Imperial College London

Natural Language Processing Project

Imperial College London

Data Science Winter Schoo Group5

Report of the NLP Project: Word Representation in Biomedical Domain

Author:
Xi Chen
Aier Yang
Xinyi Huang

February 4, 2024

Report of the NLP Project: Word Representation in Biomedical Domain

GROUP 5
Xi Chen
Aier Yang
Xinyi Huang

Contents

List of Figures

List of Tables

Report of the NLP Project: Word Representation in Biomedical Domain

***Abstract*—This project aims to explore the application of Natural Language Processing (NLP) techniques in the biomedical field, with a focus on the construction of word representations and related exploration. The project comprises four main stages: data parsing, tokenization, word representation construction, and in-depth investigation of word representations. In the first stage, fields from existing biomedical corpora were extracted. In the second stage, the extracted text underwent preprocessing steps, including tokenization, cleaning, and part-of-speech tagging. Following tokenization, the third stage focused on constructing word representations using both n-gram and skip-gram methods for the extracted words. The fourth stage involves visualizing word representations and exploring relationships between biomedical entities through co-occurrence analysis and semantic similarity analysis. Co-occurrence analysis identifies biomedical entities frequently co-occurring with COVID-19 and lists them in ranked order. Semantic similarity analysis determines biomedical entities most semantically similar to COVID-19, enhancing the understanding of relationships between words. Finally, the fifth part involves extracting intelligent information from biomedical text corpora, including biomedical entities, relationships between entities, clinical features, and insights into disease development, constructing a knowledge graph related to COVID-19. Overall, through comprehensive data processing, tokenization, word representation construction, and in-depth investigation of word relationships, this project provides valuable insights into natural language analysis of biomedical texts, particularly those related to COVID-19.**

I. Introduction

A. Background

In the rapidly evolving landscape of biomedical and artificial intelligence research, the significance of Natural Language Processing (NLP) technology in handling biomedical-related textual data is becoming increasingly prominent. Against this backdrop, this study focuses on the application of NLP techniques, with a specific emphasis on word representations within the biomedical domain. With the continuous evolution of COVID-19 and related coronaviruses, an in-depth analysis of relevant research literature becomes a crucial aspect of understanding and addressing this global challenge.

B. Objectives

The primary objective of this study is to effectively extract information related to COVID-19 from biomedical texts and construct meaningful word representations to enhance our understanding of the disease. Our goal is to provide valuable insights into biomedical research through comprehensive data parsing, tokenization, word representation construction, and word relationship analysis.

C. Dataset source

The Dataset used in this project is **COVID-19 Open Research Dataset (CORD-19).** [4]

ORD-19 is a corpus composed of academic papers, focusing on research related to COVID-19 and other coronaviruses. Planned and maintained by the Semantic Scholar team at the Allen Institute for AI, this corpus is designed to support text mining and natural language processing research. The extensive coverage of this dataset makes it a crucial resource in the field of biomedical research, providing a foundational basis for in-depth analysis and exploration in our research endeavors. In this report, we will briefly outline the methodology of the study and anticipate its contributions to the field of biomedical research. Through this research, we aim to advance a deeper understanding of biomedical texts, particularly in the context of COVID-19 research, and offer new insights for future medical investigations. In the following sections, we will delve into the background, problem statement, methodology, and expected contributions of the study to comprehensively showcase our research progress.

II. PROTOCOL

A. Design ideas

The following diagram is our mind map to solve the problem.

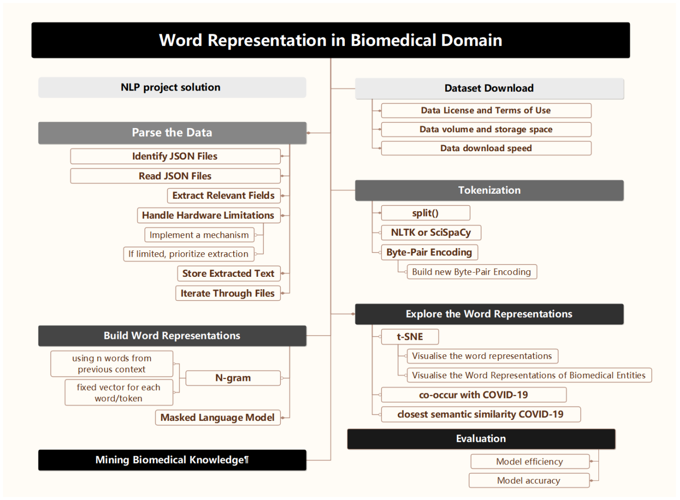

Word Representation in Biomedical Domain
NLP project solution
Parse the Data
Identify JSON Files
Read JSON Files
Extract Relevant Fields
Handle Hardware Limitations
Implement a mechanism
If limited, prioritize extraction
Store Extracted Text
Iterate Through Files
Build Word Representations
using n words from previous context
fixed vector for each word/token
N-gram
Masked Language Model
Mining Biomedical Knowledge¶
Dataset Download
Data License and Terms of Use
Data volume and storage space
Data download speed
Tokenization
split()
NLTK or SciSpaCy
Byte-Pair Encoding
Build new Byte-Pair Encoding
Explore the Word Representations
t-SNE
Visualise the word representations
Visualise the Word Representations of Biomedical Entities
co-occur with COVID-19
closest semantic similarity COVID-19
Evaluation
Model efficiency
Model accuracy

