Bonn-Aachen International Center for Information Technology (B-IT)

University of Bonn

Master Programme in Life Science Informatics

Master Thesis

Comparative analysis of protein function text-based embeddings and its potential for prediction tasks

Submitted by

Rohitha Ravinder

First Supervisor: Prof. Dr. Dietrich Rebholz-Schuhmann

Second Supervisor: Prof. Dr. Martin Hofmann-Apitius

Internal Supervisor: Dr. Leyla Jael Castro

April 17, 2023

In collaboration with ZB MED - Information Centre for Life Sciences

Acknowledgement

I would like to express my appreciation to Prof.Dr.Dietrich Rebholz-Schuhmann for giving me the opportunity to work under his guidance and supervision at ZB MED - Information Centre for Life Sciences. His support, feedback and the freedom he provided me in my work have played a vital role in shaping my research, and I am thankful for the opportunity to be a part of his group.

I am grateful to Dr. Leyla Jael Castro for her exceptional guidance and support throughout my project. Her mentorship has been invaluable in shaping my research, and her willingness to provide feedback and answer my queries has been greatly appreciated. I am deeply thankful for her time, effort, and commitment to my academic and professional growth.

I am thankful for the encouragement and support provided by Prof. Dr. Martin Hofmann-Apitius throughout my thesis. His dedication and commitment to my research, as evidenced by the time he took to review my work, have been invaluable. I appreciate the effort he put into helping me succeed and the role he played in the successful completion of my thesis.

I would also like to thank Dr. Olga Giraldo for her assistance in the downstream analysis of this project.

Furthermore, I feel privileged to have had the opportunity to study at Bonn-Aachen International Center for Information Technology (b-it). The challenging academic environment provided during my masters in Life Sciences Informatics has been instrumental in shaping my academic growth.

Lastly, I am thankful for the unwavering motivation and belief in me from my family, friends, and colleagues. Without their support, this endeavor would not have been possible, and I am grateful for their constant encouragement throughout my master's degree.

Abstract

This thesis addresses the challenging task of predicting protein function in bioinformatics, which has traditionally been addressed using embeddings to learn representations of protein sequences and infer function. However, to date, no studies have explored the use of embeddings generated based on protein function text for predicting protein function. The main objective of this study is to enhance our understanding of how text-based embeddings can improve protein function annotations. To achieve this, we specifically aim to compare and evaluate selected text-based embedding approaches exploiting protein function related information, and explore the potential of text-based embeddings for function prediction tasks using direct propagation techniques.

We specifically propose to learn and explore text-based embedding representations of protein function comment sections kept as part of the Swiss-Prot entries. A comparative study is conducted using these text-based embeddings derived from two approaches which include a combination of natural language processing frameworks such as Word2Vec and Named Entity Recognition, as well as direct propagation techniques such as sequence similarity and by-similarity prediction.

Our results demonstrate the potential of embeddings in propagating protein function. However, this is not conclusive as further exploration is necessary, including the inclusion of TrEMBL entities and the use of metrics beyond sequence similarity. This study serves as a preliminary assessment of the potential and behavior of text-based embeddings for improving protein function annotations.

Keywords: Protein function prediction, Natural Language Processing, Word embeddings, Named Entity Recognition

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Motivation

Understanding the role of proteins is crucial to life. However, there is only a small subset of proteins whose function is well characterized, thereby making protein function prediction a fundamental task in the field of Bioinformatics. Numerous techniques have been developed for protein function prediction using sequence similarity, sequence-based embeddings, protein structures or protein-protein interactions [1]. Nevertheless, to the best of our knowledge no research has been made yet that makes use of protein function text-based embeddings to evaluate their use for protein function prediction tasks. In this study, our goal is to get a better understanding of how information for protein functions can be exploited through embeddings so that the produced information can be used to improve protein function annotations.

Our work is based on the hypothesis that states a direct correlation between sequence similarity (corresponding to the BLAST [2] identity score) and similar biological function (as expressed in the protein function comment). The idea here is to capture this correlation with the help of the corresponding protein embeddings. Here, we consider the text-based embeddings that are derived from the protein function comment sections of the UniProtKB [3] reviewed entries: Swiss-Prot entries. Specifically, we aim to learn and compare two embedding models that map functions of protein to sequences of vector representations such that two proteins having similar function as stated in the function comment section appear closer in the embedding vector space. The evaluation covered by this thesis will offer a preliminary assessment based mostly on direct inspection and combination of the information provided by the original text and the derived embeddings.

1.2 Aim

The aim of this thesis is as follows:

- To compare and evaluate selected text-driven embedding approaches exploiting protein function related information.
- To explore the potential offered by text-driven embeddings for its use in function prediction tasks
- To compare protein sequence similarity vs word embedding similarity based on protein functional annotations (i.e., function comments).
- To evaluate text-driven approaches as an option to provide additional support to biologists working on protein functional annotations.

1.3 Contributions

The contributions of this thesis can be summarized as follows:

- Definition of two text-based embedding methods to represent protein function comments with its corresponding software and optimization framework [4].
- Vector space of protein function-based embeddings that can be used for further tasks such as vocabulary analyses [5].
- Dataset of proteins that could be clustered based on BLAST sequence similarity and embedding similarity. This dataset can be utilized by UniProt curators and developers to revise and refine their clustering approaches [5].

1.4 Outline

The rest of the thesis is organized in the following manner:

- Chapter 2 presents the essential background information regarding the datasets used in the current study and offers a comprehensive overview of the key concepts and theoretical framework, with a particular emphasis on word embeddings.
- Chapter 3 covers the methods used in this study, including the design of the two novel text-based embedding approaches used in this thesis, and the implementation details. This includes the retrieval and preprocessing of the datasets, text annotation, the preprocessing pipeline, and the generation of embeddings.

- Chapter 4 outlines the goals of the evaluation, the experimental setup, and provides a brief summary of the results.
- Chapter 5 presents the results and findings of the study, providing a detailed analysis of the outcomes.
- Chapter 6 provides a thorough discussion of the research, highlighting the significance, limitations of the present work and provides insights into future directions for the thesis.
- Chapter 7 concludes the study and analyzes the key findings of the thesis.

Chapter 2

Background

2.1 UniProt

The UniProt database [6] is a large and comprehensive resource maintained by the UniProt Consortium that contains information about protein sequences, their functions, and their interactions. The UniProt database is divided into several sections, including UniProt Knowledgebase (UniProtKB), UniRef, UniParc, Proteomes and Reference Proteomes. For the purpose of this study, we mainly focused on UniProtKB and UniRef.

UniProt Knowledgebase (UniProtKB) in itself comprises of two main components: UniProtKB/Swiss-Prot and UniProtKB/TrEMBL. UniProtKB/Swiss-Prot is a high-quality protein sequence database that is manually curated, where each protein entry is linked to a summary of the experimentally verified, or computationally predicted functional information added by expert biocurators. While UniProtKB/TrEMBL consists of protein entries that are computationally annotated by automated systems, UniProtKB/Swiss-Prot provides an accurate annotation of each protein entry and offers a rich source of information on protein function, structure, and post-translational modifications. The functional comments of these expert annotated proteins serve as the primary dataset for the generation of embeddings in this study.

UniRef is a comprehensive database of clustered sequences that provides a non-redundant representation of UniProtKB sequences at various levels of sequence identity. Based on the levels of sequence identity, UniRef clusters are categorized into UniRef50, UniRef90 and UniRef100 [7]. For this study, we made use of UniRef90 which is built by clustering UniRef100 sequences at 90% sequence identity. This dataset is employed as the baseline for the aim of this study.

2.2 Natural Language Processing and Embeddings

Natural Language Processing (NLP) is a subfield of artificial intelligence that focuses on enabling computers to understand, analyze, and generate human language [8]. Analysis and generation of such human (natural) language becomes

critical with the prevalence of large unstructured data. This unstructured data becomes more challenging to understand using traditional methods and requires advanced NLP techniques to extract useful insights and information. One of the key techniques used to analyze unstructured textual data is the creation of embeddings [9]. One of the advantages of using embeddings is that it can be pre-trained on large amounts of text data or trained specifically for a task or domain. Text-based embeddings, in particular, are a way of representing words or phrases as numerical vectors that capture their semantic meaning in a given context. This is important because natural language is highly contextual, and the meaning of a word or phrase can vary depending on the context in which it is used. By using text-based embeddings, machine learning models can learn to recognize the meaning of words or phrases in a way that captures their relationships with other words or phrases in a given context.

2.3 Word embeddings

Machine learning models majorly rely on numerical data, especially when working with textual data, a fixed-length representation of words using numeric values is necessary to make it usable for neural networks, such as the one used in this thesis. The challenge is to find numerical features that can be fed into these models to represent words.

One approach commonly used is the one-hot word representation method, where each word is represented by a vector with only one non-zero entry, typically with the length of the vocabulary. This simple method has been widely adopted in natural language processing tasks [10]. To create a one-hot representation, each word in the corpus is assigned a unique index within the vector range, and the resulting vector has zeros at all positions except at the word's specific index, which has the value one. This ensures that every word in the vocabulary is assigned a unique one-hot representation [11].

Although one-hot encoding is a simple approach to represent words as vectors, it has significant drawbacks for use in neural networks due to its high dimensionality and sparsity, making it computationally inefficient, especially for large vocabularies [12]. Another limitation of one-hot encoding is that it cannot represent the semantic similarity between words. Even synonyms are represented as distinct and unrelated vectors in one-hot encoding, which makes it impossible to use similarity measures for understanding the relationships between words. Numeric features can be effective for comparing words, but one-hot encoding does not facilitate the computation of such metrics, which limits its effectiveness for natural language processing tasks [13].

Dense word embeddings have become a widely adopted solution to address the drawbacks of one-hot encodings, such as their computational inefficiency and

lack of semantic meaning. These word embeddings are constructed underlying the distributional hypothesis that proposes that a word's meaning can be inferred from its contextual usage [14]. The hypothesis suggests that words with similar contextual usage tend to have similar meanings thereby incorporating semantics into word representations [15]. Subsequently, the neural network-based representations of the aforementioned distributed hypothesis, known as word embeddings model the relationship between a target word and its context words and are obtained by training the neural network models.

Different types of pretrained word embeddings, such as GloVe [16] and BERT [17], have been created to address the limitations of one-hot encoding. However, for the current study, Word2Vec was selected because it allowed for the creation of a hybrid embedding approach that combines Word2Vec with a dictionary-based Named Entity Recognition (NER) technique, as explained in section 3.1. This approach would not be possible with pretrained embeddings because they lack knowledge of the named entities, which are not actual words found in the texts used for pretraining. In contrast, pretrained embeddings are limited by their pretraining text corpus, which may not include all of the named entities relevant to a specific task. This makes them less effective for tasks that require knowledge of named entities.

2.3.1 Word2Vec

Word2Vec is a well-known algorithm for creating word embeddings which trains a neural network to capture conceptual information about each target word by considering its surrounding words or by using a target word to learn its surrounding context making the resulting embeddings more informative than the individual words alone [18].

Word2Vec has two main architectures - Continuous Bag of Words (CBOW) and Skip-gram (SG) - and can be viewed having three main components: Vocabulary Builder, Context Builder, and Neural Network [19]. While both architectures follow the same vocabulary-building process, the input transformation for context construction and the neural network architecture differ between them. CBOW and SG are similar in the sense that words sit in a context of words, which is reflected by the embeddings so words and their common context will commonly be close to each other. In both cases, the main output is a multidimensional vector space where each word in the vocabulary is assigned a vector. This makes it easier to perform mathematical operations to, for instance, find the most similar, i.e., closest in the vector space, words to a given one.

The Vocabulary builder is a crucial component of the Word2Vec model, which extracts all unique words from raw text to build the vocabulary. The vocabulary includes the word index and its frequency in the text. This is done by

iterating over each word in the corpus and storing the word in the Vocabulary Object along with its frequency. This object is later pruned using a user-defined minimum word count and maximum vocabulary size. The resulting Vocabulary object is stored as a hash table. The hash table has a fixed size as defined by the maximum vocabulary size, and when a new word is encountered, its count is incremented. This process is repeated until all words have been processed. Context Builder and the Neural Network layers for both the architectures are explained in the respective sections.

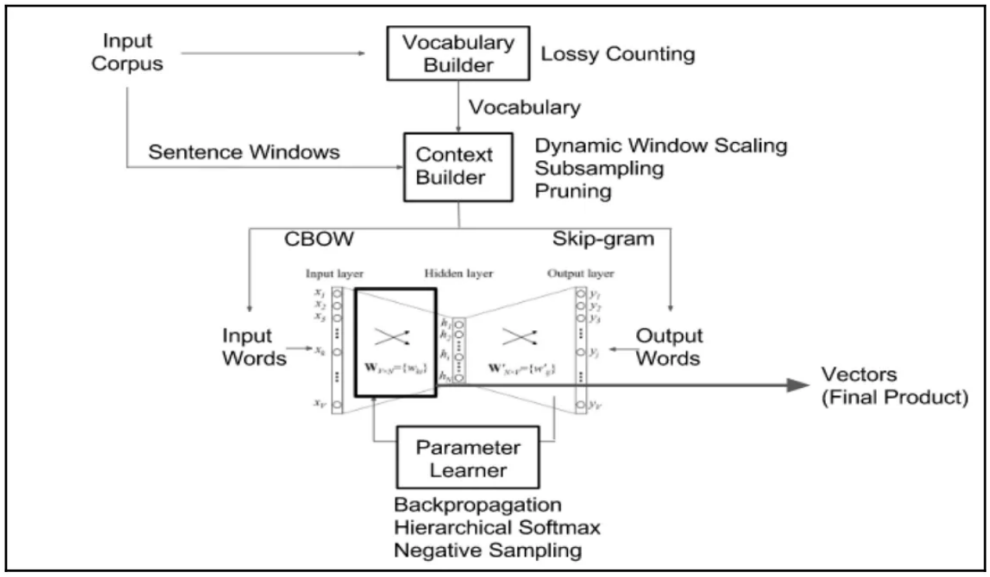


Figure 2.1: Overview of Word2Vec Architecture [19]

2.3.1.1 Skip-gram (SG)

The Skip-gram architecture is a neural network model that predicts the context words for a given target word as shown by Figure 2.2. In this process, the input sentences are converted into input-output pairs. The window size determines the number of neighboring words used to create input-output pairs. Negative sampling is used to select a limited number of combinations of the middle word and words that did not appear in its context. Additionally since the relatedness of words declines once moved further away from a certain word, combinations of the middle word and words in its context, but relatively far away, are sampled less frequently [18].

