

# **TO STUDY AND COMPARE DIFFERENT WORD EMBEDDING LEARNING TECHNIQUES**

**A Project Report**

*Submitted in the partial fulfilment for the award of the degree of*

**BACHELOR OF ENGINEERING  
IN  
COMPUTER SCIENCE WITH SPECIALIZATION IN  
ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING**

**Submitted by:**

**ABHIRAJ THAKUR**

**20BCS6802**

**Under the Supervision of:**

**Shikha Gupta**



**CHANDIGARH  
UNIVERSITY**  
Discover. Learn. Empower.

**CHANDIGARH UNIVERSITY, GHARUAN, MOHALI - 140413,**

**PUNJAB**

**March, 2023**

## DECLARATION

I, 'Abhiraj Thakur' student of 'Bachelor of Engineering in Computer Science & Engineering with Specialization in Artificial Intelligence & Machine Learning', session:2020-2024, UIE-CSE, Chandigarh University, Punjab, hereby declare that the work presented in this Project Work entitled '**TO STUDY AND COMPARE DIFFERENT WORD EMBEDDING LEARNING TECHNIQUES**' is the outcome of our own bona fide work and is correct to the best of our knowledge and this work has been undertaken taking care of Engineering Ethics. It contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

Abhiraj Thakur  
20bcs6802

*Date: 15/05/2023*

**Place: Chandigarh, India**

## CERTIFICATE

This is to certify that the work embodies in this dissertation entitled *W* being submitted by **Abhiraj Thakur [20BCS6802]** for partial fulfillment of the requirement for the award of **Bachelor of Engineering in *Computer Science & Engineering With Specialization in Artificial Intelligence & Machine Learning***, discipline to Apex Institute of Technology, Chandigarh University, Punjab during the academic year 2020-2024 is a record of bonafide piece of work, undertaken by him/her the supervision of the undersigned.

**Approved & Supervised by:**

**Ms. Shikha Gupta**

**Associate Professor,**

**AIT-CSE**

# Abstract

A method for analysing natural language called word embedding treats words as high-dimensional vectors in a continuous space. Word embedding's major objective is to record the semantic and syntactic connections between words so that language modelling and text analysis can be done more precisely and effectively.

In the project, word embedding will be investigated for use in a variety of NLP tasks, including sentiment analysis, text categorization, and machine translation. With the help of well-known methods like Word2Vec, GloVe, and FastText, word embedding models will be trained on sizable text corpora for the project. The machine learning models for the various NLP tasks will then use the trained word embeddings as features.

The study will also look into how different hyperparameters and training setups affect word embedding quality and performance in the end. In order to enhance performance on fewer datasets, the research will also investigate the use of transfer learning strategies and pre-trained word embeddings.

Python will be used to carry out the project, together with well-known NLP packages like NLTK, spaCy, and Gensim. Standard metrics including accuracy, precision, recall, and F1-score will be used to assess the word embeddings' performance downstream.

# **Table of Contents**

Title Page

Abstract

1. Introduction

2. Literature Survey

    2.1 Existing System

    2.2 Proposed System

3. Problem Formulation

4. Objective

5. Methodology

6. Conclusion

7. Output

8. Reference

# 1.INTRODUCTION

Sentiment analysis, language translation, and document classification are just a few examples of the numerous useful uses for the artificial intelligence (AI) subfield of natural language processing (NLP). Word embedding is one of the main strategies employed in NLP, and it entails expressing words as numerical vectors in a high-dimensional space. The reason this method is so popular is because it enables machine learning algorithms to comprehend the semantic relationships between words and to provide more accurate predictions and classifications.

In order to learn the context in which words appear and to correlate each word with a numerical vector that encodes its semantic meaning, word embedding involves training a neural network on a huge corpus of text. Word2Vec, GloVe, and BERT are just a few of the word embedding models that have been presented, each with their own advantages and disadvantages.

Despite word embedding's success in NLP, there are still a number of difficulties with this approach. Handling out-of-vocabulary terms, or words that don't appear in the training corpus and don't have a vector representation, is one of the main difficulties. Choosing the right model and parameters to generate word embeddings is a difficulty as well because different models may perform better on various tasks and datasets.