Fig. 1. mind map

B. Environment and Tools

Deep Learning Frameworks:TensorFlow PyTorch Keras
Natural Language Processing Tools:SpaCy NLTK gensim
Word Embedding Models:Word2Vec GloVe FastText
Pre-trained Language Models:BERT GPT (Generative Pre-trained Transformer) ELMO (Embeddings from Language Models)
Text Annotation Tools:BRAT (BioNLP Rapid Annotation Tool) Prodigy
Data Cleaning and Processing:pandas NumPy
Statistical Analysis:SciPy StatsModels
Visualization Tools:Matplotlib Seaborn Plotly
Web Frameworks:Flask Django
Version Control:Git

Project Runtime Environment:

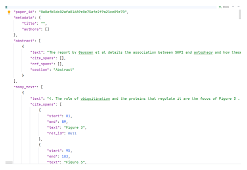
- Operating System: Windows 11Operating System: Windows 11
- Python Version: 3.7
- PyCharm Version: 2023.2.2
- Jupyter Version: 6.5.4

III. Part1-Data Parsing

The goal of the first part is to navigate through a considerable number of JSON files (approximately 5.8GB), and given the issues of large datasets and computational resources, we only summarised the text of the abstract in the field extracted. The dataset used in this project is part of the COVID-19 Open Research Dataset (CORD-19) built by Wang et al. and is available at https://www.semanticscholar.org/cord19/download.
The parsing task is performed using Python, specifically utilizing the os and json modules.

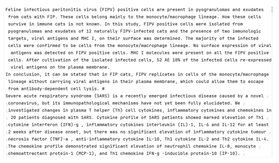
```
# Iterate through all JSON files in a
    folder
abstract_texts = []
for filename in os.listdir(folder_path):
if filename.endswith('.json'):
file_path = os.path.join(folder_path,
    filename)
with open(file_path, 'r', encoding='utf-8'
    ) as json_file:
try:
data = json.load(json_file)

 # Handle the case where the abstract
    field may be a list
 abstract_list = data.get('abstract', [])

if isinstance(abstract_list, list):
for abstract_item in abstract_list:
abstract_text = abstract_item.get('text',
    '')
abstract_texts.append(abstract_text)
 else:
 abstract_text = abstract_list.get('text',
     '')
 abstract_texts.append(abstract_text)
except json.JSONDecodeError:
print(f"Error decoding JSON in file: {
    file_path}")
```

This script smartly handles variations in the structure of the abstract field. If the abstract is a list, it iterates through each item, extracting the text field and appending it to the 'abstract-texts' list. If 'abstract' is a single abstract, it directly retrieves the text field. Also, exception handling is implemented to manage potential JSON decoding errors. By iteratively processing all JSON files in the specified folder, the script extracts text from the 'abstract' field, appending the results to the abstract-texts list. Subsequently, the script writes the extracted text to a new text file, creating a concise record of abstracts from the dataset. Finally, the text extraction from the JSON file in the specified folder was completed, providing a basis for further analysis or processing of the CORD-19 data set.
The following figure shows the comparison of the results before and after extraction.



```
"paper_id": "0a0afb5dc02afa81689e8e75afe2f9a21ce89e70",
"metadata": {
    "title": "",
    "authors": []
},
"abstract": [
    {
        "text": "The report by Gaussen et al details the association between SKP2 and autophagy and how these
        "cite_spans": [],
        "ref_spans": [],
        "section": "Abstract"
    }
],
"body_text": [
    {
        "text": "4. The role of ubiquitination and the proteins that regulate it are the focus of Figure 3 .
        "cite_spans": [
            {
                "start": 81,
                "end": 89,
                "text": "Figure 3",
                "ref_id": null
            },
            {
                "start": 95,
                "end": 103,
                "text": "Figure 3",
```

Fig. 2. Before extraction



```
Feline infectious peritonitis virus (FIPV) positive cells are present in pyogranulomas and exudates
 from cats with FIP. These cells belong mainly to the monocyte/macrophage lineage. How these cells
 survive in immune cats is not known. In this study, FIPV positive cells were isolated from
 pyogranulomas and exudates of 12 naturally FIPV-infected cats and the presence of two immunologic
 targets, viral antigens and MHC I, on their surface was determined. The majority of the infected
 cells were confirmed to be cells from the monocyte/macrophage lineage. No surface expression of viral
  antigens was detected on FIPV positive cells. MHC I molecules were present on all the FIPV positive
 cells. After cultivation of the isolated infected cells, 52 AE 10% of the infected cells re-expressed
  viral antigens on the plasma membrane.
In conclusion, it can be stated that in FIP cats, FIPV replicates in cells of the monocyte/macrophage
 lineage without carrying viral antigens in their plasma membrane, which could allow them to escape
 from antibody-dependent cell lysis. #
Severe acute respiratory syndrome (SARS) is a recently emerged infectious disease caused by a novel
 coronavirus, but its immunopathological mechanisms have not yet been fully elucidated. We
 investigated changes in plasma T helper (Th) cell cytokines, inflammatory cytokines and chemokines in
  20 patients diagnosed with SARS. Cytokine profile of SARS patients showed marked elevation of Th1
 cytokine interferon (IFN)-g , inflammatory cytokines interleukin (IL)-1, IL-6 and IL-12 for at least
 2 weeks after disease onset, but there was no significant elevation of inflammatory cytokine tumour
 necrosis factor (TNF)-a , anti-inflammatory cytokine IL-10, Th1 cytokine IL-2 and Th2 cytokine IL-4.
 The chemokine profile demonstrated significant elevation of neutrophil chemokine IL-8, monocyte
 chemoattractant protein-1 (MCP-1), and Th1 chemokine IFN-g -inducible protein-10 (IP-10).
```

Fig. 3. After extraction

IV. Part2-Tokenization

A. Use the split function for word segmentation and data preprocessing

After splitting the text into words by using Python's 'split' function, we used the regular expression 're.compile()' to remove punctuation and numbers to avoid subsequent text processing and analysis are subject to unnecessary interference. We also added a 'tqdm' progress bar display, iterating through each line of the file, processing each line, and adding the segmented words to the list 'segmented-words'. However, this method simply splits the words and does not provide further data preprocessing.