The Word2Vec model uses a neural network with three layers: an input layer, a hidden layer, and an output layer as shown by Figure 2.3. The input layer has as many neurons as there are words in the vocabulary, and the output layer has the same number of neurons as the input layer. The hidden layer has a

dimensionality of neurons that corresponds to the dimensionality of the resulting word vectors.

The example sentence:

"Machine learning predicts protein function and sequence"

can be used to illustrate how the Skip-gram architecture and Word2Vec model work. If the target word is 'protein' and the window size is 2, then the neighboring words 'learning', 'predicts', 'function', and 'and' are used to create input-output pairs. This means that the input is "protein" and the output is each of the neighboring words. Therefore, the input-output pairs would be (protein, learning), (protein, predicts), (protein, function) and (protein, and).

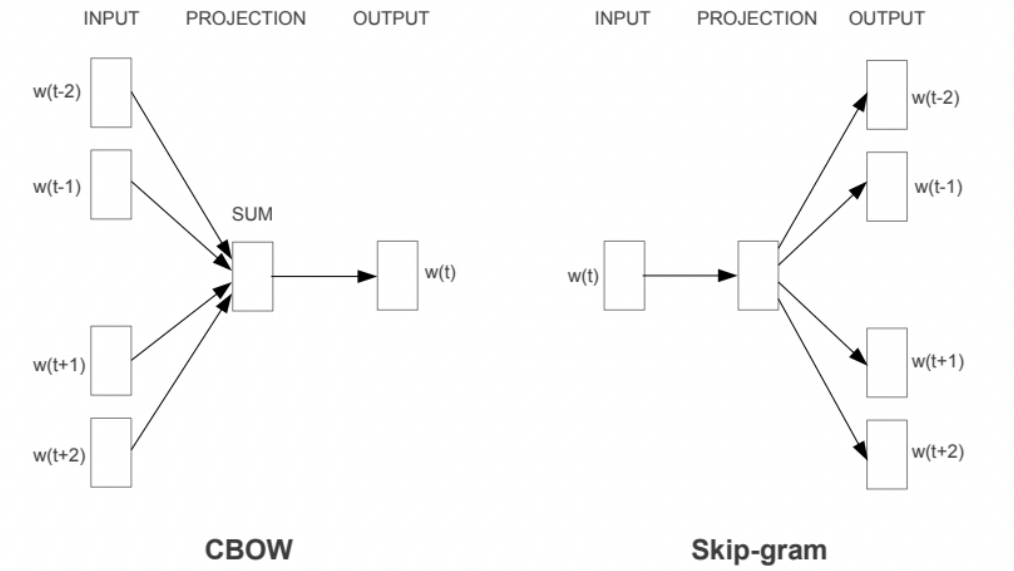


Figure 2.2: Continuous Bag of Words versus Skip-gram [20]

The neural network model is trained by going through each input sentence and converting it into suitable input-output pairs using gradient descent and backpropagation [21]. During the training process, backpropagation is done in one back pass, and error vectors are calculated for each target word. The weights of the hidden layer are updated based on the cumulative error vector. During training, the network learns to predict the context words for a given target word, based on the relationships between the words in the dataset. Once the training process is finished, the word embeddings are obtained by extracting the weight matrix of the projection layer [18].

2.3.1.2 Continuous Bag of Words (CBOW)

Continuous Bag of Words differs in the way that it predicts the current word based on its surrounding context words. More specifically, given a sequence of words, the CBOW model learns to predict the probability of each word in the sequence given the context words within a fixed window. The context words are represented as one-hot vectors, where each dimension represents a unique word in the vocabulary.

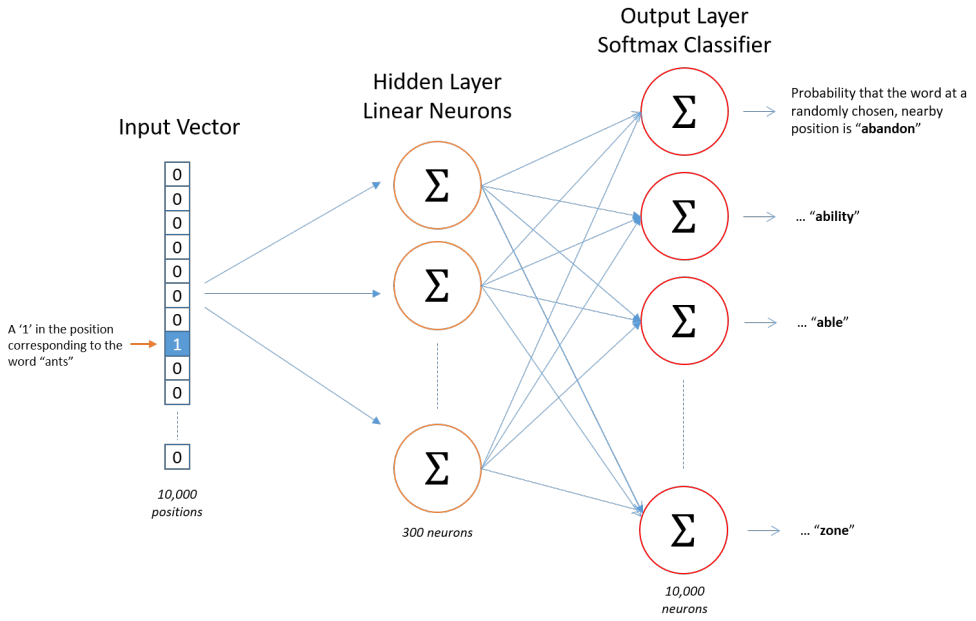


Figure 2.3: Neural Network Architecture [19]

Unlike the Skip-gram model, CBOW predicts the target word based on the context, rather than predicting the context words based on a target word as shown by Figure 2.2.

To illustrate this, let's consider the same example sentence:

"Machine learning predicts protein function and sequence"

If the window size is 2, and the target word is 'protein', CBOW will predict this word based on the surrounding context words, which are 'learning', 'predicts', 'function', and 'and'. Therefore, the input and output pairs for this example sentence would be (learning, protein), (predicts, protein), (function, protein), and (and, protein).

Similar to Skip-gram, CBOW also uses a three layer neural network for training. However, CBOW's architecture is slightly different. The input layer represents the context words, and the output layer represents the target word. The hid-

den layer, which is the same size as the input layer, serves as a bottleneck that reduces the dimensionality of the input.

During training, CBOW uses the context words to predict the target word. The backpropagation algorithm is used to calculate the error between the predicted and actual target words, and the weights of the neural network are updated accordingly. Once the training process is complete, the weight matrix of the hidden layer can be used as word embeddings to represent the words in a lower-dimensional space.

2.4 Document embeddings

Word embeddings have a limitation in that they lose the order of words, which can result in different sentences having the same representation. To address this, the Doc2Vec (Paragraph Vector) framework was introduced as an unsupervised approach to generate continuous vector representations for text pieces of varying lengths, ranging from phrases to documents . The name "Paragraph Vector" emphasizes its ability to handle variable-length texts and generate vector representations not only for individual words but also for entire documents [22].

Another approach to generate document-level embeddings is to use word embeddings based on the context in which they appear and then combine them to represent the entire document. Several methods exist to generate document embeddings from word embeddings, including averaging word embeddings, weighted averaging, and summation [23, 24]. Weighted Averaging is a method that assigns weights to individual words based on their importance in the document. One example of a weighting scheme is the tf-idf (Term Frequency - Inverse Document Frequency) scheme, where words that occur frequently in the document but rarely in the corpus are assigned higher weights while the summation method simply involves the sum of all the word embeddings of the words in the document.

In this study, we use the centroid approach (averaging word embeddings), which computes the centroid of the word embeddings to generate document embeddings. We propose two embedding approaches building on this baseline, which are explained in Section 3.1.

2.5 Named Entity Recognition

Due to the growing abundance of biomedical literature and materials, it has become difficult to efficiently search for and extract useful information [25]. Multiple sources of information along with a transformation of unstructured text data into refined knowledge is critical to facilitate research productivity. However, manual annotation and feature generation by biomedical experts can be

inefficient as they involve complex processes and demand expensive and time-intensive labor [26]. Therefore, the need for effective and precise natural language processing (NLP) methods is growing in importance as they are crucial for computational data analysis making advanced text mining techniques a necessity for automatically analyzing biomedical literature and extracting valuable information from textual data [27]. For extracting valuable information, such as relationships among objects, the identification of significant terms from texts is important. Meaningful terms or phrases in a domain, which can be distinguished from similar objects, are called named entities.

Named Entity Recognition (NER) is a NLP technique that aims to identify and classify these named entities in unstructured text data into pre-defined entity types. NER should be performed prior to tasks, such as relation extraction, since annotated mentions play an important role in research on text mining. A fundamental task of biomedical NLP is the recognition of named entities, such as genes, diseases, chemicals, and drug names, from texts. However, biomedical NER is an especially intricate undertaking because biological entities exhibit several complexities, such as: (i) continually increase with new discoveries, (ii) have numerous synonyms, (iii) frequently being referenced using abbreviations, (iv) being described using phrases, and (v) being composed of combinations of letters, symbols, and punctuation [28]. There are several approaches to biomedical NER, including: Dictionary-based NER , Rule-based NER, Machine learning-based NER, Hybrid NER and Deep Learning-based NER.

2.5.1 Ontologies

Annotating text using controlled vocabulary is a key aspect of NER as it helps to identify and classify specific types of named entities in text using a predefined set of terms or phrases. This section explains the ontologies that were used as a controlled vocabulary to annotate the function comment text as part of the Hybrid-Word2doc2Vec approach. This involves mapping the text to specific terms or concepts in the ontology in order to normalize terms that have multiple meanings or aliases, and thus standardize the text.

2.5.1.1 MeSH (Medical Subject Headings)

MeSH (Medical Subject Headings) is a controlled vocabulary thesaurus used by the National Library of Medicine (NLM) [29] for indexing and searching biomedical literature. MeSH ontology consists of a hierarchy of terms that represent various biomedical concepts such as diseases, chemical and drugs, organisms, anatomy to name a few. These terms are organized into a tree-like structure, where each term is linked to other related terms by a series of broader and narrower terms [30]. The hierarchy allows for a more nuanced understanding of the relationships between different biomedical concepts.

2.5.1.2 Gene Ontology

The Gene Ontology (GO) [6] is a controlled vocabulary used to annotate genes and gene products in a variety of organisms. The GO structure comprises of 'classes' or 'terms' that represent different biological functions, pathways responsible for carrying out various biological processes, and the specific locations within cells where these processes occur as well as relations between these classes that specify the underlying relationship between them. GO categorises these terms into three aspects: molecular function, biological process, and cellular component, and provides a standardized vocabulary to describe the functions of genes and gene products across different species [31].

2.5.2 Integration of Whatizit tool

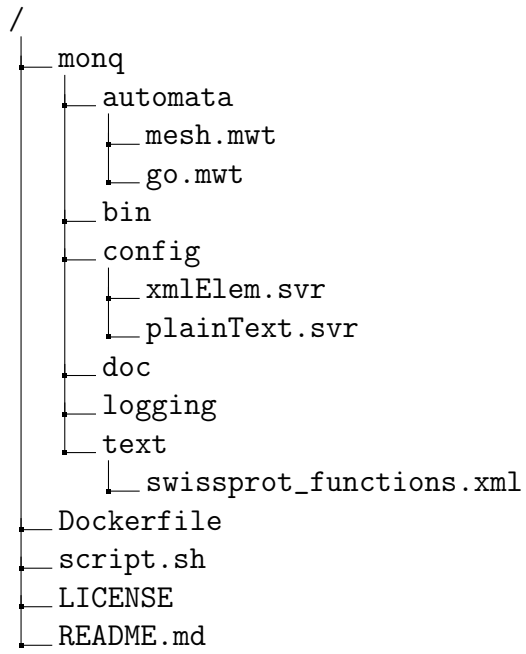
Whatizit is an open-source text processing system developed by the European Bioinformatics Institute (EBI) that provides a range of text-mining and information-extraction functionalities, including NER. Whatizit is designed specifically for the biomedical domain, and it can be used to identify and classify biomedical named entities. Whatizit uses a combination of rule-based and machine learning approaches to identify named entities in text data. It first applies a set of pre-defined rules to identify common patterns and structures that are indicative of named entities. It then uses machine learning algorithms to analyze the context and linguistic features of the text and classify the identified entities into different categories [32].

Whatizit infrastructure consists of a suite of modules that processes and annotates text. Individual modules are composed of a number of internal modules. All modules are implemented in Java partly based on special libraries for the matching of large terminology sets [33]. Whatizit can be used either as a standalone tool or as a web service through its API.

Docker version for a minimal Whatizit

The Dockerized version of minimal Whatizit primarily focuses on the automata component derived from MONQjfa [34]. This container does not include the web-based application or the dictionaries for text-mining that were previously available at EMBL-EBI.

