7837	61.39 % 5689
Incorrect Predictions: Support and Random word with higher similarity and number of incorrect predictions	42.66 % 5831	38.61 % 3578

Dataset prepared from Hypernym Traversal on WordNet	**Original Skip-Gram**	**Modified Skip-Gram**
Correct Predictions: Support and True query with higher similarity and number of correct predictions	55.66 % 62738	59.60 % 49438
Incorrect Predictions: Support and Random word with higher similarity and number of incorrect predictions	44.34 % 49978	40.40 % 33509

5.4 Evaluation Analysis

We expected that the mean for the modified model would be greater than the original model and standard deviation shall become lesser. But, this does not happen as this is depicted in the figures above. Also, when we take a closer look at the standard deviation, we see it decreases for both correct and incorrect predictions for the modified version of skip gram model. Table 5.3 and 5.4 mentions the summary of the gaussian distribution for correct and incorrect predictions for classic skip-gram and modified skip-gram model.

Table 5.3: Table summarizing mean and standard deviation for different evaluation procedures by classic and modified skip-grams models we developed for the correct predictions.

Dataset prepared only from synset directly	**Single Support Mean and Standard Deviation(SD)**	**Multiple Support Mean and Standard Deviation(SD)**
Classic Skip Gram	Mean = 0.31 SD = 0.19	Mean = 0.35 SD = 0.17
Modified Skip Gram	Mean = 0.27 SD = 0.13	Mean = 0.30 SD = 0.12

Dataset prepared from Hypernym Traversal on WordNet	**Single Support Mean and Standard Deviation(SD)**	**Multiple Support Mean and Standard Deviation(SD)**
Classic Skip Gram	Mean = 0.28 SD = 0.18	Mean = 0.32 SD = 0.16
Modified Skip Gram	Mean = 0.25 SD = 0.13	Mean = 0.30 SD = 0.12

Table 5.4: Table summarizing mean and standard deviation for different evaluation procedures by classic and modified skip-grams models we developed for the incorrect predictions.

Dataset prepared only from synset directly	**Single Support Mean and Standard Deviation(SD)**	**Multiple Support Mean and Standard Deviation(SD)**
Classic Skip Gram	Mean = 0.30 SD = 0.21	Mean = 0.30 SD = 0.19
Modified Skip Gram	Mean = 0.23 SD = 0.11	Mean = 0.28 SD = 0.11

Dataset prepared from Hypernym Traversal on WordNet	**Single Support Mean and Standard Deviation(SD)**	**Multiple Support Mean and Standard Deviation(SD)**
Classic Skip Gram	Mean = 0.27 SD = 0.21	Mean = 0.30 SD = 0.18
Modified Skip Gram	Mean = 0.23 SD = 0.12	Mean = 0.27 SD = 0.11

This analysis shows us that this manner of modifying skip-gram architecture does not allow us to achieve word-embeddings that can be used for facets generation tasks. Two intuitive explanations for this result are:

1. The parameter space for word-embeddings for modified skip-gram is 4 times more than for the classic skip-gram and hence, it might need more training to give some efficient results.
2. We observe from the summary presented in table 5.3 and 5.4 that the standard deviation (SD) decreases across both correct and incorrect predictions for the modified skip-gram as compared to classic skip-gram model. This is because it is not only bringing related words together but also un-related words closer to each

other. In this process, most of the space in the word-embeddings model for modified skip-gram is sparse and the entire words that are trained are moved to a high density area of space. And because of this, it is harder to decipher among unrelated and similar words.

Hence, we come to the conclusion that this method of modifying skip-gram architecture can not be used for the tasks of facets generation.

6. Conclusion

One of the important conclusions in the task facet generation through word-embeddings arises from its inability to represent multiple words that represent a single meaning. This can be observed from the analysis that when “Collie” is the given to the state-of-the-art “Google’s word2vec model”, other breed of working dogs like “Eskimo dog”, “Tibetan mastiff”, “Siberian Husky”, “Schipperke” are not presented as facets. Rather, cities like “Bunbury”, “Albury” are suggested which are completely irrelevant. On speculation, two significant reasons are suggested:

1. Representations of many multiple words that convey single meaning do not exist in the word-embedding model.
2. In the training corpus, there is a document on “Collie” in “Bunbury” and not enough documents on the categories of working dogs are together.