B. Use NLTK for word segmentation and data preprocessing

For further data preprocessing, we used online stop word resources and the NLTK library to segment the text data into sentences, and words, remove stop words, use regular expressions to exclude punctuation, and convert uppercase letters to lowercase [2].
Removing stop words and punctuation marks avoids excessive text dimensionality when embedding subsequent words. At the same time, for N-gram and skip-gram, frequent stop words will affect the accuracy of prediction and reduce the accuracy of actual semantics. Concentration of information. Converting uppercase to lowercase ensures that word embeddings do not treat the same words as different words. The purpose of dividing

the word segmentation results into sentences and storing them by row is to more accurately capture the sentence-level semantics and structure in subsequent steps.

C. Use Byte Pair Encoding (BPE) tokenizer

Compared with NLTK's static vocabulary method, BPE based on the data-adaptive method can better handle out-of-vocabulary words. Since this project is based on literature in the biomedical field, there will inevitably be professional terms that do not exist in the static vocabulary, which will affect the preprocessing effect. Therefore, we used BEP for word tokenization to perform more fine-grained segmentation of words while processing professional vocabulary more accurately.

D. New Byte-Pair Encoding (BPE)

• Model Implementation

Based on 2.3, we built our own BPE tokenizer. The implementation principle can be divided into three main steps: data preprocessing, merging character pairs, and tokenization. Finally, we evaluated the tokenization performance of this model.

Data Preprocessing: Load word from files, build vocabulary, and count the frequency of each word; Count the frequency of each character pair.

```
def get_vocab(filename):
    vocab = collections.defaultdict(int)
    with open(filename, 'r', encoding='utf-8') as fhand:
        for line in fhand:
            words = line.strip().split()
            for word in words:
                vocab[' '.join(list(word)) + ' </w>'] += 1
    return vocab
```

Merging Character Pairs:

Using the ‘re.compile()‘ function to create a regular expression object ‘p‘ for matching character pairs in the vocabulary; In the loop, using the ‘p.sub()‘ method to replace the original character pairs with the merged characters, obtaining the merged word ‘w-out‘, and adding it to the dictionary ‘v-out‘.

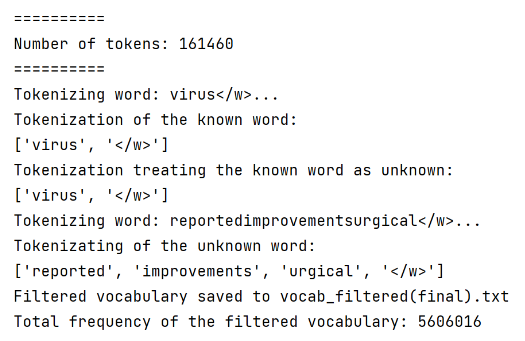
```
def merge_vocab(pair, v_in):
    v_out = {}
    bigram = re.escape(' '.join(pair))
    p = re.compile(r'(?<!\S)' + bigram + r'(?!\S)')
    for word in v_in:
        w_out = p.sub(''.join(pair), word)
        v_out[w_out] = v_in[word]
    return v_out
```

Tokenization: First, locate all matching positions of the current token in the input string ‘string‘, and record the starting and ending positions of each match. Next, process each matching position by recursively tokenizing the substring before the match. Then, add the current token to the token list and update the starting position of the substring, followed by recursively tokenizing the remaining substring. After all tokens have been processed, return the tokenized string list.

```
for i in range(len(sorted_tokens)):
    token = sorted_tokens[i]  # Get the current token
    token_reg = re.escape(token.replace('.', '[.]'))  # Escape special characters and replace '.' with '[.]'
  # ......steps omitted.
      # Get the remaining substring.
    remaining_substring = string[substring_start_position:]
    # Tokenize the remaining substring recursively.
    string_tokens += tokenize_word(string=remaining_substring, sorted_tokens=sorted_tokens[i + 1:],unknown_token=unknown_token)
      break
```

• Model evaluation

We use some code snippet to check the tokenization process for both known and unknown words. First, it sorts the tokens based on their frequencies, sorting them in descending order by their lengths and frequencies. Then, it tokenizes the known and unknown words using the sorted tokens list and prints the results.



```
==========
Number of tokens: 161460
==========
Tokenizing word: virus</w>...
Tokenization of the known word:
['virus', '</w>']
Tokenization treating the known word as unknown:
['virus', '</w>']
Tokenizing word: reportedimprovementsurgical</w>...
Tokenizating of the unknown word:
['reported', 'improvements', 'urgical', '</w>']
Filtered vocabulary saved to vocab_filtered(final).txt
Total frequency of the filtered vocabulary: 5606016
```

Fig. 4. BPE results display

E. Discussion of Different Tokenization Methods

Based on the comparison and analysis of the above results, we get a comparison of the advantages and disadvantages of different tokenization methods.

V. Part3-Build Word Representations

A. N-gram model

• Introduction to the Principle

N-Gram is an algorithm based on statistical language models. Its fundamental concept involves the application of a sliding window of size N, measured in bytes, to the content within a text. This operation generates a sequence of byte fragments with a length of N. Each byte

Method	Pros	Cons
Split	Simple and quick	Limited data preprocessing beyond basic word splitting
NLTK	Comprehensive data preprocessing & sentences segmentation	Out-of-vocabulary words
BPE	Well-suited for processing professional terms; Ability to handle compound words and rare terms; Preserves context information better than basic splitting; Suitable for handling languages with rich morphology	Computationally intensive; May generate a large vocabulary size; Requires pre-training on a large corpus

TABLE I. COMPARISON OF SPLIT, NLTK, AND BPE.

fragment is referred to as a "gram." The algorithm then performs a statistical analysis of the frequency of occurrence for each gram. Following this, a predetermined threshold is applied to filter the grams, resulting in a list of key grams. This list forms the vector feature space of the text, where each gram represents a dimension in the feature vector.