The hierarchy of the Docker-based Whatizit typically involves several components that work together to create and deploy applications. The directory is composed of three primary components: a monq folder, a shell script named script.sh, and a Dockerfile. The monq folder is comprised of six sub-folders, namely automata, bin, config, doc, logging and text. The Docker-based software hierarchy can be depicted as follows:



automata: This folder contains the dictionary files used by the Whatizit tool to perform NER. The dictionaries are in a MWT format. MWT (Markup Word Text) is a specific text format used by the Whatizit tool for annotating and extracting information from text. These dictionaries include information about the entities to be annotated, the entity types as well as how to handle synonyms and variant forms.

bin: This folder includes the configuration scripts needed to run the infrastructure.

config: This folder contains two server files: xmlElem.svr and plainText.svr. These files follow an XML markup structure defining the configuration parameters for starting the corresponding servers. These parameters include a description of the server, its accessibility, hostname or IP address, port number and specifies the command to be executed by the server component when it starts up. The primary command responsible for the annotation process is Java based which makes use of the 'DictFilter' class to process data along with settings such as heap size and input/output encoding specified as parameters.

doc: This file contains the documentation on the MONQ infrastructure.

logging: This folder contains log files for the infrastructure services.

text: This folder contains text files that need to be annotated using Whatizit. There are two formats that can be used for the input text: plain text files or XML files.

The shell script is mainly responsible for starting the 'xmlElem' and the 'plain-

Text' server using the configuration files in the bin folder.

The Dockerfile is utilized for constructing a Docker image comprising a base image of Tomcat server and the necessary packages. Moreover, the Dockerfile declares the command that is executed upon launching a container from this image, which is to execute the 'script.sh' script.

2.6 Cosine similarity

Cosine similarity is a measure of similarity between two vectors, typically used in machine learning and natural language processing (NLP) applications. It measures the cosine of the angle between two vectors in a multi-dimensional space, enabling orientation judgment rather than magnitude evaluation. [35]. As a result, it measures the angle of the vectors and employs it as a similarity metric. Cosine similarity can be used to perform a variety of NLP tasks, such as text classification, document similarity, and recommendation systems. In text classification, cosine similarity can be used to determine the similarity between a test document and a set of training documents, and assign the test document to the most similar class. In document similarity, cosine similarity can be used to compare the similarity between two documents, based on the similarity of their embedded words.

To compute the cosine similarity between two document embeddings, we first represent each document as a vector in a high-dimensional space. Then, we compute the cosine of the angle between the two vectors, which ranges from -1 (completely dissimilar) to 1 (completely similar). A cosine similarity of 0 means that the two vectors are orthogonal, or completely unrelated [36].

Definition

Given two non-zero vectors (A and B), the cosine between them can be derived from the Euclidean dot product:

$$A \cdot B = \|A\| \, \|B\| \cdot cos\Theta \tag{2.1}$$

$$cos\,(\Theta) = \frac{A \cdot B}{\|A\| \, \|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}} (2.1) \tag{2.2}$$

where, A_i and B_i are components of vectors A and B.

Thus, the similarity between two non-zero given vectors A and B is the cosine of the angle between them.

$$similarity(A, B) = cos\Theta_{A,B} \tag{2.3}$$

In the present study, cosine similarity is employed as a metric to measure the similarities between the text comments of Swiss-Prot entries. These entries are transformed into documents and numerical embeddings are generated for each of them to capture their semantic meaning. The embeddings are normalized and their dot product is computed to identify the degree of similarity between any two function texts.

2.7 Sequence similarity

Sequence similarity refers to the degree of similarity or homology between two or more biological sequences, such as nucleotide or amino acid sequences. The sequence of amino acids in proteins determines their structure and function, and as proteins evolve and adapt, sequence similarity reflects functional similarity. This relationship is reflected in the functional annotation of proteins and can provide insight into both their evolutionary relationships and functional properties.

One of the most widely used methods for quantifying sequence similarity is the percentage identity score [37], which measures the percentage of amino acid residues that are identical between two protein sequences and is calculated as stated by equation 2.4.

$$percentage\ identity\ score = \frac{number\ of\ identical\ residues}{total\ number\ of\ residues} X100\% \quad (2.4)$$

Since the introduction of the percent identity score, numerous other measures of sequence similarity have been developed, including methods that take into account gaps and insertions in the alignment of two sequences, and methods that incorporate evolutionary models to account for changes in sequence over time. These measures have been applied to a wide range of biological problems, from the identification of new protein families and the prediction of protein structure and function to the reconstruction of evolutionary relationships and the design of new drugs. The methods and tools developed for measuring sequence similarity have been crucial for advancing our understanding of protein structure, function, and evolution. However, for the purpose of this study we only focus on the sequence similarity searching in terms of percentage identiity score using BLAST [2].

Chapter 3

Methods

This chapter discusses the design and implementation of two embedding approaches utilized in this study, along with an outline of the entire thesis

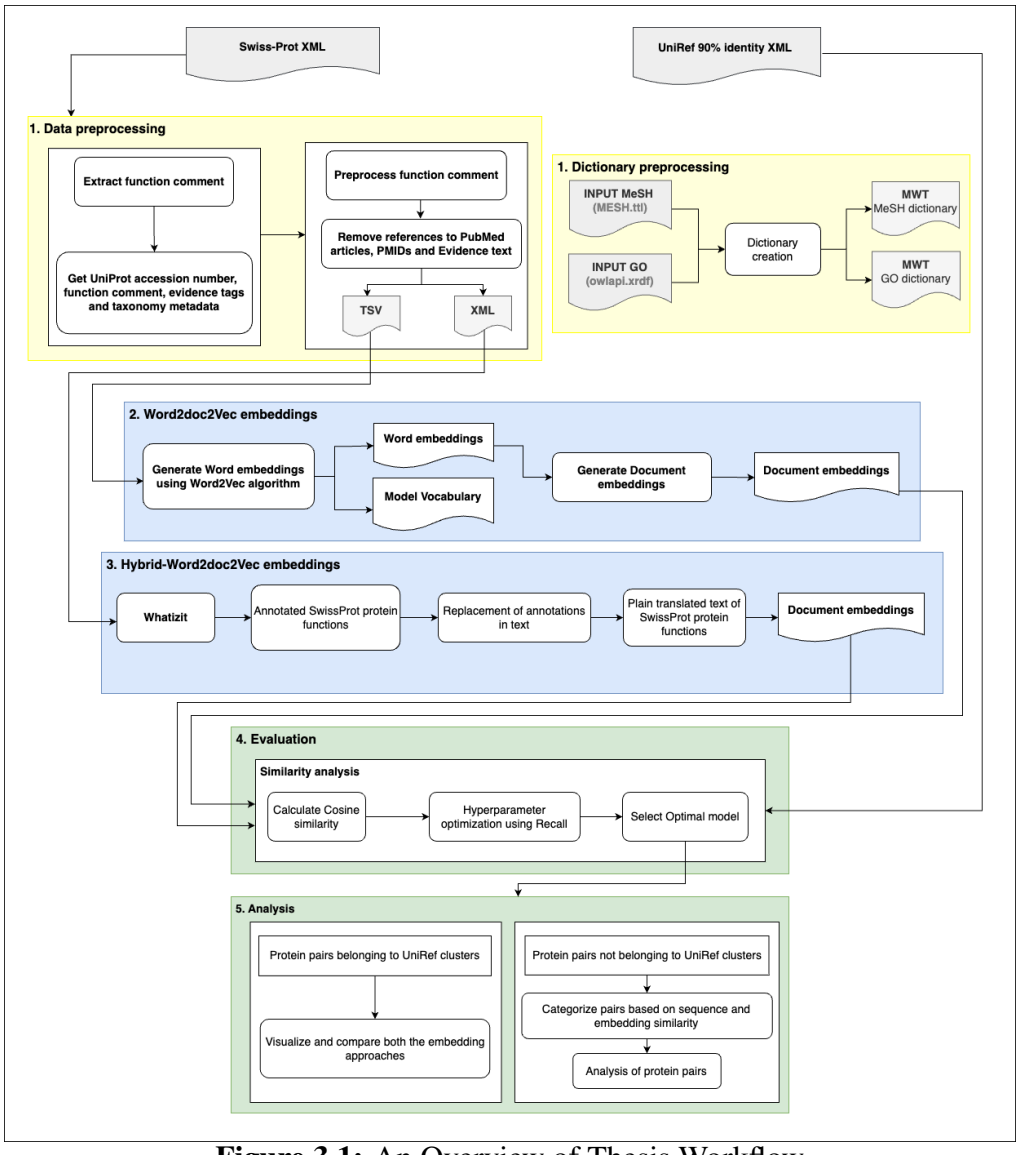


Figure 3.1: An Overview of Thesis Workflow

workflow as illustrated by Figure 3.1. Additionally, each stage of the pipeline is explained in detail.

3.1 Embedding approaches

3.1.1 Word2doc2Vec

Word2doc2Vec is a text-based embedding approach that utilizes the popular word embedding technique, Word2Vec (as explained in section 2.3.1), to generate document-level embeddings. This method first generates embeddings for individual words in a given text corpus using Word2Vec. These word embeddings capture the semantic and syntactic relationships between words based on their co-occurrence in the corpus.

Once the word embeddings are generated, the next step is to calculate document-level embeddings. This is done by calculating the centroid of the word embeddings for all the words in a given document. The centroid represents the center of mass of all the word embeddings and is used as a representation of the document. It is a relatively simple and efficient method that can be easily implemented and scaled to large datasets.

3.1.2 Hybrid-Word2doc2Vec

Hybrid-Word2doc2Vec is a text-based approach that is similar to Word2doc2Vec in that it uses Word2Vec to generate embeddings for individual words and then calculates document-level embeddings by taking the centroid of the word embeddings. However, the key difference is that the text is first annotated using two controlled vocabularies, namely the Medical Subject Headings (MeSH) and the Gene Ontology (GO) before generating the embeddings.

The process used in Hybrid-Word2doc2Vec includes a lightweight NER approach where chunks of text are recognized and associated to a concept coined in a controlled vocabulary. By incorporating MeSH and GO annotations, the Hybrid-Word2doc2Vec approach can improve the semantic meaning of the generated embeddings. Annotating words with biomedical concepts can help to normalize words that have multiple meanings in the context of biomedical research. It also provides additional context to words that are otherwise ambiguous.

Overall, the Hybrid-Word2doc2Vec approach with MeSH and GO annotations is a promising approach for text-based embeddings in biomedical research, providing a semantically-rich representation of text that can capture the nuances of biomedical concepts and facilitate downstream analysis.

3.2 Implementation

3.2.1 Materials

This section explains the datasets utilized in this study and the methods employed for retrieving them.

3.2.1.1 UniProtKB/Swiss-Prot

The XML file of Release 2022_02 of UniProtKB/Swiss-Prot was downloaded using FTP (File Transfer Protocol) download from UniProt's FTP site: `https://ftp.uniprot.org/pub/databases/uniprot/knowledgebase/complete/` to obtain the most up-to-date information available at the time of analysis. The file includes detailed information for every protein entry including but not limited to the fields of protein function, protein names and taxonomy, association to diseases & variants, protein interaction, protein structure, protein sequence & isoforms and similar proteins. For the purpose of this study, we focused specifically on the following fields, as they were deemed essential for our analysis:

accesssion : A unique identifier for each protein entry in Swiss-Prot. It is a combination of letters and numbers that allows users to quickly and easily identify individual proteins.

function : Information on the biological function of each protein. It includes a description of the protein's role in the cell, as well as any relevant biochemical or physiological processes. The text from this function comment of every protein entry is used to generate embeddings.

evidence tags : Information on the types of evidence used to support the annotation of each protein sequence. This may include experimental data, sequence similarity, or other types of evidence.

taxon : Information on the organism from which the protein sequence was derived. It includes the scientific name of the organism, as well as any relevant taxonomic information.

taxonomy lineage : A hierarchical classification of the organism from which the protein sequence was derived. It includes information on the kingdom, phylum, class, order, family, genus, and species of the organism.

taxonomic identifier : A unique identifier for the organism from which the protein sequence was derived. It is typically a numerical code that can be used to quickly and easily search for all protein sequences derived from a particular organism.

3.2.1.2 UniProt Reference Clusters - UniRef90

The XML file of UniRef90 release 2022_02 was downloaded using FTP from the UniProt's FTP site: `https://ftp.uniprot.org/pub/databases/uniprot/uniref/uniref90/`. The file provides a wide range of information on protein clusters out of which we focused on the following fields:

cluster ID : A unique identifier assigned to each protein cluster in the UniRef database. This ID is used to differentiate between different clusters and to track and analyze specific clusters of interest.

cluster name : In addition to the Cluster ID, each protein cluster in the UniRef database also has a name that reflects the protein family or group represented by the cluster. The name of the cluster is often based on the functional or structural characteristics of the proteins in the cluster.

member count : This indicate the number of protein sequences that are included in a specific cluster.

members : The individual protein sequences that are grouped together in a specific UniRef cluster. These members share a high degree of sequence identity and are grouped together to create a non-redundant representation of the UniProtKB sequences.

sequence length : The length of the protein sequence in amino acids. This domain provides important information on the size and complexity of the protein, which can impact its biological function and significance.

3.2.1.3 MeSH (Medical Subject Headings)

The Turtle (TTL) format of the 2022 MeSH RDF release was obtained from the website of the National Library of Medicine (NLM): `https://www.nlm.nih.gov/databases/download/mesh.html`. The RDF TTL file contains the complete MeSH vocabulary with information about each MeSH term, including its unique identifier, preferred label, synonyms, and relationships to other terms in the hierarchy. The hierarchy represented using broader and narrower relationships is indicative of the parent-child relationships between different MeSH terms.

3.2.1.4 Gene Ontology

The 2022 release of the Gene Ontology was downloaded from the GO Consortium's website: `http://geneontology.org/docs/download-ontology` as an XRDF file. The GO XRDF file consists of a set of OWL classes, properties, and individuals that describe the concepts and relationships within the ontology. The file contains the entire Gene Ontology with each of its GO term associated with a unique identifier, preferred label, synonyms, as well as the aspect

(Molecular Function, Cellular Component and Biological Process).