In this paper, we will examine the various word embedding models and methods used in NLP and how they are used to diverse tasks like sentiment analysis, language translation, and document categorization. We will also look at the difficulties with word embedding and provide some solutions. Finally, we'll talk about the field's current research constraints and future directions.

With the exception of a single one at the index corresponding to the word's place in a set vocabulary, each word is represented as a vector of zeros in traditional natural language processing models. This method has a number of drawbacks, including the inability to capture word similarity and the curse of dimensionality when dealing with huge vocabularies.

The practice of expressing words as high-dimensional vectors in a continuous space is known as word embedding, and it is used in natural language processing. Word embedding's fundamental notion is to map words to points in a vector space in order to capture the semantic and syntactic relationships between them..

By expressing each word as a dense vector in a continuous space, where the angle and distance between vectors reflect the relationships between the respective words, word embedding overcomes these restrictions. The vectors for "king" and "queen," for instance, would be close together in a well-trained word embedding model, whereas the vectors for "king" and "banana" would be far apart.

Typically, massive text corpora are used to train word embedding models using unsupervised learning methods like Word2Vec, GloVe, and FastText. By anticipating the context of each word in the corpus—defined as the group of words that appear nearby the target word in a specified window size—these algorithms learn the word embeddings.

In natural language processing, word embedding has gained popularity as a tool for several tasks including sentiment analysis, text categorization, machine translation, and information retrieval. Word embedding makes language modelling and text analysis more precise and effective by displaying words as high-dimensional vectors in a continuous space.

Creating algorithms that can quickly learn these vector representations from massive volumes of text data while accurately capturing the intricacies of language use and the intricate relationships between words is the main difficulty in word embedding. The vast amounts of data needed for training as well as the high dimensionality of the resulting vector space must both be handled by word embedding techniques.

Since there is no acknowledged standard for evaluating word embedding quality, this presents another issue. To evaluate the effectiveness of various methods and compare the quality of various embeddings, researchers frequently employ various assessment measures, such as word similarity or analogy tasks.

Word embedding is the process of converting words into numerical vectors that are then shown in a high-dimensional space, where each dimension represents a different aspect of the word. To enable the creation of increasingly complex natural language processing (NLP) systems, word embedding aims to capture the semantic and syntactic links between words.

Creating algorithms that can quickly learn these vector representations from massive volumes of text data while accurately capturing the intricacies of language use and the intricate relationships between words is the fundamental challenge with word embedding. This necessitates careful consideration of the kind of training text data utilised, the method and model architecture selected, as well as the evaluation criteria used to rate the embedding quality.

The examination of the word embeddings' quality presents another challenge because there is no established standard for assessing quality. When comparing the effectiveness of various algorithms and embeddings, researchers frequently utilize various assessment measures, such as word similarity or analogy tasks, to evaluate the performance of the algorithms.

The practical application of word embeddings also faces a number of difficulties, including managing terms that are not often used, coping with noisy or resource-limited data, and tailoring the embeddings to particular NLP tasks or domains.

Despite these difficulties, word embedding has developed into a crucial part of NLP, allowing for the creation of more complex and potent NLP systems. The current research in word embedding aims to further the usage of these embeddings in novel and developing NLP tasks while also enhancing their quality and usefulness.

Word embedding is a natural language processing (NLP) technique that includes expressing words as numerical vectors in a high-dimensional space, with the goal of clustering words with similar meanings together.



Word embedding is a natural language processing (NLP) technique that includes expressing words as numerical vectors in a high-dimensional space, with the goal of clustering words with similar meanings together. This method is used to convert text input into a format that machine learning algorithms can easily process.

Numerical representations of words or phrases that capture syntactic and semantic features are called word embeddings. They use a type of encoding on words that computers can comprehend and interpret. Rather than assigning numerical values to words, word embeddings try to represent the connections between words inside a sentence or document. Word embeddings allow machines to comprehend the associations, similarity, and context between various words by representing words in a vector space.