The model operates under the assumption that the appearance of the Nth word is only correlated with the preceding N-1 words and is independent of any other words. The probability of the entire sentence is the product of the probabilities of each individual word occurrence.

$$P(w_1 w_2 \dots w_n) = P(w_1) \cdot P(w_2|w_1) \dots P(w_m|w_1, \dots, w_{m-1}) \tag{1}$$

This probability is evidently difficult to compute. It may be beneficial to leverage the assumption ofm a Markov chain, which posits that the current word is only related to a limited number of preceding words. Hence, there is no need to trace back to the very first word, significantly reducing the length of the aforementioned equation.

$$P(w_n|w_{n-1}, w_{n-2}, ..., w_1) \tag{2}$$

By utilizing Bayes' theorem, the conditional probability values mentioned above can be statistically calculated:

$$P(w_n|w_{n-1}, w_{n-2}) = \frac{\text{count}(w_{n-2}, w_{n-1}, w_n)}{\text{count}(w_{n-2}, w_{n-1})} \tag{3}$$

In the selection of N, as N increases, there is more constraint information on the occurrence of the next word, resulting in a more accurate model but requiring greater computational resources. After considering the size of the dataset and computational capabilities, we opted for the Tri-Gram model (N=3).

• **Model Implementation**

The model architecture consists of an embedding layer to represent input word indices as dense vectors, followed by a linear layer acting as a hidden layer with ReLU activation, and another linear layer serving as the output layer [5].

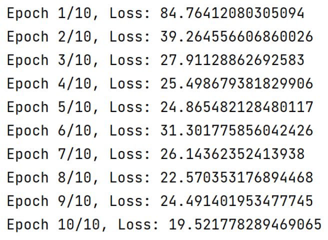
Embedding layer The embedding layer is initialized with a randomly initialized embedding matrix of size '(vocab-size, embed-size)', where 'vocab-size' represents the vocabulary size and 'embed-size' represents the dimensionality of the word embeddings. During training, this embedding matrix is learned and optimized alongside other model parameters.

Linear Layer with ReLU activation The embedding layer is followed by a linear layer, which serves as a hidden layer in the network. This hidden layer applies the Rectified Linear Unit (ReLU) activation function, introducing non-linearity to the model's transformations.

Output layer There is another linear layer, acting as the output layer, responsible for generating the final predictions based on the learned representations [3].

```
class NGramModel(nn.Module):
    def __init__(self, vocab_size,
    embed_size, context_size):
        super(NGramModel, self).__init__()
        self.embeddings = nn.Embedding(
    vocab_size, embed_size)
        self.linear1 = nn.Linear(
    context_size * embed_size, 128)
        self.linear2 = nn.Linear(128,
    vocab_size)
```

• **Model Training and Result Evaluation** The following is the process and results of model training.



```
Epoch 1/10, Loss: 84.76412080305094
Epoch 2/10, Loss: 39.264556606860026
Epoch 3/10, Loss: 27.91128862692583
Epoch 4/10, Loss: 25.498679381829906
Epoch 5/10, Loss: 24.865482128480117
Epoch 6/10, Loss: 31.301775856042426
Epoch 7/10, Loss: 26.14362352413938
Epoch 8/10, Loss: 22.570353176894468
Epoch 9/10, Loss: 24.491401953477745
Epoch 10/10, Loss: 19.521778289469065
```

Fig. 5. Training Course

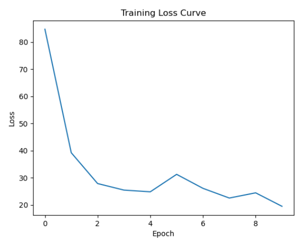


Fig. 6. Training Loss Curve

B. Skip-gram with Negative Sampling

• **Introduction to the Principle**

Skip-gram The Skip-gram model is a neural network-based model for learning word embeddings, belonging to the family of techniques known as word embeddings. It is an unsupervised learning method based on neural networks, designed to learn distributed representations of words from large-scale text corpora.

Firstly, each word in the text corpus is represented as a one-hot vector, where only one element is 1 and the rest are 0s. Then, the Skip-gram model typically consists of two layers of neural networks: an input layer and an output layer. The number of units in the input layer equals the size of the vocabulary, while the number of

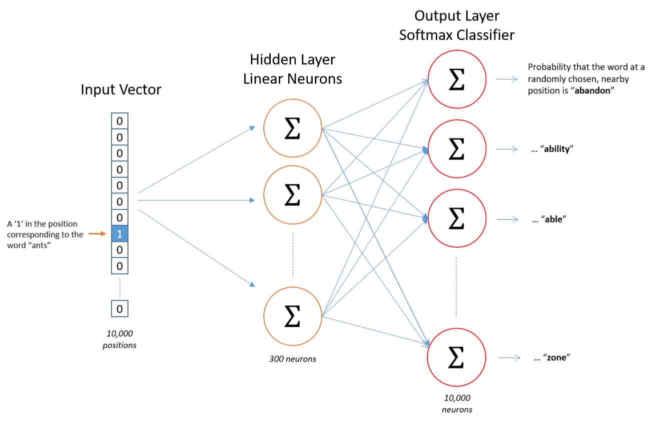


Fig. 7. Network diagram1

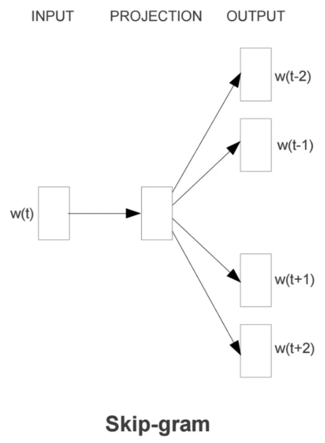


Fig. 8. Network diagram2

units in the output layer equals the dimensionality of the word vectors [6].