3.2.2 Dataset preprocessing

This section details the necessary preprocessing steps to extract the fields mentioned in sections 3.2.1.1 and 3.2.1.2 from their corresponding files. The emphasis is on the text coming from the function comment section of Swiss-Prot and the relevant clusters from UniRef90.

3.2.2.1 Swiss-Prot

The Swiss-Prot XML file was parsed using the xml.etree.ElementTree Python library [38] to extract the specific fields for each protein entry. Figure 3.2 illustrates the entire data preprocessing pipeline. The extracted data was then saved in a TSV file, with six columns: accession, function, evidence_tags, taxon, taxonomy_lineage, and taxon_id. Protein entries with no function comment were excluded from the TSV. The function text column was further preprocessed to eliminate any references to PubMed articles, PMIDs, and evidence tags. The removed evidence tags were added as an additional column for each protein entry. The resulting TSV file consists of seven columns and was used as the starting point for the Word2doc2Vec approach.

Additionally, the TSV file was converted into an XML file utilizing the same Python library, with each protein accession serving as an element along with its associated data. This XML file was used as the starting point for the Hybrid-Word2doc2Vec approach.

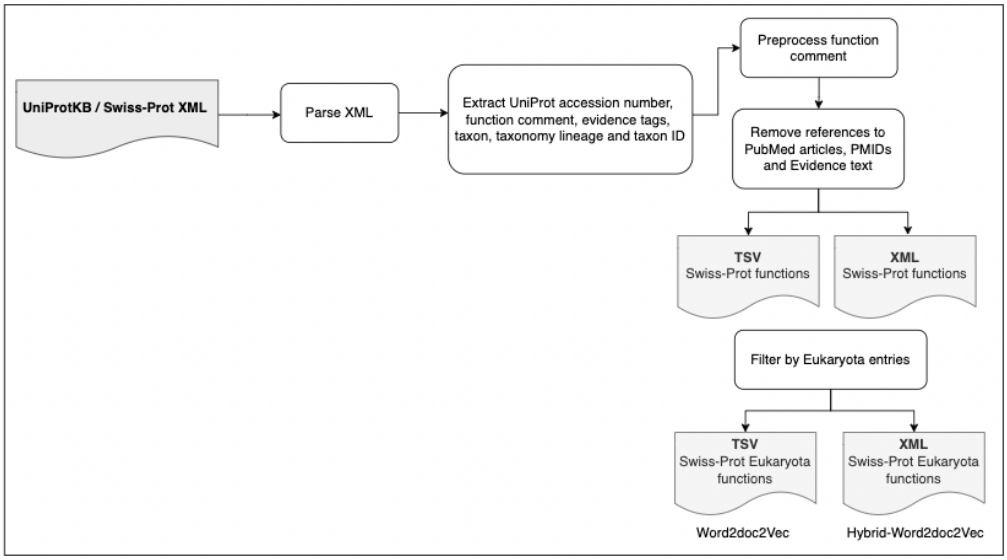


Figure 3.2: Schematic representation of the data preprocessing pipeline used for Swiss-Prot protein entries.

To optimize computational resources, we limited the dataset used for generating embeddings to a single super kingdom. We conducted an analysis to count the number of occurrences of each superkingdom, and ultimately selected Eukaryota based on the results of this analysis as presented in Table 5.2.

3.2.2.2 UniRef90

A set of parameters was established to serve as the criteria for identifying clusters of interest from the UniRef 90% identity XML file. The clusters having Eukaryota as the common taxon were the main focus, based on the primary dataset used for embeddings as described in section 3.2.2.1. Three main criteria were defined: (i) clusters should have Eukaryota as the common taxon, (ii) the number of members in each cluster should be at least two, and (iii) each cluster should have at least one member that is a Swiss-Prot entry. These criteria were used to generate a TSV file that acted as a filter to extract our clusters of interest. This was done in order to reduce computation time and memory usage, as the UniRef90 XML file has a large file size of 298 GB.

The filtering process consisted of two main steps. In the first step, a RESTful API request was made to UniProt's API to extract UniRef Eukaryota clusters belonging to 90% identity with a cluster size greater than or equal to 2 using the streaming endpoint. The resulting response was stored in an intermediate TSV file, which was then used to further filter the clusters using criterion (iii), which required each cluster to have at least one Swiss-Prot member. The resulting clusters were stored in the final TSV file, which consisted of four columns: cluster_id, cluster_name, types, and cluster_size.

To parse the UniRef 90 XML, xml.etree.ElementTree library [38] mentioned earlier was used. While iterating through each element (UniRef cluster) of the XML, it was checked if the element was present in the generated TSV file. If it was present, the corresponding members for that element were extracted. These clusters were then written to a TSV file that contained four columns: cluster_id, cluster_name, members, and cluster_size. Figure 3.3 depicts the entire preprocessing pipeline to parse and extract the clusters of interest from Uniref90 XML file.

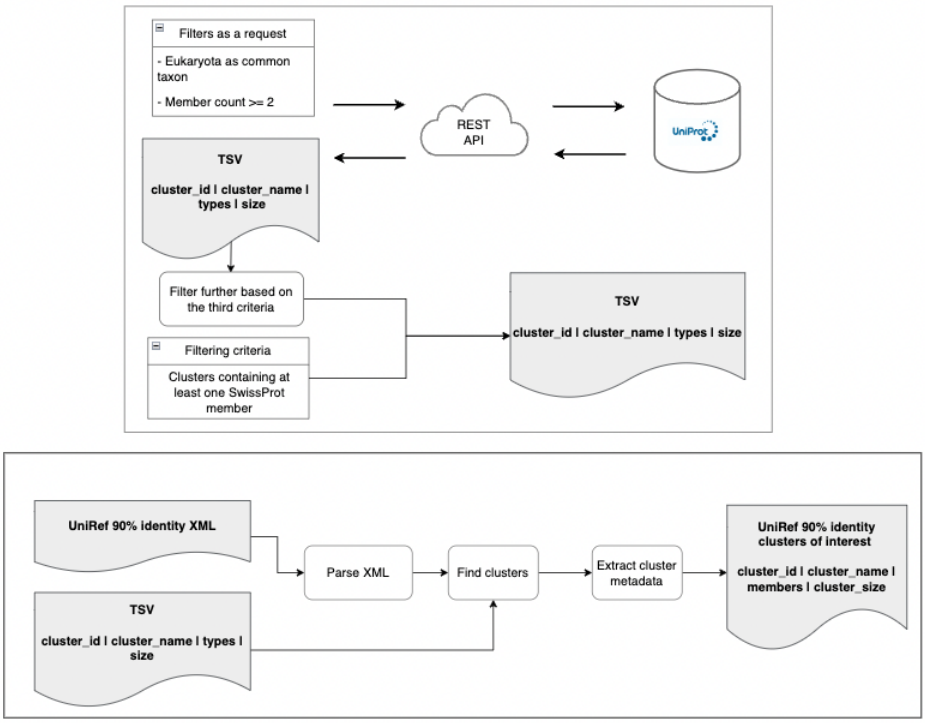


Figure 3.3: Schematic representation of the cluster preprocessing pipeline for UniRef90 clusters

3.2.3 Dictionary preprocessing

This section outlines the process of creating dictionaries essential for annotating function text using the Whatizit tool and presents an overview of the process in Figure 3.4. These dictionaries serve the purpose of identifying expressions in the text and linking them to a concept in a controlled vocabulary. Two ontologies, Medical Subject Headings (MeSH) and Gene Ontology (GO), were utilized to generate these dictionaries. To create the MeSH dictionary, the MeSH TTL file was parsed using the RDFLib Python library [39]. The required information such as class ID, preferred label, synonyms, concept unique identifier (CUI), and semantic types were extracted for each term, while also checking for obsoletion. The preprocessing pipeline for MeSH involved removing all obsolete terms and filtering out terms with class IDs that included semantic types. Similarly, the GO XRDF file was parsed using the same library, and the preprocessing pipeline involved eliminating all deprecated terms, extracting class ID, preferred label, synonyms, and aspect (Molecular function, Biological process, or Cellular component) for the remaining terms. The terms obtained from both ontologies were used to create the corresponding MWT dictionaries.

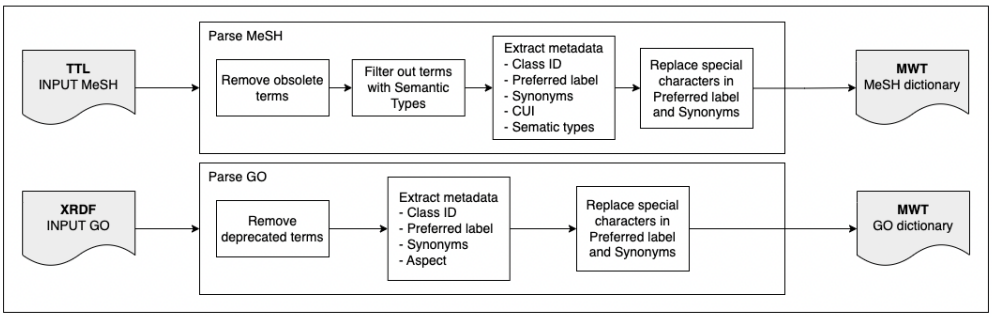


Figure 3.4: Schematic representation of dictionary preprocessing for MeSH and GO for the annotation of function text using Whatizit

The structure of the MWT dictionary is based on XML, with the first part consisting of three lines that define the XML prolog. The first line declares the version of XML used (1.0) and the character encoding used in the document (UTF-8). The second line specifies the XML namespace, which is based on the code repository. The third line contains the annotation template and includes a 'z:Ontology' element for each ontology term, along with its corresponding attributes. For MeSH, the template includes a 'z:MESH' element with three attributes (id, cui, semantics) and the term to be annotated. These attributes specify the MeSH ID, CUI, and semantics of the term. For the GO MWT dictionary, the template includes a 'z:GO' element with two attributes (id and aspect) and the term to be annotated. These attributes specify the GO ID and aspect of the term.

```
<?xml version='1.0' encoding='UTF-8'?>
<mwt xmlns:z="https://github.com/zbmed-semtec/protein-function-embeddings-thesis#">
<template><z:MESH id='%1' cui='%2' semantics='%3'>%0</z:MESH></template>

<t p1="http://purl.bioontology.org/ontology/MESH/C000659400" p2="C4202352"
    p3="http://purl.bioontology.org/ontology/STY/T004">Acaulospora ignota</t>
<t p1="http://purl.bioontology.org/ontology/MESH/C000684122" p2="C3424521"
    p3="http://purl.bioontology.org/ontology/STY/T004">Polypaecilum pisci</t>
<t p1="http://purl.bioontology.org/ontology/MESH/C000684122" p2="C3424521"
    p3="http://purl.bioontology.org/ontology/STY/T004">Aspergillus pisci</t>

</mwt>
```

Listing 3.1: MeSH Ontology MWT dictionary excerpt

The second part of the MWT dictionary shows how the annotation template is used to annotate a specific term. The 't' element contains the text to be annotated, which is the Preferred Label (e.g., "Acaulospora ignota" or "Polpaecilum

pisci"), along with its three attributes specified in the template (p1 for Mesh ID, p2 for CUI, and p3 for the semantics of the MeSH term). If a term has synonyms (e.g., "Polpaecilum pisci" and "Aspergillus pisci"), each synonym is added as a separate ‘t’ element but is given the same class ID, CUI, and semantics. An excerpt from the MeSH MWT dictionary is shown in Listing 3.1, and it demonstrates how this works.

The GO MWT dictionary follows a similar pattern, with the annotation template containing a "z:GO" element with two attributes (id and aspect) for each ontology term and the element content as the term to be annotated. All synonyms are also added as separate ‘t’ elements. A portion of the GO MWT dictionary is illustrated in Listing 3.2.

```
<?xml version='1.0' encoding='UTF-8'?>
<mwt xmlns:z="https://github.com/zbmed-semtec/protein-function-embeddings-thesis#">
<template><z:GO id='%1' aspect='%2'>%0</z:GO></template>

<t p1="http://purl.obolibrary.org/obo/GO_0000001"
    p2="biological_process">mitochondrion inheritance</t>
<t p1="http://purl.obolibrary.org/obo/GO_0000001"
    p2="biological_process">mitochondrial inheritance</t>
<t p1="http://purl.obolibrary.org/obo/GO_0000002"
    p2="biological_process">mitochondrial genome maintenance</t>
<t p1="http://purl.obolibrary.org/obo/GO_0000003"
    p2="biological_process">reproduction</t>

</mwt>
```

Listing 3.2: Gene Ontology MWT dictionary excerpt

There are some special characters that can cause the MWT format to become invalid if they are present in the text to be annotated (i.e., Preferred Label and Synonyms). To prevent this from happening, these characters are replaced with their corresponding HTML encoding. The following conversions are used:

```
& is converted to &amp;
" is converted to &quot;
' is converted to &apos;
< is converted to &lt;
> is converted to &gt;
```

This ensures that the MWT format remains valid even when these special characters are present in the text to be annotated.

3.2.4 Annotation

This section describes the process of annotating function text using MeSH and GO vocabulary. The annotation is done using the minimal dockerized version of Whatizit, as explained in section 2.5.2. The annotation process involves five phases: setting up the Docker container, formatting the input file, annotating the function text, formatting the output file, and translating the function text.