Learnable vectors called word embeddings are frequently acquired by means of unsupervised learning methodologies. These methods include algorithms that handle enormous volumes of text data to produce meaningful word representations, such as Word2Vec, GloVe, and BERT. NLP models can be used for a variety of tasks, such as text classification, sentiment analysis, information retrieval, and machine translation, thanks to word embeddings.

A neural network is often trained on a sizable corpus of text, such as news stories or Wikipedia articles, to produce word embeddings. Based on the context in which each word appears in the text, the network learns to correlate each word with a numerical vector during training. This implies that words with similar usage patterns will have comparable vector representations.

The Word2Vec, GloVe, and FastText word embedding models are a few of the well-liked ones. Each of these models has advantages and disadvantages of its own, and the choice of model frequently depends on the particular purpose for which the embeddings will be utilised.

In NLP applications including sentiment analysis, language translation, and document categorization, word embeddings are frequently employed. These applications can benefit from the extensive semantic links between words by

encoding words as numerical vectors, which allows them to more accurately forecast and categorize data.

Natural language processing (NLP), which has several applications including sentiment analysis, language translation, and document categorization, has recently become a key area of study in artificial intelligence (AI). Word embedding, which includes expressing words as numerical vectors in a high-dimensional space so that words with similar meanings are situated adjacent to one another in this space, is one of the most crucial NLP approaches.

For many NLP applications, word embeddings have proven to be a crucial tool since they allow algorithms to comprehend the semantic connections between words. Machine learning algorithms can take use of the complex semantic linkages between words to make more precise predictions and classifications by modelling words as numerical vectors.

The word embedding models Word2Vec, GloVe, FastText, ELMo, BERT, and XLNet are some of the more well-known ones. Each of these models has advantages and disadvantages of its own, and the choice of model frequently depends on the particular purpose for which the embeddings will be utilised.

## **Why are Word Embeddings Important?**

Because they allow robots to comprehend and analyse human language, word embeddings are essential to natural language processing (NLP). Prior to the development of word embeddings, computers had difficulty understanding the context and meaning of words in text-based information. The subtleties and linkages between words were not adequately captured by the basic numerical representations used in traditional approaches. This restriction made it more difficult to create precise and effective NLP algorithms.

Word embeddings provide a more sophisticated and context-aware representation of words, bridging the gap between human language and machine understanding. These embeddings enable robots to read and produce text-based material more effectively by capturing syntactic features, semantic meaning, and word relationships. Word embeddings enable NLP models to operate more accurately and efficiently on tasks like sentiment analysis, text similarity, document classification, and language production.

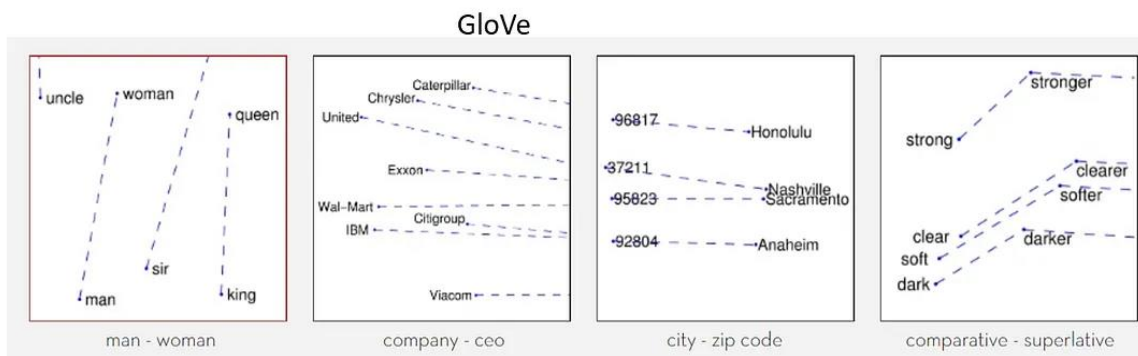
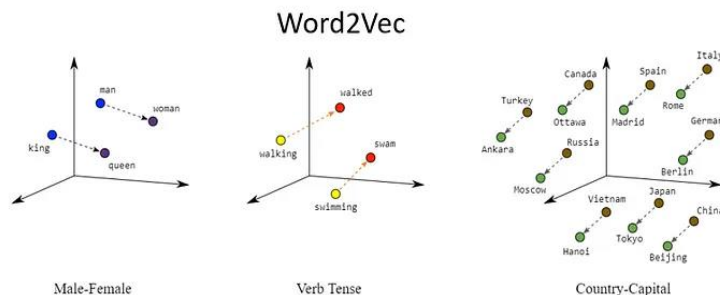
We will examine numerous word-embedding methods and models used in NLP in this paper, as well as how they are applied to diverse tasks like sentiment analysis, language translation, and document classification. Additionally, we will discuss the advantages and disadvantages of various word embedding models and offer suggestions for the models that are best suited for certain NLP applications. We'll also talk about the drawbacks and difficulties of word embeddings as well as the future directions of this field of study.