Negative Sampling Negative sampling is a technique used to accelerate training and improve the quality of obtained word embeddings. Unlike the conventional approach where each training sample updates all weights, negative sampling involves updating only a small subset of weights for each training sample. This reduces the computational overhead during gradient descent.

• **Model Implementation**

Embedding Layer The in-embed layer initialized with a randomly initialized embedding matrix of size (vocab-size, embed-size). It is used to map word indices from the vocabulary to dense vector representations.

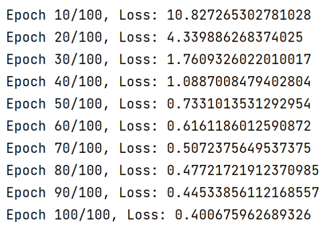
Huffman parameter layer This layer Implemented Huffman tree to replace the mapping from the hidden layer to the output softmax layer. And the size of the Huffman coding parameter layer needs to match the dimension of the output from the embedding layer.

```
class SkipGramModelWithHuffman(nn.Module):
    def __init__(self, vocab_size,
    embed_size, huffman_size):
        super(SkipGramModelWithHuffman,
    self).__init__()
        self.in_embed = nn.Embedding(
    vocab_size, embed_size)
        self.out_huffman = nn.Linear(
    embed_size, huffman_size)
    self.vocab_size = vocab_size
```

Forward function It employs binary cross-entropy loss function, considering both positive and negative samples. By integrating negative sampling techniques, the function circumvents the necessity to compute probabilities for all potential context words. This strategic approach effectively mitigates computational complexity and accelerates the training process.

```
# Negative sampling
neg_context = torch.randint(0, self.
    vocab_size, (batch_size,
    num_neg_samples), device=target.device
    )
neg_out_embeds = self.out_embed(
    neg_context)
```

• **Model Training and Result Evaluation** The following is the process and results of model training.



```
Epoch 10/100, Loss: 10.827265302781028
Epoch 20/100, Loss: 4.339886268374025
Epoch 30/100, Loss: 1.7609326022010017
Epoch 40/100, Loss: 1.0887008479402804
Epoch 50/100, Loss: 0.7331013531292954
Epoch 60/100, Loss: 0.6161186012590872
Epoch 70/100, Loss: 0.5072375649537375
Epoch 80/100, Loss: 0.47721721912370985
Epoch 90/100, Loss: 0.44533856112168557
Epoch 100/100, Loss: 0.400675962689326
```

Fig. 9. Training Course

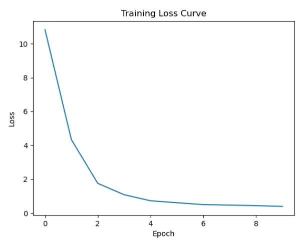


Fig. 10. Training Loss Curve

C. Masked Language Model (MLM)

• **Basic Principles** BERT stands for Bidirectional Encoder Representations from Transformers. It is a pre-trained language representation model. BERT emphasizes moving away from traditional unidirectional language models or shallow concatenation of two unidirectional language models for pre-training. Instead, it adopts a new approach called masked language model (MLM) to generate deep bidirectional language representations [7].

• **Code implement**

Although the implementation of the code is relatively simple, the configuration of the environment and the downloading and loading of the pre-trained models are quite challenging. We followed the instructions on the website "https://github.com/dmis-lab/biobert?tab=readme-ov-file" to complete the above steps.

VI. Part4-Explore the Word Representations

A. *Visualise the word representations by t-SNE*

• Basic Principles
t-SNE is a nonlinear dimensionality reduction technique used to map high-dimensional data into two or three-dimensional space. Its main idea is to measure the similarity of data points in high-dimensional and low-dimensional space based on probability distributions, and then optimize the mapping by minimizing the difference between these two distributions. During optimization, t-SNE attempts to map similar data points to neighboring positions in the low-dimensional space, thereby preserving their similarity relationships.

• Code Implement
First, load the vector representations of words and convert the vectors into numpy arrays.

```
# Extract words and vector representations
words, vectors = zip(*[(line[0], list(map(
    float, line[1:]))) for line in
    word_vectors])
```

Choose the parameter n-components for t-SNE to determine whether to reduce dimensionality to two or three dimensions.

```
# Convert lists to numpy arrays
vectors = np.array(vectors)

# Choose t-SNE parameters
tsne = TSNE(n_components=2, random_state
    =42)

# Use t-SNE for dimensionality reduction
word_tsne = np.zeros((len(words), 2))  #
    Initialize array to store t-SNE
    results
for i in tqdm(range(0, len(words), 1000),
    desc="t-SNE Progress"):
     end_idx = min(i + 1000, len(words))
     word_tsne[i:end_idx] = tsne.
    fit_transform(vectors[i:end_idx])
```

Finally, visualize the dimensionality reduction results.

```
plt.figure(figsize=(10, 8))
plt.scatter(word_tsne[:, 0], word_tsne[:,
    1], alpha=0.5, s=10)
```

• Results Diaplay The following figure shows the visualization results.