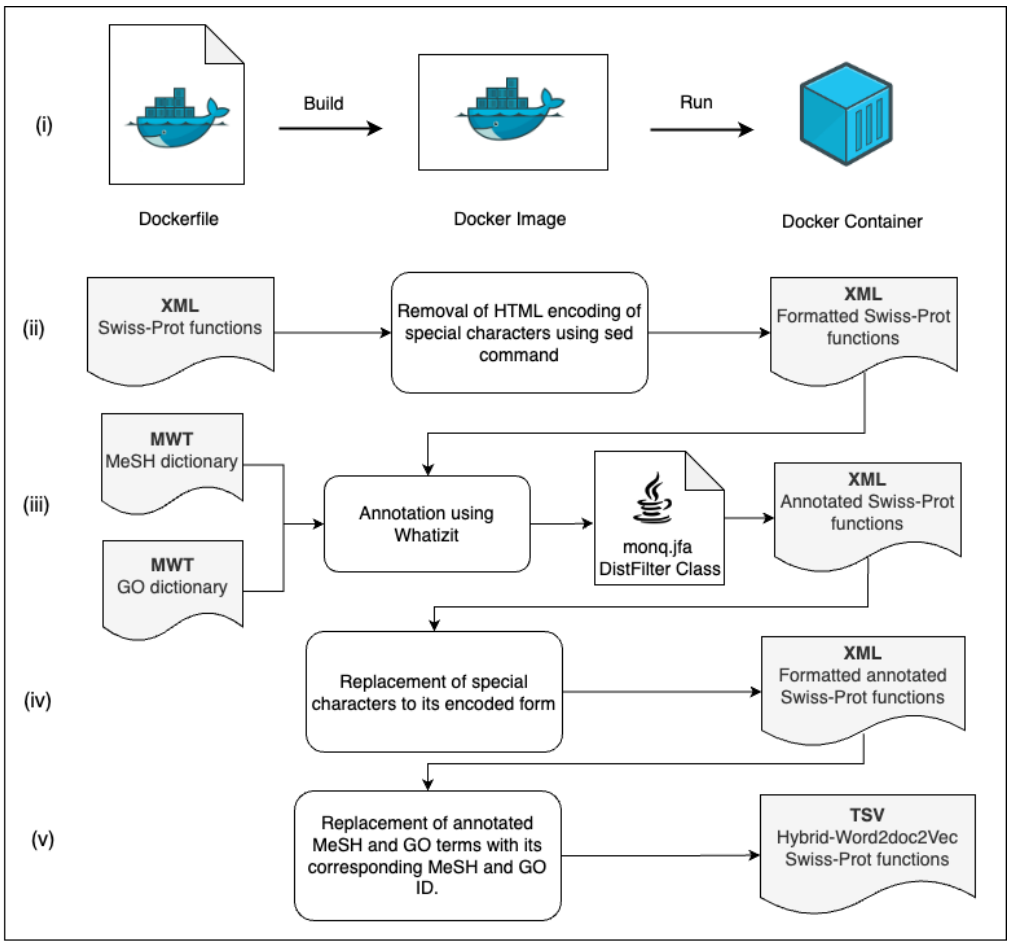


Figure 3.5: Overview of Annotation process

(i) Setting up the Docker container: This involves creating a Docker image for the minimal version of Whatizit, running the image, and executing the container to create an annotation server. The process is shown in Listing 3.3.

```
sudo docker build -t simple_whatizit .
sudo docker run -d simple_whatizit
sudo docker exec -it <container_ID> /bin/bash
cd $MONQ
```

Listing 3.3: Building and Running Docker Image of Minimal Whatizit

(ii) Formatting the input file: The second phase is formatting the input XML file that contains the function text. This is done by replacing the encoded version of special characters with their non-encoded version. The reason for this is that Whatizit cannot detect special characters in their encoded form. To accomplish this, the sed linux command is used to replace the encoded version with the plain version. The sed command used is as follows:

```
sed 's/&lt;/</g;s/&gt;/>/g;s/&amp;/\&/g' {input_path} > {formatted_input_path}
```

(iii) Annotating the function text: In this phase, the formatted input XML file is annotated with the dictionary of choice in the server file using DistFilter of MONQ. The following command is used for this purpose:

```
cat {formatted_input_path}  DistFilter svr=xmlElem > {output_path}
```

(iv) Formatting the output file: The resulting annotated XML file is then formatted by replacing the non-encoded special characters with their encoded versions to create a valid XML file. The BeautifulSoup [40] Python library was used for this purpose.

The function text was annotated first using the MeSH MWT dictionary, and then with the GO MWT dictionary, following Phases 2, 3, and 4 each time.

(v) Translating the function text: This phase involves the translation of the annotated terms in the function text to plain text. In order to do so, all the annotated terms were replaced by their corresponding MeSH ID and GO ID. The resulting function text was saved as TSV file having two columns: accession and function text. This was used for the downstream Hybrid-Word2doc2Vec approach.

3.2.5 Text preprocessing

The preprocessing of the function texts for both approaches using their corresponding TSV files included converting the text into lower case, removing punctuations except for hyphens and numbers, and tokenization. The lower method of string was used to convert the text into lower case, while NLTK library [41] was employed for word tokenization. To remove all the special characters from the text, regular expressions [42] were used with the exception of hyphens and numbers. Figure 3.6 depicts the text preprocessing pipeline.

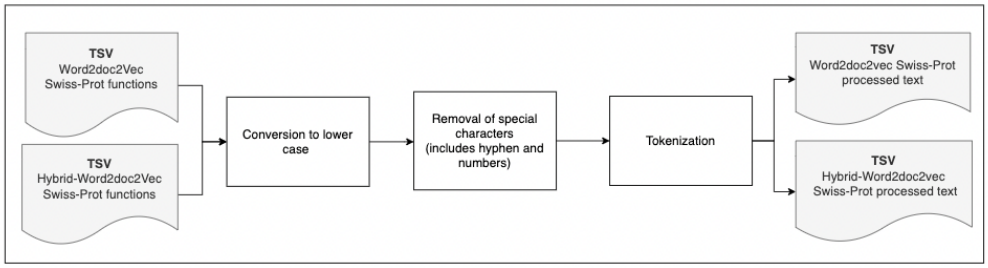


Figure 3.6: Overview of text preprocessing pipeline for function texts of Swiss-Prot entries

3.2.6 Generation of Embeddings

This section discusses the creation of Word2Vec models using the Gensim library [43], which were trained on the complete Swiss-Prot function text corpus and the generation of embeddings for the Swiss-Prot Eukaryota function texts. Additionally, the hyperparameters used in this process are outlined. An overview of the embedding generation process is depicted in Figure 3.7.

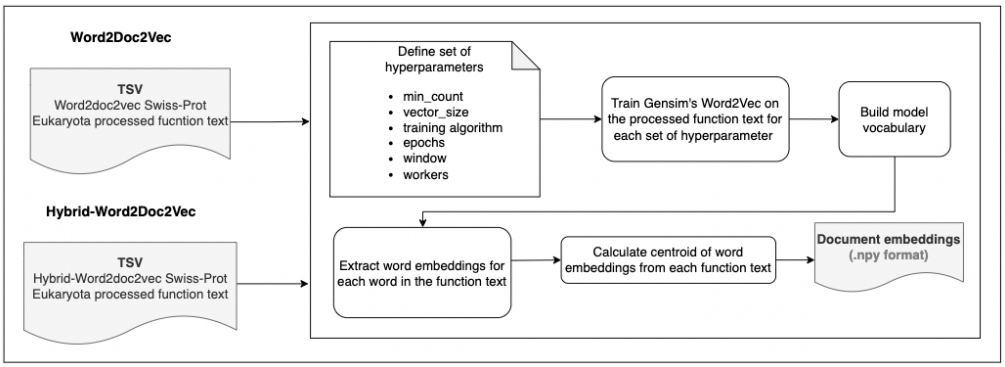


Figure 3.7: Overview of the process of generating embeddings from function texts

Hyperparameters

Table 3.1 provides a summary of the hyperparameter combinations used to generate the embeddings using both approaches. These parameters include min_count, vector_size, epochs, window, and workers, as well as the training

algorithm, which is either Continuous Bag of Words (cbow) or Skip-gram (sg). The training algorithm specifies the architecture of Word2Vec, as explained in sections 2.3.1.1 and 2.3.1.2. Min_count acts as a threshold value for excluding all words with a total frequency below this value. Vector_size determines the dimensionality of the word vectors to be generated. Epochs denote the number of iterations over the corpus. Window refers to the maximum distance between the current and predicted word within a sentence, and workers indicates the number of worker threads used to train the model for faster training on multicore machines.

The parameters of epochs, window, and workers were held constant at values of 5, 5, and 4, respectively, while varying the training algorithm parameter based on the approach (cbow: 0 or sg: 1). In addition, two different values for the vector_size (200 and 400) and three different values for the min_count (1, 2, and 3) were used.

Table 3.1: Hyperparameter combinations for Word2Vec models

Model	Vector size	Minimum count	Algorithm	Epochs	Window	Workers
1	200	1	cbow	5	5	4
2	200	2	cbow	5	5	4
3	200	3	cbow	5	5	4
4	200	1	sg	5	5	4
5	200	2	sg	5	5	4
6	200	3	sg	5	5	4
7	400	1	cbow	5	5	4
8	400	2	cbow	5	5	4
9	400	3	cbow	5	5	4
10	400	1	sg	5	5	4
11	400	2	sg	5	5	4
12	400	3	sg	5	5	4

Section 4.4 explains the Hyperparameter optimization pipeline. The optimization was performed using a Reduced Eukaryota dataset and the evaluation metric used was Recall.

Generation of word embeddings

To generate and train the Word2Vec model, the preprocessed text of both approaches are fed as input in the form of a TSV file. Along with hyperparameters, preprocessed function texts as an array are passed as parameters to the Word2Vec module. We used the model.build_vocab method of Gensim in order to build the Vocabulary object and save the model files. The output of this pro-

cess are three resulting files: word2vec.model, word2vec.model.syn1neg.npy and word2vec.wv.vectors.npy.

word2vec.model: This file contains the trained Word2Vec model itself. It is typically saved as a binary file using Python's pickle module, which allows the model to be loaded and used in future applications. The model contains information such as the vocabulary, the word vectors, and the parameters used during training.

word2vec.model.syn1neg.npy: This file contains the negative-sampling weights of the Word2Vec model. Negative sampling is a technique used during training to improve the efficiency and accuracy of the model. These weights are used to calculate the probability of each negative sample being selected during training.

word2vec.wv.vectors.npy: This file contains the word vectors of the Word2Vec model. Word vectors are distributed representations of words that capture their semantic and syntactic properties. Each row of this matrix corresponds to a word in the vocabulary, and each column corresponds to a dimension of the vector space. The values in each cell represent the weight of that word in that dimension.

Generation of document embeddings

To generate document embeddings from word embeddings, we employ the centroid approach as explained in section 2.4. To achieve this, we load the Word2Vec model using the load() method of the Word2Vec class. This method reads the pre-trained Word2Vec model from the word2vec.model file. Then, we extract the word embeddings for each Eukaryota accession from its function text by using model.wv[word] and store these values. Next, we compute the centroid of the word embeddings to obtain the average position of all word embeddings in the vector space. The centroid method assumes that the word embeddings follow a normal distribution around the centroid and that the distribution is isotropic, meaning that the variance is the same in all dimensions. Finally, the resulting document embeddings for each accession are stored in individual .npy format files.

Formatting embeddings

To leverage the performance benefits of Numba [44], we convert the embeddings from their default 64-bit floating point format to 32-bit. This not only improves the computational efficiency but also reduces the memory usage during similarity calculations in the later stages of the pipeline. Moreover, to facilitate faster loading and manipulation of the embeddings in the evaluation pipeline, we save them as a Pandas DataFrame in a pickle file using the 'to_pickle' function.

Chapter 4

Evaluation

4.1 Evaluation Aim and Setup

This section provides an in-depth explanation of the experiments conducted and the metrics utilized to assess the effectiveness of the proposed embedding techniques. Specifically, the evaluation aims to compare the behavior of the

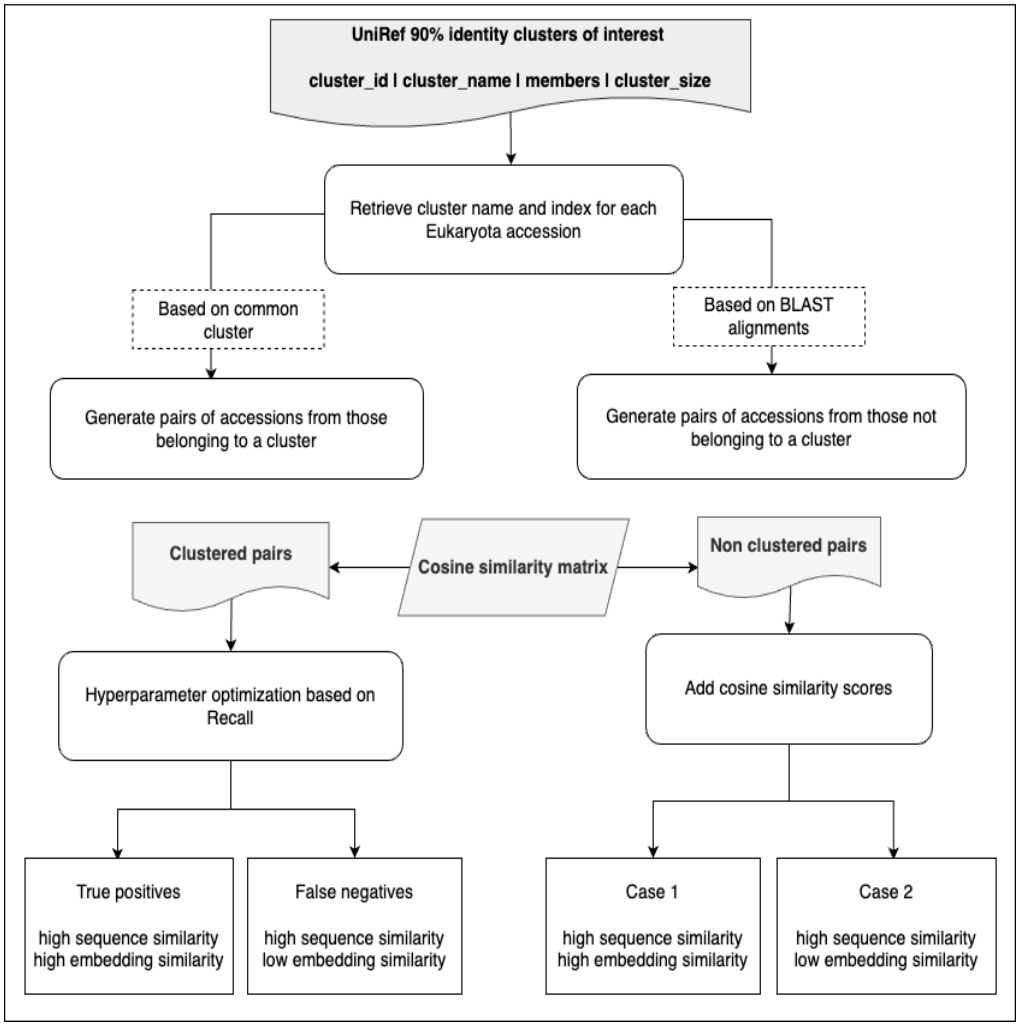


Figure 4.1: Overview of Evaluation Setup

embeddings with respect to sequence similarity. To accomplish this, a two-phase evaluation process is established. The first phase (as explained in sections 4.2, 4.3 and 4.4) involves generating a set of ground truth data, which is used to evaluate the different hyperparameters for the embeddings in relation to pairs of proteins found within the sequence clusters derived from the UniRef 90% clusters of interest. The second phase (as explained in sections 4.5 and 4.6) examines pairs of proteins that are not part of the UniRef 90% clusters of interest. This involves utilizing BLAST to create these pairs and analyzing the trends observed using embeddings. Figure 4.1 gives an overview of the Evaluation setup.