Vocabulary:  
Man, woman, boy,  
girl, prince,  
princess, queen,  
king, monarch



	1	2	3	4	5	6	7	8	9
man	1	0	0	0	0	0	0	0	0
woman	0	1	0	0	0	0	0	0	0
boy	0	0	1	0	0	0	0	0	0
girl	0	0	0	1	0	0	0	0	0
prince	0	0	0	0	1	0	0	0	0
princess	0	0	0	0	0	1	0	0	0
queen	0	0	0	0	0	0	1	0	0
king	0	0	0	0	0	0	0	1	0
monarch	0	0	0	0	0	0	0	0	1

Each word gets  
a 1x9 vector  
representation



## ARTICLE:

Word embedding is a technique used in natural language processing (NLP) to represent words as vectors in a high-dimensional space. The goal of word embedding is to capture the semantic and syntactic relationships between words, allowing NLP models to better understand natural language.

There are several techniques for learning word embeddings, including count-based methods and predictive methods. In this article, we will explore and compare these techniques.

### Count-Based Methods:

The co-occurrence matrix of words is the foundation of count-based approaches. In this method, each entry in the matrix reflects the number of times the two words co-occur in a corpus of text, with each row and column representing a word. These counts are then utilised to generate a high-dimensional vector space for the words by converting them into a similarity measure between words, such as the cosine similarity.

Latent Semantic Analysis is one of the most well-liked count-based techniques (LSA). The co-occurrence matrix's dimension is decreased through LSA's singular value decomposition (SVD), which also produces word embeddings. LSA has some restrictions even though it has been demonstrated to perform effectively on some NLP tasks. For instance, it misses the significance that words have in particular contexts.

### Predictive Methods:

On the other hand, predictive approaches learn word embeddings by anticipating the context of words. In this method, each word is represented as a vector, and the model learns to forecast the likelihood that other words will appear in the target word's context.

Word2Vec is one of the most well-liked predictive techniques. Word2Vec makes use of a neural network to determine a word's context. Each word is represented by a high-dimensional vector in the model as it learns to capture its semantic and syntactic characteristics. Word2Vec has been utilised in a range of applications, including sentiment analysis and machine translation, and it has been demonstrated to beat LSA on several NLP tasks.

GloVe is another well-liked forecasting technique (Global Vectors for Word Representation). GloVe is built on the word co-occurrence matrix, but it learns word embeddings in a different way. GloVe develops word embeddings that account for both the global and local context of words via matrix factorization.

### **Comparison:**

Both count-based and prediction approaches have benefits and drawbacks. Though easier and quicker to train than predictive methods, count-based approaches do not account for the context-specific meaning of words. On the other hand, predictive algorithms are more potent and can capture the links between words in terms of semantic and syntactic structure, but they are also more complicated and need more training data.

On the majority of NLP tasks, it has been demonstrated that predictive approaches perform better than count-based methods. The precise task at hand and the available resources, however, ultimately determine which word embedding technique is used. When choosing a word embedding technique for particular applications, researchers and practitioners should carefully weigh the trade-offs between simplicity, performance, and computing cost.