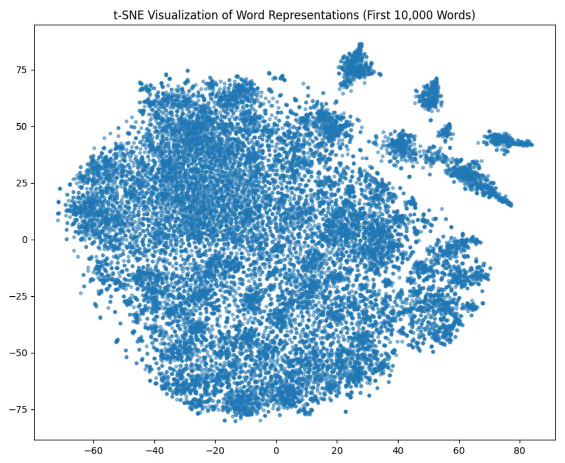


Fig. 11. t-SNE visualization

B. *Visualise the Word Representations of Biomedical Entities by t-SNE*

• Approach
Besides obtaining the word list through the link provided by the tutorial,We accessed the list of biomedical entity names through the website's API (https://www.ebi.ac.uk/ols/ontologies/hp) [1] and matched them one by one with the dimensionally reduced word vector representations obtained from 4.1. We finally filtered out a total of xxx biomedical entities contained in the text and visualized them.

• Code implement
Obtaining Biomedical Entities List: This function retrieves the list of biomedical entities from a specific ontology by making calls to the EBI OLS (Ontology Lookup Service) API. The parameter 'ontology-id' of the function is a string used to specify the ID of the desired ontology. Subsequently, the function constructs a URL to make a GET request to the OLS API in order to fetch the term list for the specified ontology. If the request is successful (HTTP status code 200), the terms from the response are extracted and returned as a list of biomedical entities. If the request fails (HTTP status code is not 200), an error message is printed, and an empty list is returned.

```
def get_biomedical_entities_from_ols(
    ontology_id):
    ols_api_url = f'https://www.ebi.ac.uk/
    ols/api/ontologies/{ontology_id}/terms
    ?size=8000'
    response = requests.get(ols_api_url)

    if response.status_code == 200:
        terms = response.json().get('
    _embedded', {}).get('terms', [])
        biomedical_entities = [term['label
    '] for term in terms]
        return biomedical_entities
    else:
        print(f"Error accessing OLS API.
    Status Code: {response.status_code}")
        return []
```

Matching biomedical entities with vectors: The function iterates over each biomedical entity and finds the closest match in the word-vectors dictionary using the get-close-matches function from the difflib module. If a match is found with a similarity score above the specified threshold, the biomedical entity along with its closest match word and corresponding vector are added to the matched-biomedical-entities dictionary.

```python
def match_biomedical_entities_with_vectors(word_vectors, biomedical_entities, threshold=0.8):
    matched_biomedical_entities = {}

    for entity in biomedical_entities:
        closest_matches = get_close_matches(entity, word_vectors.keys(), n=1, cutoff=threshold)
        if closest_matches:
            matched_biomedical_entities[entity] = {
                'word': closest_matches[0],
                'vector': word_vectors[closest_matches[0]]
            }

    return matched_biomedical_entities
```

Results Display:
To further explore the data of Matched biomedical entities, we performed cluster analysis and focused on visualizing the results in two-dimensional space.

(1)Application of t-SNE algorithm
First, we chose the t-SNE algorithm, which performs well in dimensionality reduction and visualization and helps preserve the local structure of the original data, to reduce the dimensionality of the word embedding results. After many debuggings, we selected perplexity=15. After focusing on retaining The local structure takes into account the global structure. Finally, we obtained the two-dimensional vector set ‘vectors-2d‘, which retains the important semantic information of the original high-dimensional vectors and lays the foundation for subsequent cluster analysis.

(2)Use the elbow rule to determine the optimal number of clusters
Use the KMeans algorithm to iterate from 1 to 10 clusters. For each cluster, use the KMeans algorithm to create a model with the parameters init=’k-means++’, max-iter=300, n-init=10, random-state=0 Then calculate the function ‘calculate-wcss‘, assuming that the center of the i-th cluster is C and the j-th data point is x, then the square distance within the cluster is the square of the Euclidean distance, that is:

$$WCSS_i = \sum_{j \text{ in cluster } i} \|\mathbf{x}_j - \mathbf{C}_i\|_2 \tag{4}$$

The total WCSS is the sum of the squared distances within each cluster:

$$WCSS = \sum_i WCSS_i \tag{5}$$

Iterates using the KMeans algorithm with different number of clusters, returning a list of WCSS values.

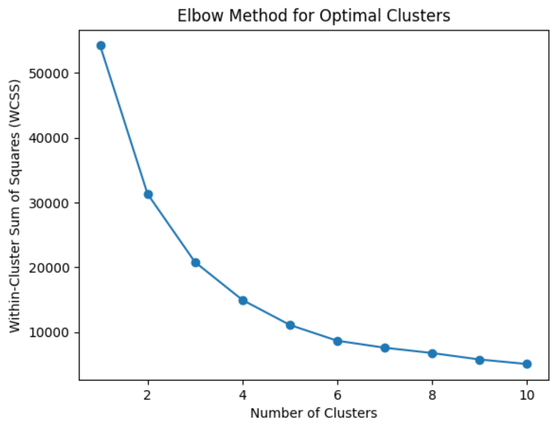


Fig. 12. Elbow Method for Optimal Clusters

Then we drew the elbow rule diagram, with the abscissa being the number of clusters and the ordinate being WCSS. By observing the graph, we selected the optimal number of clusters to be 4, which is the ”elbow point” of the elbow rule.