4.2 Retrieving clusters for Eukaryota accessions

To establish the ground truth data according to our proposed hypothesis, we first retrieve the UniRef clusters for all Eukaryota accessions. This is achieved by using the UniRef 90% identity clusters TSV file that contains the clusters of interest, as well as all the Eukaryota accessions. By retrieving the corresponding cluster index value for each accession, we obtain a numeric value between 0 and 137,047, based on the total number of Uniref90% clusters of interest specified in Table 5.3. A cluster index value of 0 indicates that the accession does not belong to any cluster. The resulting cluster index values for each accession are then stored as a TSV file. Figure 4.2 depicts the process of generating the aforementioned TSV file and demonstrates how the values are stored in its two columns.

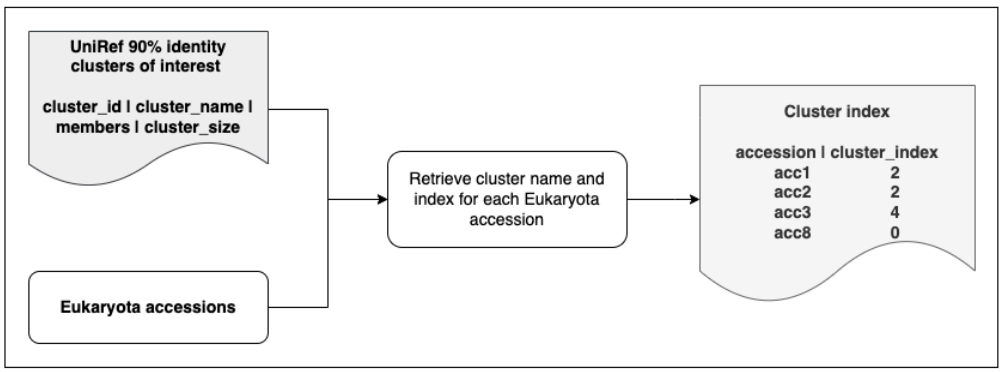


Figure 4.2: Schematic representation of the process of retrieving Cluster index for Swiss-Prot Eukaryota protein entries

The statistics obtained from the resulting cluster index file are presented in Table 4.1. Out of a total of 161,354 Swiss-Prot Eukaryota accessions as per Table 5.2, approximately 137,789 belong to a UniRef 90% identity cluster of interest,

while the remaining 23,565 do not belong to any Uniref 90% identity cluster of interest.

Table 4.1: Distribution of Eukaryota Entries Belonging to UniRef90% Clusters of Interest

Dataset	Statistics
Total Eukaryota accessions	161,354
Eukaryota accessions belonging to a cluster	137,789
Eukaryota accessions not belonging to a cluster	23,565

4.3 Generating Clustered pairs

To generate clustered pairs, pairs of accessions are created from those accessions that belong to the same cluster of interest. The process involves filtering out all the accessions that exclusively belong to a cluster, while retaining a threshold of 2 accessions for each cluster. In order to compute all the possible combinations of accessions that belong to the same cluster, the combinations iterator from the itertools Python library [45] was utilized. Figure 4.3 provides a schematic representation of this process, while Table 4.2 displays the number of accession pairs that were generated.

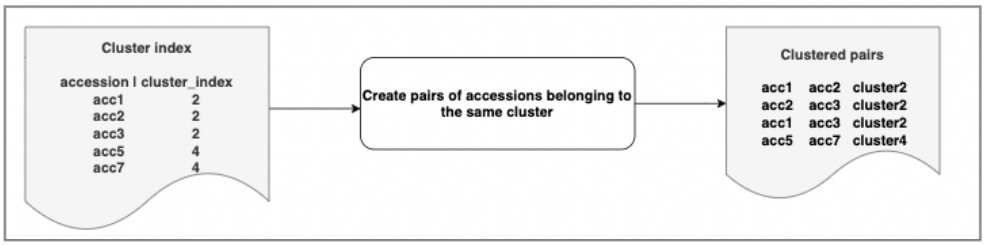


Figure 4.3: Schematic representation of the process of generating Clustered pairs

Table 4.2: Count of Eukaryota protein pairs belonging to a cluster

Dataset	Statistics
Eukaryota protein pairs belonging to a cluster	156,772

4.4 Hyperparameter optimization

The following section describes the pipeline for Hyperparameter optimization, which aims to assess the performance of all embedding models. This pipeline can be broken down into three primary stages:

Reduction of Dataset

To minimize the amount of computation time and memory usage, a subset of Eukaryota entries was extracted by utilizing the fraction method of Pandas Dataframe, which splits the data. A subset containing 32,271 entries was created by selecting a 20% fraction of the original dataset.

Calculation of Cosine similarity

To evaluate the similarity between embeddings, cosine similarity is utilized. A cosine similarity matrix with dimensions 32,271 x 161,354 is initialized using NumPy [46] array. The matrix is populated by iterating through each cell, extracting the corresponding embeddings based on the index of the cell (i, j), and calculating their cosine similarity. Only cosine similarity values greater than or equal to 0.90 are stored. The process of computing cosine similarity and populating the matrix is optimized using the @njit(fastmath=True) decorator from the Numba library [44]. The entire cosine matrix is saved in .npz format, a compressed binary format used by Python's Numpy library to store matrices.

Recall

To optimize the hyperparameters, we first utilize the cosine similarity matrix by extracting the corresponding cosine similarity score for each pair of accessions from the clustered pairs TSV file. These scores are then added as an additional column to the TSV file (accession1 | accession2 | cluster_index | cosine_similarity), and this process is repeated for all 24 sets of hyperparameters.

Recall is a performance metric used to evaluate the effectiveness of a machine learning model in identifying all relevant instances of a specific class from a given dataset. It is calculated as the proportion of true positive predictions made by the model over the total number of actual positive instances in the dataset. To compute Recall, we count the number of true positives (TP) and false negatives (FN) for all hyperparameter combinations. The Recall formula is given by Equation 4.1.

$$Recall = \frac{True\,Positives}{True\,Positives + False\,Negatives} \tag{4.1}$$

where,

True Positives represent the number of accession pairs having high sequence (greater than or equal to 90% sequence identity) and high embedding similarity (greater than or equal to 0.9 cosine similarity).

False Negatives represent the number of accession pairs having high sequence similarity (greater than or equal to 90% sequence identity) but low embedding similarity (less than 0.9 cosine similarity).

4.5 BLAST

BLAST (Basic Local Alignment Search Tool) [2] a widely used bioinformatics tool for sequence alignment and comparison. In this evaluation, BLAST was used for two main purposes. Firstly, to retrieve the exact percentage identity scores of proteins belonging to clustered pairs as explained in section 4.2, which are known to exhibit a sequence similarity of 90% or higher. Secondly, to generate pairs of proteins that do not belong to any cluster by using the BLAST results and corresponding percentage identity scores.

To handle the large amount of data involved in this process, ElasticBLAST version 1.0.0 [47, 48] was utilized. ElasticBLAST is a cloud-based service provided by the National Center for Biotechnology Information (NCBI) [49] that allows for high-throughput BLAST searches on cloud platforms such as Amazon Web Services (AWS) or Google Cloud Platform (GCP).

In this study, the Google Cloud Platform was used to perform ElasticBLAST. The Swiss-Prot FASTA file from the release of 2022_02 was used as the input for the BLAST search. All sequences corresponding to the Eukaryota entries were extracted using the Biopython [50] SeqIO library and divided into batches of 10,000 sequences each, totaling 17 batches. These batches were then used as input for the ElasticBLAST search.

The configuration file in Listing 4.1 was utilized to set up ElasticBLAST. The BLASTP program was chosen and a value of 0.01 was specified as the e-value. Furthermore, the search was constrained solely to the Eukaryota taxon by providing its taxon ID. The results were obtained in batches, which were subsequently merged, parsed, and transformed into a TSV file containing the accession pairs and its corresponding sequence percentage identity score.

```
[cloud-provider]
gcp-region -us-east4
gcp-zone-us-east4-b

[cluster]
num-nodes = 1
labels = owner=USER

[blast]
program blastp
db = swissprot
queries=gs://elasticblastp/queries/swissprot. fasta
results = gs://elasticblastp/results/
options = -task blastp -evalue 0.01 -outfmt -taxids 2759 "7 std sskingdoms ssciname"
```

Listing 4.1: ElasticBLAST Configuration file

4.6 Generating Non-Clustered pairs

To investigate proteins that are not part of the UniRef 90% clusters of interest, we created Non-Clustered pairs. To do this, we used BLAST alignments for each protein and retrieved their sequence percentage identity score from the BLAST results TSV file. The corresponding cosine similarity values was also stored in the TSV file. This process was carried out using the top two models selected for each approach based on section 4.3, resulting in two TSV files. Each file contains pairs of accessions, their sequence percentage identity, and cosine similarity score.

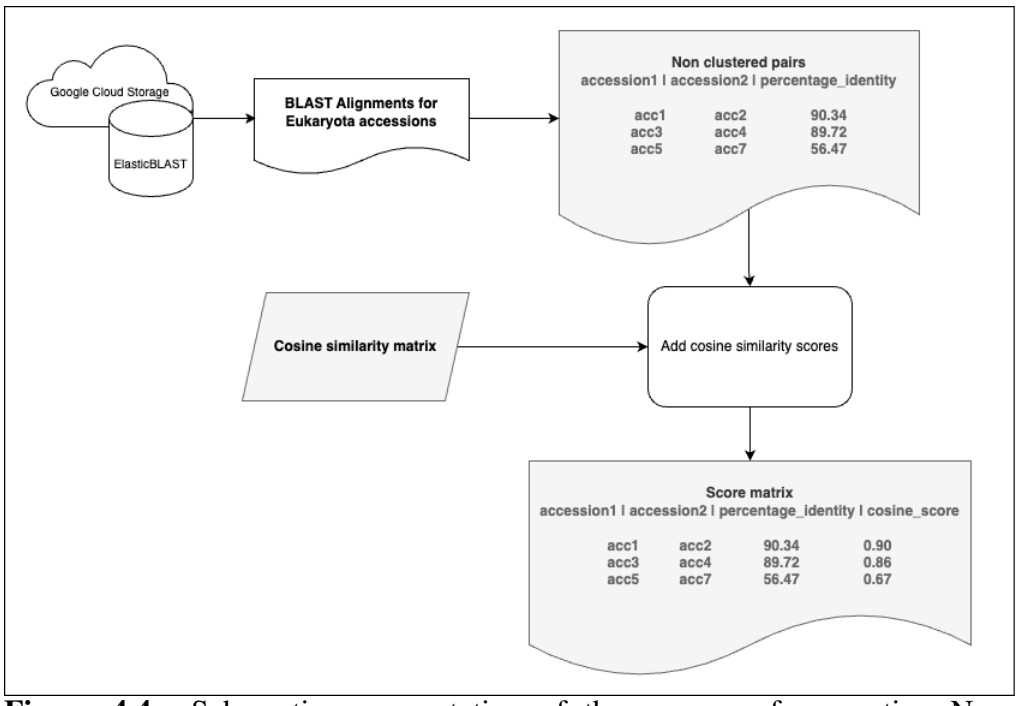


Figure 4.4: Schematic representation of the process of generating Non-Clustered pairs

A schematic representation of the process is illustrated in Figure 4.4 and the count is shown in Table 4.3 wherein these files provide a useful resource for further analysis of the Non-Clustered proteins.

Table 4.3: Count of Eukaryota protein pairs not belonging to a cluster

Dataset	Statistics
Eukaryota accession pairs not belonging to a cluster	115,019

Incorporating cosine similarity scores for both clustered and non-clustered pairs enabled us to evaluate the similarity scores and obtain a better understanding of the differences between these two sets of proteins.

Chapter 5

Results

5.1 Data Analysis and Statistical Findings

The Swiss-Prot XML file of release 2022_02 has a total of 567,483 protein entries, with only 460,009 having a written function comment. Table 5.1 provides the statistics for this. The dataset was further reduced by counting the super kingdom for those with a function comment, with the counts presented in Table 5.2.

Table 5.1: Statistics of Swiss-Prot protein Entries

Dataset	Statistics
Swiss-Prot proteins	567,483
Swiss-Prot proteins with function comments	460,009

Throughout this study, the main focus of interest was on the Eukaryota subset of proteins. While the Word2Vec model was trained on the entire Swiss-Prot corpus containing function comments, embeddings were only generated for the Eukaryota proteins.