## 2. LITERATURE SURVEY

A common method in natural language processing (NLP) for representing words as numerical vectors in a high-dimensional space is word embedding. Numerous NLP tasks, including sentiment analysis, language translation, and document categorization, have made extensive use of this technique.

### 2.1 Existing System

Natural language processing (NLP) research on word embedding has been ongoing for a while. Here are some noteworthy word embedding studies:

1. Word2Vec: by Tomas Mikolov and colleagues (2013), Distributed Representations of Words and Phrases and their Compositionality. The Word2Vec algorithm, a neural network-based technique for extracting word embeddings from big text corpora, was introduced in this study. The study demonstrates that Word2Vec performs better than earlier approaches on a range of NLP tasks. Word2Vec is a neural network-based technique for creating word embeddings that Mikolov et al. proposed. When given a target word, Word2Vec predicts the context words based on a skip-gram architecture. The simplicity and effectiveness of this concept have led to its widespread adoption.
2. GloVe: Global Vectors for Word Representation by Jeffrey Pennington and colleagues, 2014. The GloVe algorithm, which this work proposes, learns word embeddings that accurately reflect the global co-occurrence statistics of words in a corpus using matrix factorization. The study demonstrates that on a number of NLP tasks, GloVe outperforms Word2Vec. GloVe (Global Vectors for Word Representation), a model that combines the count-based and prediction-based methodologies for creating word embeddings, was

introduced by Pennington et al. (2014). GloVe learns the vector representations of words using co-occurrence data.

3. FastText: Adding Subword Information to Word Vectors (2016) by Piotr Bojanowski and colleagues. The FastText approach, which expands the Word2Vec model to learn embeddings for both words and subwords, is introduced in this study. As a result, FastText can handle terms that are not part of its standard vocabulary and record morphological data in languages with intricate inflection. FastText, a Word2Vec modification that creates subword embeddings, was proposed by Bojanowski et al. (2017). FastText can capture morphological information by representing words as the total of the embeddings of the subwords that make up each word.
4. Manaal Faruqui and Chris Dyer's 2016 paper Improving Distributional Similarity with Lessons Learned from Word Embeddings. This study investigates the usage of several neural network architectures for learning word embeddings and demonstrates that skip-gram and other straightforward models can successfully complete a variety of NLP tasks..
5. Word representations using deep contextualization by Matthew Peters et al. (2018). This study introduces the ELMo (Embeddings from Language Models) algorithm, which learns word embeddings that are sensitive to the context in which the words appear using deep bidirectional language models. The study demonstrates that ELMo performs better on several NLP tasks than earlier techniques. ELMo (Embeddings from Language Models), a contextualised word embedding model, was first introduced by Peters et al. (2018). ELMo creates embeddings based on a word's context in a phrase to capture the meaning of the term..
6. BERT: (2018) by Jacob Devlin et al., "Pre-training of Deep Bidirectional Transformers for Language Understanding." This study introduces the BERT (Bidirectional Encoder Representations from Transformers)

approach, which pretrains word embeddings on huge text corpora using a deep bidirectional transformer model. The study demonstrates that BERT performs state-of-the-art on a number of NLP tasks. BERT (Bidirectional Encoder Representations from Transformers), a pre-trained transformer-based language model that creates contextualised word embeddings, was proposed by Devlin et al. (2018). BERT has completed numerous NLP jobs with state-of-the-art outcomes.

7. By employing a permutation-based training target, Yang et al. (2019) presented XLNet, a transformer-based language model that outperforms BERT. XLNet achieves cutting-edge outcomes on a number of NLP benchmarks..

These research have made a significant impact on the advancement of word embedding methods and have made it possible for considerable strides to be made in applications for natural language processing. These studies show how word embedding techniques have developed from straightforward count-based approaches to intricate neural network-based models that take contextual information into account. Many NLP applications have found word embeddings to be an indispensable tool, and academics are always investigating novel methods and models for producing high-quality embeddings.