(3) Application of K-means clustering
Finally, we use the KMeans algorithm for clustering, using ’k-means++‘ to initialize the cluster centers, with the maximum number of iterations set to 300, ‘n-init=10‘ and ‘random-state=0‘. Through t-SNE visualization and K-means clustering, we successfully map high-dimensional word vectors into two-dimensional space and discover the underlying clustering structure in the data.

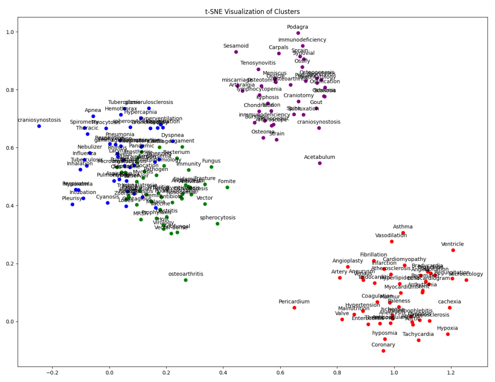


Fig. 13. t-SNE Visualization of Clusters

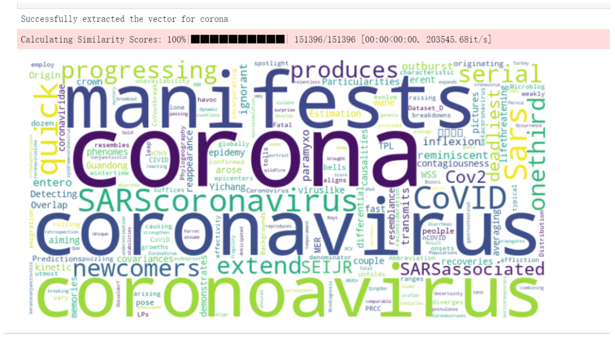
For example, we found that most of the blue points in the picture represent respiratory diseases, while most of the green points are related to infectious diseases. The two seem to be highly correlated in the visualization results, which may imply an indirect relationship between respiratory diseases and infectious diseases.

C. Co-occurrence

We try to find the biomedical entities which frequently co-occur with COVID-19. Using the Numpy and Scipy libraries, we manually constructed a co-occurrence matrix based on text data. In the code, the dok-matrix class is used to create a Sparse Matrix to more efficiently represent a matrix with most elements being zero, taking advantage of limited computing resources. Then, by traversing the text data and accumulating the corresponding positions of each pair of words in the text in the co-occurrence matrix, a co-occurrence matrix is constructed. Traverse the text data, collect all occurrences of words, and create a unique index for each word to locate the word's position in the co-occurrence matrix. After that, the co-occurrence matrix is filled, the text of each line is traversed, and for each pair of words in the text, the corresponding elements of the co-occurrence matrix are updated. Since it is an undirected graph, the matrix is updated symmetrically, ensuring that the co-occurrence relationships between words are recorded in both directions in the matrix. At the same time, we also normalize the values in the co-occurrence matrix, converting these raw frequencies into probabilities so that the sum of each row is 1. This helps to compare the co-occurrence probabilities of different words under different text lengths, allowing us to better understand the co-occurrence relationship between biomedical entities and COVID-19.

D. Semantic Similarity

Based on the word embedding method mentioned in the previous article, we find words that are semantically similar to the word COVID-19 by calculating the cosine similarity between word vectors. Cosine similarity is a commonly used vector similarity measurement method. Through the similarity of vectors, we can obtain the semantic similarity of natural languages. The formula of cosine similarity is similarity = dot-product / (norm-vec1 * norm-vec2), which measures the similarity of word vectors by calculating the dot product of two vectors and the normalized vector length. We used the cosine-similarity function to calculate and return the cosine similarity score, and then generated a word cloud diagram that is semantically similar to COVID-19 based on the score.



```
Successfully extracted the vector for corona
Calculating Similarity Scores: 100%|████████████| 151396/151396 [00:00<00:00, 203545.68it/s]
```

Fig. 14. word cloud figure

VII. Part5-Mining Biomedical Knowledge

A. Knowledge Graph Concept

What is called a knowledge graph (KG) is a model of representing knowledge as a diagram of interconnected nodes and edges, which is a collection of interlinked descriptions of concepts, entities, relationships, and events. Nodes represent entities and edges represent relationships between entities. For example, a node could be a patient and an agency such as a doctor. An edge would categorize the relationship as a doctor-patient relationship between them. KG put data in context via linking and semantic metadata and this way provides a framework for data integration, unification, analytics, and sharing. KG is benefiting the healthcare industry in many aspects and this part will provide a summary of three advantages.

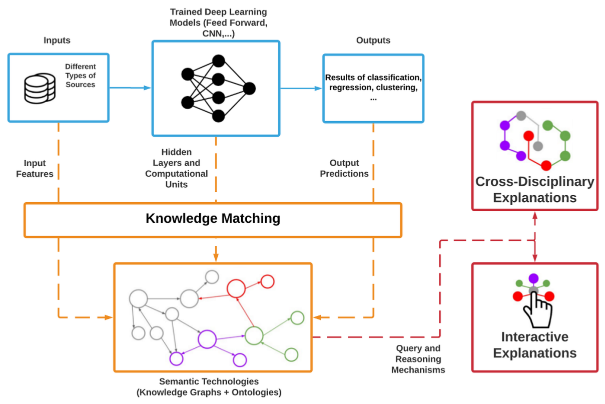

Inputs
Different Types of Sources
Trained Deep Learning Models (Feed Forward, CNN,...)
Outputs
Results of classification, regression, clustering, ...
Input Features
Hidden Layers and Computational Units
Output Predictions
Knowledge Matching
Cross-Disciplinary Explanations
Interactive Explanations
Query and Reasoning Mechanisms
Semantic Technologies (Knowledge Graphs + Ontologies)

Fig. 15. Knowledge Graph

B. Implementation Approach

We extracted 25 words from the biomedical entities extracted from Track4 and their relevance to COVID-19 to construct the knowledge graph. Specifically, we utilized **Neo4j Workspace** to generate the knowledge graph online. In the construction of various node relationships. We plan to use Cypher queries to create a knowledge graph about the field of medical research and COVID-19. A brief description of the relationships built is as follows:

Epidemiology association: Describes the relationship between epidemiology and COVID-19, specifically the association between the field of epidemiology and the COVID-19 disease.