A specific observation to be noted from point 2 above is, even though enough training is provided, to move similar words to a more proximate space is not possible for all types of facets. The work of Saedi-et-al [18], we already know that conversions of semantic networks to semantic spaces performs substantially superior to word embeddings based on large collections of texts. And hence, while training word-embeddings on large collections of texts, associated words are always a nuisance for facets generation tasks as they are more closer than similar words when there are not enough documents for all facets categories.

Therefore, a significant effort should be made to bridge the gap between semantic networks (WordNet, DBpedia etc.) to semantic spaces (word-embeddings) for establishing more proximity among words that belong to similar categories. These forms

of semantic spaces (word-embeddings models) are much better suited to tasks of facets generation than semantic spaces trained on large collections of texts. Herein lies possible **future opportunities** for using semantic spaces for facets generation tasks.

From our experiments conducted in chapter 5, we expected that use of positional co-occurrence could bring similar words more closer than the associated words. We expected that this model would have higher mean and a shallow standard deviation. But this was not the case. After training for 50 Epochs, when we compared this model, with the standard model, we observed that there occurs a shallowing of standard deviation across both correct and incorrect predictions. This can be observed from significant change in values of Standard Deviation of similarity measures noted in tables 5.3 and 5.4 in chapter 5.

This further implies that it brings word vectors to a high density area of space, and the remaining space is sparse. But this definitely is not a good thing as this shows that this method of training moves all the words to a much closer space from where it becomes harder to decipher among unrelated and similar words and hence this approach is not a good approach for tasks of facets generation.

Another intuitive explanation for such behaviour is vocabulary size, since we increased the vocabulary size four times, the parameter space is four times and hence this form of model development might need more training examples or more training time or both and this could be the reason why we lose performance, when we train for 50 epochs. This reasoning hence calls for possibly more training with more time and more data.

6.1 Future Work

1. The conversions of semantic networks into semantic spaces has gained momentum recently and the word-embeddings models so obtained have performed substantially better than word-embeddings models trained on large collections of texts for semantic similarity tasks. However, these models have not been used for the tasks for facets generation and present a huge potential to provide algorithmic support for the tasks of facets generation. Further, we need

some procedure to combine two types of word-embeddings - embeddings trained on large texts of data and embeddings derived from semantic networks. This bridging of the gap, would be able to draw a more stricter line between related words and associated words and facilitate the ability of word-embeddings models trained on large corpus of texts for the tasks of facets generation and further improve the efficiency of the word-embeddings models for other tasks as well.

2. Despite this unsuccessful attempt, we learned from our analysis in chapter 4, when we moved from single support to multiple support, we observed that SD decreases and mean increases, this means that Word embeddings models are able intuitively able to catch the implied senses when multiple queries are presented and is a potential candidate for facets generation and hence, other techniques should be considered for the developing word-embeddings models for the tasks of facets generation.

Bibliography

[1] Elekes, Á., Schäler, M., & Böhm, K. (2017, June). On the various semantics of similarity in word embedding models. In *2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL)* (pp. 1-10). IEEE. [Abstract], 1, 2.4, 4.5

[2] Medelyan, O., Witten, I. H., Divoli, A., & Broekstra, J. (2013). Automatic construction of lexicons, taxonomies, ontologies, and other knowledge structures. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *3*(4), 257-279. [Pre-requisites], 2, 3.5, 3.5.2

[3] Fellbaum, C. (2012). WordNet. *The encyclopedia of applied linguistics*. 1, 2

[4] Moldovan, D. I., & Mihalcea, R. (2000). Using WordNet and lexical operators to improve internet searches. *IEEE Internet Computing*, *4*(1), 34-43. 2.1

[5] Dakka, W., Ipeirotis, P. G., & Wood, K. R. (2005, October). Automatic construction of multifaceted browsing interfaces. In *Proceedings of the 14th ACM international conference on Information and knowledge management* (pp. 768-775). 2.3

[6] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*. 1

[7] Giatsoglou, M., Vozalis, M. G., Diamantaras, K., Vakali, A., Sarigiannidis, G., & Chatzisavvas, K. C. (2017). Sentiment analysis leveraging emotions and word embeddings. *Expert Systems with Applications*, *69*, 214-224. 1