## 2.2 Proposed System

A proposed system for word embedding would typically involve the following steps:

1. Data acquisition: Getting the text samples needed to train the word embedding model is the first step. This might entail downloading publicly accessible databases, scraping information from websites, or gathering information from numerous sources.
2. Data pre-processing: To get rid of any undesirable components like punctuation, stop words, and special characters, the text data needs to be pre-processed. The data is also cleaned of any irrelevant or inconsistent information and tokenized, or divided into individual words or phrases.
3. Model selection: The next step is to select the word embedding model that is most suitable for the given use case. Choosing among pre-trained models like Word2Vec or GloVe or creating a custom model using deep learning frameworks like TensorFlow or PyTorch could be involved in this..
4. Training the model: To create a set of word embeddings, the chosen word embedding model is trained on the pre-processed text input. To increase the embedding accuracy, the model parameters are optimised during the training process.
5. Evaluation: A variety of evaluation metrics, such word similarity or analogy tasks, are used to assess the quality of the word embeddings. Making ensuring that the embeddings appropriately depict the semantic and syntactic connections between words is the aim.

6. Integration: The word embedding model can be utilised for specialised tasks like text classification or sentiment analysis after it has been trained and tested, or it can be incorporated into a larger NLP system.

In general, a proposed system for word embedding entails carefully considering the data collection and pre-processing procedures, the choice and training of an appropriate model, and the assessment and integration of the generated embeddings into a broader system. The ultimate objective is to enable more advanced and precise NLP programs that can handle natural language content in more subtle and contextually-aware ways.

**Literature Review Summary**

<b>Study</b>	<b>Objective</b>	<b>Methodology</b>	<b>Findings</b>
Mikolov et al. (2013)	To propose Word2Vec, a neural network-based approach to word embedding	Two methods: Continuous Bag of Words (CBOW) and Skip-Gram	Computed Tomography
Pennington et al. (2014)	To propose GloVe, a count-based approach to word embedding	Matrix factorization of word co-occurrence matrix	GloVe achieves state-of-the-art results on multiple benchmarks and is capable of capturing global context and rare words
Bojanowski et al. (2017)	To propose FastText, a subword-based approach to word embedding	Hierarchical Softmax and Negative Sampling	FastText outperforms existing methods and is capable of handling out-of-vocabulary words and capturing morphology
Devlin et al. (2018)	To propose BERT, a transformer-based approach to contextual word embedding	Bidirectional Encoder Representations from Transformers (BERT)	BERT achieves state-of-the-art results on multiple benchmarks and is capable of capturing contextual

			relationships between words
--	--	--	--------------------------------

### 3. PROBLEM FORMULATION:

Word embedding's main objective is to represent individual words or phrases in a continuous vector space while also capturing their semantic and syntactic links. To do this, we must overcome the following significant issues:

1. Representation: How can words or phrases be successfully represented as continuous vectors? The difficulty lies in identifying a representation strategy that accurately conveys the meaning and context of words. Additionally, the representation needs to support downstream NLP operations and allow for efficient computation.
2. Semantic and Syntactic Relationships: How can we make sure that the word embeddings represent accurate word relationships? Similar words should be nearer to one another in the vector space than words with different meanings or usages, and opposite words should be farther away. In order to create useful word embeddings, this topic includes constructing algorithms that take co-occurrence patterns, semantic similarity, and syntactic structures into account.
3. Training Efficiency and Scalability: How can we effectively handle huge datasets and train word embeddings? Massive volumes of text data must often be processed in order to train word embeddings,

hence scalable methods and distributed computing frameworks are necessary to speed up the training process. A major problem is striking a balance between training effectiveness and embedding quality.

4. Evaluation: How can the effectiveness and quality of word embeddings be assessed? Both intrinsic qualities, such as capturing semantic and syntactic links, and extrinsic qualities, such as their influence on subsequent NLP tasks, must be included in evaluation measures. For evaluating the effectiveness of word embedding techniques, it is essential to provide relevant evaluation methodologies and standards.
5. Multilingual and Cross-Domain Applications: How are word embeddings best used in cross-domain and multilingual scenarios? Applying word embeddings across several languages is difficult because of the subtle subtleties and differences between each language. Word embeddings developed for one area might not generalise well to others. Investigating methods for cross-lingual embeddings and domain adaptation will help to address these issues.