Table 5.2: Statistics of Swiss-Prot protein entries based on Super Kingdom

Dataset	Statistics
Eukaryota	161,354
Bacteria	272,438
Archaea	14,066
Viruses	12,151

The UniRef90 % identity XML file of release 2022_02, contained a total number of 147,407,377 clusters. From these clusters, we retrieved 137,047 clusters of interest using the preprocessing steps outlined in section 3.2.2.2. These clusters served as the ground truth to compare Word2Vec model hyperparameters and establish the best hyperparameter. Table 5.3 presents these statistics.

Table 5.3: Statistics of clusters retrieved from UniRef90 % Identity XML File

Dataset	Statistics
UniRef 90% clusters	47,407,377
UniRef 90% clusters of interest	137,047

5.2 Model Vocabulary

Table 5.4 presents the vocabulary sizes obtained from Word2Vec models that were trained on Swiss-Prot protein function comments using varying minimum count (min_count) parameters. The vocabulary size indicates the number of unique words for which the Word2Vec model learned word embeddings.

It can be observed that the Hybrid-Word2doc2Vec approach has a larger vocabulary size compared to the Word2doc2Vec approach with the same min_count parameter.

Table 5.4: Vocabulary Sizes of the trained Word2Vec Models

Approach	Minimum count	Vocabulary size
Word2doc2Vec	1	96,414
Word2doc2Vec	2	71,015
Word2doc2Vec	3	57,851
Hybrid-Word2doc2Vec	1	101,120
Hybrid-Word2doc2Vec	2	74,971
Hybrid-Word2doc2Vec	3	61,186

5.3 Hyperparameter Optimization

In order to determine the best model for capturing the essence of function comments using embeddings, Recall was utilized as explained in section 4.4. Tables 5.5 and 5.6 present the Recall scores, which are relatively close to each other. It should be noted that the scores are overall low because only a subset of Eukaryota entries, specifically 20%, was used for the process. Overall, it was observed that the skip-gram algorithm outperformed the cbow algorithm slightly.

Based on the Recall scores for both the approaches, we selected Model 4 as the best hyperparameter combination for the embeddings. This model has hyperparameters of vector size, minimum count, and algorithm set to 200, 1, and skip-gram, respectively, with Recall scores of 0.386229 for Word2doc2Vec and 0.385406 for Hybrid-Word2doc2Vec.

Table 5.5: Recall scores for Word2doc2Vec

Model	Vector size	Minimum count	Algorithm	Epochs	Window	Workers	Recall
1	200	1	cbow	5	5	4	0.378332
2	200	2	cbow	5	5	4	0.378492
3	200	3	cbow	5	5	4	0.380903
4	200	1	sg	5	5	4	**0.386229**
5	200	2	sg	5	5	4	0.385655
6	200	3	sg	5	5	4	0.383920
7	400	1	cbow	5	5	4	0.378403
8	400	2	cbow	5	5	4	0.378549
9	400	3	cbow	5	5	4	0.378575
10	400	1	sg	5	5	4	0.385036
11	400	2	sg	5	5	4	0.378626
12	400	3	sg	5	5	4	0.384609

Table 5.6: Recall scores for Hybrid-Word2doc2Vec

Model	Vector size	Minimum count	Algorithm	Epochs	Window	Workers	Recall
1	200	1	cbow	5	5	4	0.378326
2	200	2	cbow	5	5	4	0.378426
3	200	3	cbow	5	5	4	0.378179
4	200	1	sg	5	5	4	**0.385406**
5	200	2	sg	5	5	4	0.237714
6	200	3	sg	5	5	4	0.384577
7	400	1	cbow	5	5	4	0.378441
8	400	2	cbow	5	5	4	0.378485
9	400	3	cbow	5	5	4	0.378288
10	400	1	sg	5	5	4	0.384303
11	400	2	sg	5	5	4	0.177289
12	400	3	sg	5	5	4	0.383671

5.4 Similarity analysis

5.4.1 Clustered pairs

The optimal models selected for both approaches were used to generate the cosine similarity matrix for the entire SwissProt Eukaryota corpus. The resulting matrix had dimensions of 161,354 x 161,354. Subsequently, Recall was calculated for these optimal models using the clustered pairs in a similar way as explained in section 4.3. Table 5.7 presents the corresponding statistics.

Table 5.7: Recall scores for Clustered pairs based on optimal models

Approach	True positives	False nagatives	Recall
Word2doc2Vec	152,754	4,018	0.9743
Hybrid-Word2doc2Vec	152,353	4,419	0.9718

Figures 5.1 and 5.2 present scatter plots that demonstrate the relationship between sequence and embedding similarity based out of the clustered set of proteins. Each point on the plot represents a pair of proteins. The x-axis displays the cosine similarity value calculated between the embeddings of a pair of proteins, while the y-axis represents the sequence similarity, calculated using the percentage identity score obtained through BLAST analysis, as outlined in section 4.5.

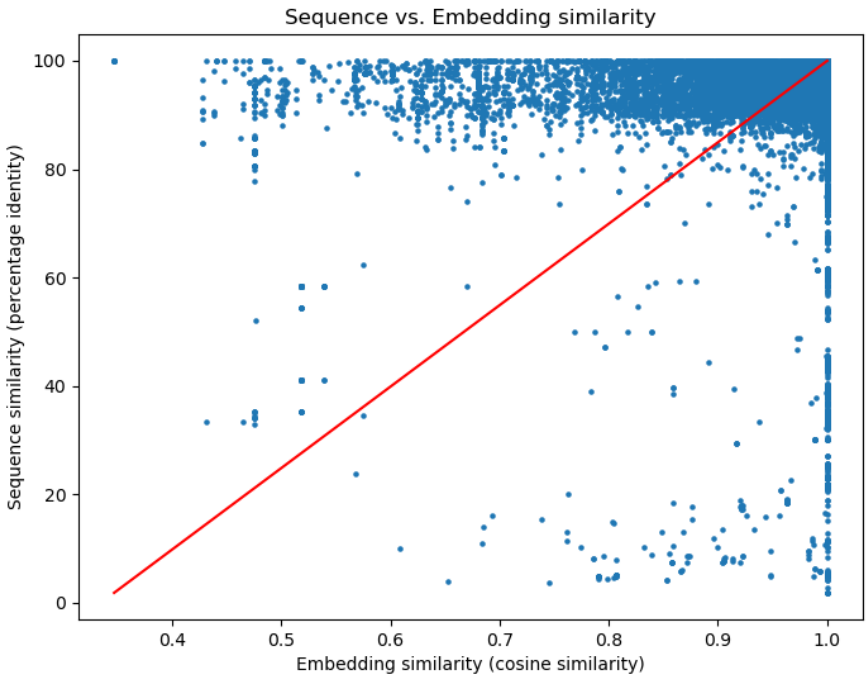


Figure 5.1: Scatter plot representing Sequence vs. Embedding similarity of Swiss-Prot Eukaryota protein pairs using Word2doc2Vec approach

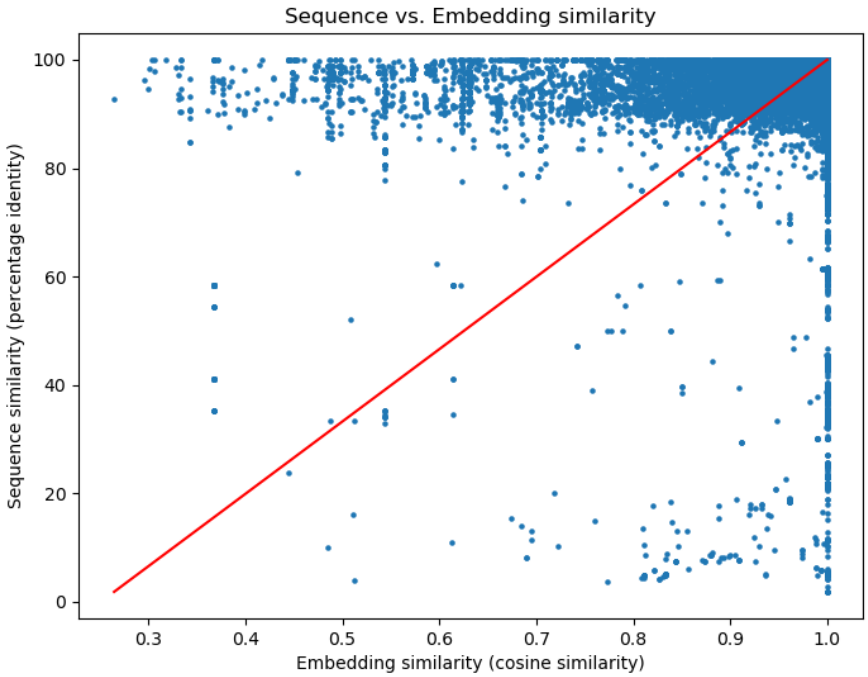


Figure 5.2: Scatter plot representing Sequence vs. Embedding similarity of Swiss-Prot Eukaryota protein pairs using Hybrid-Word2doc2Vec approach

5.4.2 Non-Clustered pairs

Based on the generated Non-Clustered pairs in section 4.5 we subsequently divided them into two categories: Case 1 and Case 2. Case 1 pertains to protein pairs with a high degree of sequence similarity (at least 90% sequence identity) and a high degree of embedding similarity (at least 0.90 cosine similarity). Conversely, Case 2 corresponds to protein pairs with a high degree of sequence similarity (at least 90% sequence identity) but a low degree of embedding similarity (less than 0.90 cosine similarity).

Table 5.8 displays the counts for the two cases outlined above. Our analysis reveals that a substantial number of protein pairs belong to Case 1, which demonstrates a high level of both sequence and embedding similarity.

Table 5.8: Count of Non-Clustered pairs based on optimal models.

Approach	Case 1	Case 2
Word2doc2vec	110,429	4,590
Hybrid-word2doc2vec	110,302	4,717

Chapter 6

Discussion

Our outcomes demonstrate that text-based embeddings are effective and have potential in predicting protein functions. We defined and trained two embedding approaches, Word2doc2Vec and Hybrid-Word2doc2Vec, built on top of the unsupervised Word2Vec model using protein function text from Swiss-Prot entries. Both approaches successfully encoded functional information into the vector space, allowing for transfer learning.

We found that both approaches were equally efficient in capturing the sequence-function correlation hypothesis when compared to the baseline dataset of sequence clusters from UniRef. The Word2doc2Vec model reflected sequence similarity well in the embedding vector space, despite only being exposed to plain function text without any of the MeSH and GO annotations. However, while Named Entity Recognition-based embeddings have advantages, the Hybrid-Word2doc2Vec model did not show significant improvements. Based on the analysis of the vocabulary size of the models, we found that the Hybrid-Word2doc2Vec model did not significantly reduce the vocabulary build by the models. A possible reason for that, that should be explored futher, is the lack of a large set of named entities in the Swiss-Prot function comment texts, making this approach less suitable for these texts.

6.1 Analysis of protein pairs

The analysis aimed to identify protein candidates for curation by examining protein pairs, which were categorized into three groups in the Results section. These groups consisted of Non-clustered pairs, sub-categorized as Case 1 and Case 2, and False negatives (Case 3) arising from Clustered protein pairs. The study focused on protein pairs belonging to three plant model organisms: Arabidopsis Thaliana, Oryza Sativa (Rice), and Zea mays (Maize). To aid in embedding similarity, specifically cosine similarity, the Word2doc2Vec approach was employed.

- **Case 1: Non-Clustered proteins with high sequence similarity and high embedding similarity**

  An additional analysis was conducted on these protein pairs by comparing them to the latest release of UniProt at the time the results were generated, which was the 2023_01 release. The UniProt ID mapping tool was used to map all Swiss-Prot Eukaryota accessions to the UniRef90 database. Upon comparing these non-clustered proteins with the latest UniProt release, it was discovered that approximately 13,142 protein pairs were eventually clustered. This suggests that the evidence from two sources, embeddings and sequence, could identify protein pairs that should have been clustered in UniProt, but were somehow missed by their clustering algorithm.

- **Case 2: Non-Clustered proteins with high sequence similarity and low embedding similarity**

  In some cases, two proteins may have the same function but lack the same description in the function text. However, upon closer inspection, it is discovered that they have the same function but the function text lacks an adequate description. In such cases, caution should be exercised, and any direct references to accession numbers should be included in the function text. Additionally, including text from the caution comment can provide further information and context. If there is a slight difference in the function text, the threshold on the cosine similarity could be relaxed since the embeddings for the proteins still appear to be very close in the vector space, having a cosine similarity value slightly below 0.90.

  In other cases, two proteins may have a high sequence similarity, but their functions are actually different. In such cases, additional information such as domains and motifs and structural information can be added as part of the function text. These details can provide more context and clarify the differences in function between the proteins.

- **Case 3: Clustered proteins with high sequence similarity and low embedding similarity**

  There are several reasons as to why the sequence similarity might not correlate with functional similarity such as differences in post-translational modifications, such as phosphorylation or glycosylation. These can lead to different functional outcomes even when the amino acid sequence is identical. Moreover, Splice variants, which may have similar sequences but differ in the presence or absence of certain domains, can also result in functional differences.

  To aid in the identification of functional differences, the function text should include evidence tags and GO annotations whenever possible.

Miscellaneous comments could also be useful in providing additional information about the function. Moreover, even small differences in amino acid positions could result in different protein structures and therefore different functions. This underscores the importance of considering not only sequence similarity but also structural information when analyzing protein function.

6.2 Limitations

- This thesis only provides a preliminary evaluation, primarily based on direct inspection and the combination of information from the original text and derived embeddings.
- To generate the embeddings, it would be beneficial to incorporate additional information related to protein function, such as evidence tags, caution comments, and miscellaneous sections from Swiss-Prot. However, these additional sources are beyond the scope of this thesis.
- Based on the analysis of several proteins in Case 2 and Case 3, it appears that a cosine similarity threshold of 0.90 may be overly stringent and could be relaxed further.
- Prediction analyses typically rely on well-annotated function texts to predict the functions of unknown protein entries. In this study, we only utilized Swiss-Prot protein entries which proves as a good starting point. However, it is necessary to translate these findings to propagate the predicted functions to TrEMBL protein entries that lack functional annotations.