[8] Ye, X., Shen, H., Ma, X., Bunescu, R., & Liu, C. (2016, May). From word embeddings to document similarities for improved information retrieval in software

engineering. In *Proceedings of the 38th international conference on software engineering* (pp. 404-415). 1

[9] Zuccon, G., Koopman, B., Bruza, P., & Azzopardi, L. (2015, December). Integrating and evaluating neural word embeddings in information retrieval. In *Proceedings of the 20th Australasian document computing symposium* (pp. 1-8). 1

[10] Musto, C., Semeraro, G., de Gemmis, M., & Lops, P. (2016, March). Learning word embeddings from wikipedia for content-based recommender systems. In *European Conference on Information Retrieval* (pp. 729-734). Springer, Cham. 1

[11] Ozsoy, M. G. (2016). From word embeddings to item recommendation. *arXiv preprint arXiv:1601.01356*. 1

[12] Levy, O., Goldberg, Y., & Dagan, I. (2015). Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, *3*, 211-225. 2.3

[13] Feddoul, L., Schindler, S., & Löffler, F. (2019, September). Automatic Facet Generation and Selection over Knowledge Graphs. In *International Conference on Semantic Systems* (pp. 310-325). Springer, Cham. 1, 2.3, 2.4

[14] Jing, H. (1998). Usage of WordNet in natural language generation. In *Usage of WordNet in Natural Language Processing Systems*. 2.1

[15] Richens, T. (2008, August). Anomalies in the WordNet verb hierarchy. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)* (pp. 729-736). 2.1

[16] Dimitrova, T. S. V. E. T. A. N. A., & Stefanova, V. A. L. E. N. T. I. N. A. (2016). Adjectives in Wordnet: Semantic issues. In *Proceedings of the 12th International Conference Linguistic Resources and Tools for Processing the Romanian Language* (pp. 131-141). 2.1

[17] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (pp. 3111-3119). 2.4, 5.1, 5.1.1

[18] Saedi, C., Branco, A., Rodrigues, J., & Silva, J. (2018, July). Wordnet embeddings. In *Proceedings of the third workshop on representation learning for NLP* (pp. 122-131). 2.3

[19] Grover, A., & Leskovec, J. (2016, August). node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 855-864). 2.3

[20] Perozzi, B., Al-Rfou, R., & Skiena, S. (2014, August). Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 701-710). 2.3

[21] Roweis, S. T., & Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *science*, *290*(5500), 2323-2326. 2.3

[22] Belkin, M., & Niyogi, P. (2003). Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, *15*(6), 1373-1396. 2.3

[23] Tzitzikas, Y., Manolis, N., & Papadakos, P. (2017). Faceted exploration of RDF/S datasets: a survey. *Journal of Intelligent Information Systems*, *48*(2), 329-364. 2.3

[24] Mäkelä, E., Hyvönen, E., & Saarela, S. (2006, November). Ontogator—a semantic view-based search engine service for web applications. In *International Semantic Web Conference* (pp. 847-860). Springer, Berlin, Heidelberg. 2.3

[25] Schraefel, M.C., Smith, D.A., Owens, A., Russell, A., Harris, C., Wilson, M.: The evolving mSpace platform: leveraging the semantic web on the trail of the Memex. In: Proceedings of the Sixteenth ACM Conference on Hypertext and Hypermedia, HYPERTEXT 2005, pp. 174–183. ACM (2005). 2.3

[26] Huynh, D. F., & Karger, D. (2009, April). Parallax and companion: Set-based browsing for the data web. In *WWW Conference. ACM* (Vol. 61). 2.3

[27] Moreno-Vega, J., & Hogan, A. (2018, October). GraFa: Scalable faceted browsing for RDF graphs. In *International Semantic Web Conference* (pp. 301-317). Springer, Cham. 2.3

[28] Papadakos, P., & Tzitzikas, Y. (2014). Hippalus: Preference-enriched Faceted Exploration. In *EDBT/ICDT Workshops* (Vol. 172). 2.3

[29] Hahn, R., Bizer, C., Sahnwaldt, C., Herta, C., Robinson, S., Bürgle, M., & Scheel, U. (2010, May). Faceted wikipedia search. In *International Conference on Business Information Systems* (pp. 1-11). Springer, Berlin, Heidelberg. 2.3