Pulmonology association: Describes the relationship between pulmonology and COVID-19-related symptoms, including dyspnea, cough, fever, sore throat, shortness of breath, fatigue, and loss of taste or smell.

Immunology association: Describes the relationship between immunology and COVID-19, focusing on the association between the field of immunology and the COVID-19 disease.

Treatment association: Describes the relationship between different treatment methods and COVID-19, including anti-inflammatory drugs, antiviral medications, oxygen therapy, ventilator support, anticoagulants, steroids, monoclonal antibodies, mechanical ventilation, etc.

Association between symptoms: Describes the relationship between COVID-19-related symptoms, including dyspnea, cough, fever, sore throat, shortness of breath, fatigue, and loss of taste or smell.

However, due to our unfamiliarity with Cypher, we failed to generate the relationship connectors that meet the above requirements within the specified time frame, and only created independent categorical nodes, as shown in the figure below:

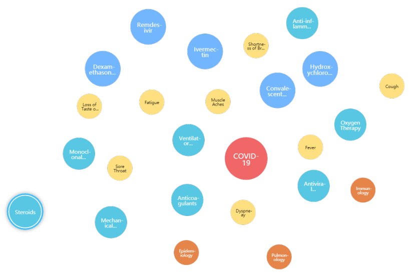
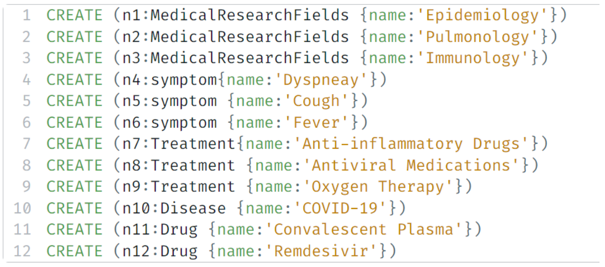


Fig. 16. Knowledge Graph figure generated



```
CREATE (n1:MedicalResearchFields {name:'Epidemiology'})
CREATE (n2:MedicalResearchFields {name:'Pulmonology'})
CREATE (n3:MedicalResearchFields {name:'Immunology'})
CREATE (n4:symptom{name:'Dyspneay'})
CREATE (n5:symptom {name:'Cough'})
CREATE (n6:symptom {name:'Fever'})
CREATE (n7:Treatment{name:'Anti-inflammatory Drugs'})
CREATE (n8:Treatment {name:'Antiviral Medications'})
CREATE (n9:Treatment {name:'Oxygen Therapy'})
CREATE (n10:Disease {name:'COVID-19'})
CREATE (n11:Drug {name:'Convalescent Plasma'})
CREATE (n12:Drug {name:'Remdesivir'})
```

Fig. 17. Screenshot of the code

VIII. Conclusion

This project aims to segment, vectorize, and visually cluster semantic recognition of large text data. We constructed and trained our own N-gram and Skip-gram models, and utilized pre-trained BERT models from others to achieve word vector representation. Finally, we used the t-SNE algorithm to visualize dimensionality reduction, implemented semantic recognition through clustering algorithms. Lastly, we attempted to represent COVID-19 and its semantically related biomedical entities using knowledge graphs.

In prospect, several avenues for future exploration and enhancement present themselves. Primarily, there is scope for further refinement of our segmentation and vectorization models to yield more precise and semantically rich representations of textual data. Furthermore, the exploration of diverse clustering algorithms or the amalgamation of multiple methodologies holds promise for augmenting the semantic recognition prowess of our system. Moreover, the integration of supplementary contextual information or domain-specific knowledge into our models has the potential to bolster their efficacy, particularly in domains characterized by specialized terminology such as biomedical research. By focusing on these areas, we can continue to advance the capabilities of our NLP system and contribute to the ongoing progress in the field.

References

[1] European Bioinformatics Institute. Ontology Lookup Service (OLS). https://www.ebi.ac.uk/ols/ontologies/hp. Accessed on: February 4, 2024.

[2] stopwords-iso. English stopwords collection. https://github.com/stopwords-iso/stopwords-en. Accessed: February 4, 2024.

[3] S Suyanto, S Andi, R I Nafi, et al. Augmented-syllabification of n-gram tagger for indonesian words and named-entities. *Heliyon*, 8(11), 2022.

[4] Lucy Lu Wang and Kenneth Lo. Cord-19: The covid-19 open research dataset. https://www.semanticscholar.org/cord19/download, 2020.

[5] J H Wei, W J Jun, G C Bang, et al. Enhancing n-gram based metrics with semantics for better evaluation of abstractive text summarization. *Journal of Computer Science and Technology*, 37(5), 2022.

[6] T Yachun. Research on word vector training method based on improved skip-gram algorithm. *Advances in Multimedia*, 2022.

[7] J Zhang, J Yuan, J Zhang, et al. Multi-meta information embedding enhanced bert for chinese mechanics entity recognition. *Applied Sciences*, 13(20), 2023.