6.3 Outlook to future perspectives

The field of Machine/Deep learning is rapidly evolving, and so is any protein function prediction approach relying on these technologies. Current trends suggest that neuro-symbolic learning offers promising avenues for future research. Neuro-symbolic learning combines semantic and deep learning technologies, including the use of large knowledge bases in the form of knowledge graphs. More formally, neuro-symbolic learning aims to combine the strengths of symbolic and sub-symbolic systems to improve predictive models' performance and explainability [51]. Symbolic systems use knowledge bases and logical deduction to solve tasks, while sub-symbolic methods employ machine learning, particularly artificial neural networks, to handle unstructured data. Although sub-symbolic methods have shown superiority over humans in tasks such as text processing, they are generally black boxes that cannot be inspected. In contrast, symbolic systems can be inspected to interpret how a decision follows from

input. Symbolic and sub-symbolic systems complement each other, and they differ in the types of problems and data they excel at.

In the case of protein function prediction, the semantic layer coming from ontologies can be combined with deep learning approaches such as embeddings to capture more nuanced relationships between proteins and their functions, leading to more accurate predictions.

To cover a broader spectrum for function prediction it is essential to combine semantics with other data sources, such as gene expression data, protein-protein interaction networks, and drug-target associations, to complement the limited information from protein annotations. The replacement of words by concepts is a step in the right direction, as it combines terms from ontologies with embeddings. Still, more work is needed, and more options exist. For instance, Graph embeddings, which can integrate heterogeneous data sources into a unified representation, can be used to get embeddings from ontologies as already targeted by the mOWL library [52], in combination with knowledge graphs, and word embeddings, can improve the accuracy and robustness of function prediction models.

However, combining embeddings from multiple sources can be challenging as different sources may have different scales, levels of noise, and quality of information. Therefore, more research is needed to develop effective methods for integrating and combining embeddings from multiple sources to improve the accuracy and robustness of function prediction models.

Overall, the integration of neuro-symbolic learning and knowledge graphs offers a promising approach to protein function prediction and link prediction tasks, by combining the strengths of neural networks and symbolic reasoning with the ability to integrate diverse data sources in a unified representation. This can help to better understand the complex relationships between proteins and their functions.

Chapter 7

Conclusion

This thesis serves as an exploratory assessment that demonstrates the potential of using embeddings generated from protein function texts to propagate them into the vector space. This provides a promising starting point for further research to explore the use of embeddings in prediction analysis with the help of TrEMBL entries. The findings also suggest the possibility of improving the way function vectors are compared in the vector space model and exploring beyond similarity metrics. Furthermore, the use of embeddings offers a pathway to improve functional annotations and create a controlled vocabulary for functional comments.

Although function prediction is a well-established area of study due to its importance in biology, pharmaceuticals, and life sciences in general, further research is required to advance, for instance, predictions concerning drug and protein associations. To achieve this, we need to explore neuro-symbolic learning to encompass a broader spectrum that includes ontologies as background knowledge.

Bibliography

[1] Amelia Villegas-Morcillo et al. 'Unsupervised protein embeddings outperform hand-crafted sequence and structure features at predicting molecular function'. In: *Bioinformatics* 37.2 (Aug. 2020). Ed. by Arne Elofsson, pp. 162–170. DOI: 10.1093/bioinformatics/btaa701. URL: https://doi.org/10.1093/bioinformatics/btaa701.

[2] Stephen F Altschul et al. 'Basic local alignment search tool'. In: *Journal of molecular biology* 215.3 (1990), pp. 403–410.

[3] The UniProt Consortium. 'UniProt: the Universal Protein Knowledgebase in 2023'. In: *Nucleic Acids Research* 51.D1 (Nov. 2022), pp. D523–D531. ISSN: 0305-1048. DOI: 10.1093/nar/gkac1052. eprint: https://academic.oup.com/nar/article-pdf/51/D1/D523/48441158/gkac1052.pdf. URL: https://doi.org/10.1093/nar/gkac1052.

[4] Rohitha Ravinder, Leyla Jael Castro and Dietrich Rebholz-Schuhmann. *Protein Function Embeddings: First Beta Release*. Version v1.0.1. Mar. 2023. DOI: 10.5281/zenodo.7781870. URL: https://doi.org/10.5281/zenodo.7781870.

[5] Rohitha Ravinder, Leyla Jael Castro and Dietrich Rebholz-Schuhmann. *Protein Function Embeddings: First Beta Release of Datasets*. Version v1.0.0. Zenodo, Apr. 2023. DOI: 10.5281/zenodo.7793384. URL: https://doi.org/10.5281/zenodo.7793384.

[6] Michael Ashburner et al. 'Gene ontology: tool for the unification of biology'. In: *Nature genetics* 25.1 (2000), pp. 25–29.

[7] Baris E Suzek et al. 'UniRef clusters: a comprehensive and scalable alternative for improving sequence similarity searches'. In: *Bioinformatics* 31.6 (2015), pp. 926–932.

[8] Ronan Collobert and Jason Weston. 'A unified architecture for natural language processing: Deep neural networks with multitask learning'. In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 160–167.

[9] Billy Chiu and Simon Baker. 'Word embeddings for biomedical natural language processing: A survey'. In: *Language and Linguistics Compass* 14.12 (2020), e12402.

[10] Shirui Wang, Wenan Zhou and Chao Jiang. 'A survey of word embeddings based on deep learning'. In: *Computing* 102 (2020), pp. 717–740.

[11] Christopher D Manning. *An introduction to information retrieval*. Cambridge university press, 2009.

[12] Yoav Goldberg. 'Neural network methods for natural language processing'. In: *Synthesis lectures on human language technologies* 10.1 (2017), pp. 1–309.

[13] Jochen Hirschle. *Deep Natural Language Processing: Einstieg in Word Embedding, Sequence-to-Sequence-Modelle und Transformer mit Python*. Carl Hanser Verlag GmbH Co KG, 2022.

[14] John Firth. 'A synopsis of linguistic theory, 1930-1955'. In: *Studies in linguistic analysis* (1957), pp. 10–32.

[15] Zellig S Harris. 'Distributional structure'. In: *Word* 10.2-3 (1954), pp. 146–162.

[16] Jeffrey Pennington, Richard Socher and Christopher D Manning. 'Glove: Global vectors for word representation'. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[17] Jacob Devlin et al. 'Bert: Bidirectional encoder representations from transformers'. In: (2016).

[18] Tomas Mikolov et al. 'Efficient estimation of word representations in vector space'. In: *arXiv preprint arXiv:1301.3781* (2013).

[19] Vishwas Bhanawat. *The Architecture of Word2Vec*. `https://medium.com/@vishwasbhanawat/the-architecture-of-word2vec-78659ceb6638`. Accessed on March 31, 2023. Feb. 2019.

[20] Tomas Mikolov, Quoc V Le and Ilya Sutskever. 'Exploiting similarities among languages for machine translation'. In: *arXiv preprint arXiv:1309.4168* (2013).

[21] David E Rumelhart, Geoffrey E Hinton and Ronald J Williams. 'Learning representations by back-propagating errors'. In: *nature* 323.6088 (1986), pp. 533–536.

[22] Quoc Le and Tomas Mikolov. 'Distributed representations of sentences and documents'. In: *International conference on machine learning*. PMLR. 2014, pp. 1188–1196.

[23] Sukrat Gupta et al. 'Task-optimized word embeddings for text classification representations'. In: *Frontiers in Applied Mathematics and Statistics* 5 (2020), p. 67.

[24] Fernando Enrıquez, José A Troyano and Tomás López-Solaz. 'An approach to the use of word embeddings in an opinion classification task'. In: *Expert Systems with Applications* 66 (2016), pp. 1–6.

[25] Dieter Galea, Ivan Laponogov and Kirill Veselkov. 'Exploiting and assessing multi-source data for supervised biomedical named entity recognition'. In: *Bioinformatics* 34.14 (2018), pp. 2474–2482.

[26] Rion Snow et al. 'Cheap and fast–but is it good? evaluating non-expert annotations for natural language tasks'. In: *Proceedings of the 2008 conference on empirical methods in natural language processing*. 2008, pp. 254–263.

[27] Xu Wang, Chen Yang and Renchu Guan. 'A comparative study for biomedical named entity recognition'. In: *International Journal of Machine Learning and Cybernetics* 9 (2018), pp. 373–382.

[28] Hyejin Cho, Wonjun Choi and Hyunju Lee. 'A method for named entity normalization in biomedical articles: application to diseases and plants'. In: *BMC bioinformatics* 18.1 (2017), pp. 1–12.

[29] National Library of Medicine (US). *National Library of Medicine*. Accessed: March 22, 2021. 2021. URL: https://www.nlm.nih.gov/.

[30] National Library of Medicine (US). *Medical Subject Headings (MeSH)*. Accessed: March 22, 2021. 2021. URL: https://www.ncbi.nlm.nih.gov/mesh.

[31] The Gene Ontology Consortium. 'The Gene Ontology resource: enriching a GOld mine'. In: *Nucleic Acids Research* 49.D1 (2021), pp. D325–D334. DOI: 10.1093/nar/gkaa1113. eprint: https://academic.oup.com/nar/article-pdf/49/D1/D325/36385438/gkaa1113.pdf. URL: https://doi.org/10.1093/nar/gkaa1113.

[32] Dietrich Rebholz-Schuhmann et al. 'Text processing through Web services: calling Whatizit'. In: *Bioinformatics* 24.2 (Nov. 2007), pp. 296–298. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btm557. eprint: https://academic.oup.com/bioinformatics/article-pdf/24/2/296/49044576/bioinformatics_24_2_296.pdf. URL: https://doi.org/10.1093/bioinformatics/btm557.

[33] Harald Kirsch, Sylvain Gaudan and Dietrich Rebholz-Schuhmann. 'Distributed modules for text annotation and IE applied to the biomedical domain'. In: *International journal of medical informatics* 75.6 (2006), pp. 496–500.

[34] *GitHub - HaraldKi/monqjfa: NFA and DFA implementation in Java — github.com*. https://github.com/HaraldKi/monqjfa. [Accessed 21-Mar-2023].

[35] Joaquim Gómez Sánchez. *Analysis and Comparison of Text Similarity Measures*. 2014. URL: https://upcommons.upc.edu/bitstream/handle/2117/343180/155956.pdf.

[36] Wikipedia. *Cosine similarity — Wikipedia, The Free Encyclopedia*. [Online; accessed 25-March-2023]. 2023. URL: https://en.wikipedia.org/w/index.php?title=Cosine_similarity&oldid=1051437197.

[37] M.O. Dayhoff, R.M. Schwartz and B.C. Orcutt. 'Amino-Acid Sequence Divergence among Homologous Proteins'. In: *Atlas of Protein Sequence and Structure* 5.Suppl. 3 (1969), pp. 345–352.

[38] Jeff Kunce. *xmltree: An XML tree library for Python*. https://pypi.org/project/xmltree/. Accessed on March 9, 2023. 2012.

[39] Carl Boettiger. *rdflib: A high level wrapper around the redland package for common rdf applications*. Zenodo, 2018. DOI: 10.5281/zenodo.1098478. URL: https://doi.org/10.5281/zenodo.1098478.

[40] Leonard Richardson. 'Beautiful Soup: a Python package for parsing HTML and XML'. In: *Journal of Open Source Software* 2.18 (2007), p. 97.

[41] Steven Bird, Edward Loper and Ewan Klein. 'Natural Language Processing with Python'. In: *O'Reilly Media, Inc.* (2006). URL: https://www.nltk.org/.

[42] Python Software Foundation. *Python v3.10.0 documentation*. Python Software Foundation. 2021. URL: https://docs.python.org/3/library/re.html.

[43] Radim Řehůřek and Petr Sojka. 'Software Framework for Topic Modelling with Large Corpora'. In: (2010), pp. 45–50.

[44] Stanley Lam, Antoine Pitrou and Mark Seibert. *Numba: A LLVM-based Python JIT compiler*. http://numba.pydata.org/. 2015.

[45] Python Software Foundation. *itertools – Functions creating iterators for efficient looping*. Accessed: March 23, 2023. Python Software Foundation. https://docs.python.org/3/library/itertools.html, 2021.

[46] Charles R. Harris et al. 'Array programming with NumPy'. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[47] Mahesh Hegde et al. 'ElasticBLAST: scalable and efficient BLAST searches for the cloud'. In: *Nucleic Acids Research* 48.W1 (2020), W537–W543.

[48] Mahesh Hegde et al. *ElasticBLAST: Scalable and efficient BLAST searches for the cloud*. Version 1.1. 2020. URL: https://github.com/elasticblast/elasticblast.

[49] Johanna McEntyre and Jim Ostell. 'The NCBI Handbook'. In: *National Center for Biotechnology Information* (2017). URL: https://www.ncbi.nlm.nih.gov/books/NBK143764/.

[50] Peter J. A. Cock et al. *Biopython: freely available tools for computational molecular biology and bioinformatics*. Bioinformatics, 2009. URL: http://biopython.org.

[51] Semantic Web Journal. *Call for Papers: Special Issue on Neuro-Symbolic Artificial Intelligence and Semantic Web*. https://www.semantic-web-journal.net/blog/call-papers-special-issue-neuro-symbolic-artificial-intelligence-and-semantic-web. Dec. 2022.

[52] Fernando Zhapa-Camacho, Maxat Kulmanov and Robert Hoehndorf. 'mOWL: Python library for machine learning with biomedical ontologies'. In: *Bioinformatics* 39.1 (2023), btac811.

Declaration

I hereby certify that this material is my own work, that I used only those sources and resources referred to in the thesis, and that I have identified citations as such.

Rohitha Ravinder

Bonn, April 17, 2023