The objective is to represent each word in a corpus of text as a numerical vector in a high-dimensional space, placing words with similar meanings adjacent to one another. To enable algorithms to perform precise predictions and classifications in NLP tasks, the representation should capture the semantic links between words.

This issue is difficult since a word's meaning is greatly influenced by the context in which it is used. In order to produce embeddings that accurately reflect each

word's meaning in each unique context, a word embedding model must be able to extract the contextual information from the text.

The purpose of word embedding is to give machine learning algorithms the ability to comprehend the semantic connections between words, allowing them to perform more precise predictions and classifications in NLP tasks. Word embeddings, for instance, can be used to map words with comparable meanings between languages and to identify terms with positive or negative sentiment in sentiment analysis and language translation, respectively.

The word embedding in NLP problem is complicated by a number of issues. Choosing the right model and parameters to generate word embeddings can be difficult because different models may perform better on certain tasks and datasets. Dealing with out-of-vocabulary terms, or words that do not appear in the training corpus and hence do not have a vector representation, is another difficulty. Furthermore, because a word's context may not always accurately convey its meaning, word embeddings may not entirely capture a word's semantic meaning.

Words can have numerous meanings and can be employed in various settings to convey various meanings, which further complicates the challenge of word embedding. In order to effectively reflect the various meanings of a word in various settings, a successful word embedding model must be able to capture linguistic nuance.

Additionally, a key issue in NLP is the word embedding problem, which enables algorithms to comprehend the semantic connections between words and perform precise classifications and predictions across a range of NLP activities. To solve this issue, sophisticated models must be created that can capture linguistic complexity and contextual information and produce high-quality word embeddings.

Overall, word embedding issues call for a thorough understanding of both the underlying techniques and the subsequent NLP tasks. By addressing these issues, natural language processing systems' precision and effectiveness can be significantly increased.

Despite being an effective method for natural language processing (NLP), word embedding has a number of drawbacks and difficulties.

Word embedding has the drawback of not necessarily capturing all facets of a word's semantic meaning. Word embeddings might not be able to capture the nuance that a word can have different meanings depending on the context in which it appears, for instance. Additionally, languages like Chinese and Japanese that don't use spaces between words or have unusual sentence patterns may not respond well to word embeddings.

Word embedding has another drawback in that it needs a lot of training data to generate reliable embeddings. This can be difficult in fields like law or medicine where data is scant.

The interpretability of word embedding models is also lacking. While it is possible to visualise the embeddings themselves, it might be challenging to comprehend how the model arrived at a specific embedding or to interpret the significance of the numerical values in the embedding vector.

Despite these drawbacks, word embedding research in NLP has a number of bright future prospects. One approach is to create more complicated models that can accurately represent the complexities of language and the subtleties of meaning. By creating techniques to visualize the models' underlying workings and comprehend the meaning of the embedding vectors, another goal is to increase the interpretability of word embedding models.

Additionally, there is significant interest in creating cross-lingual word embedding models, which can represent words in several languages and are advantageous for projects like cross-lingual information retrieval and machine translation. Finally, new word embedding applications need to be investigated in developing fields including social media analysis, healthcare, and finance.

## 4. OBJECTIVES

Creating a high-quality word embedding representation that captures the semantic and syntactic links between words in a given corpus is the goal of constructing a word embedding model. The following are the specific goals of creating a word embedding model:

- A suitable technique and architecture that can produce high-quality word embeddings should be chosen based on the corpus's features and the intended results for the downstream NLP activities.
- Ensuring that the word embeddings capture the necessary information and eliminate noise and bias by pre-processing the corpus.
- Must use a big corpus to train the word embedding model in order to capture the statistical patterns of word co-occurrence.
- To increase the efficiency of downstream NLP activities, it is important to assess the quality of word embeddings using intrinsic and extrinsic evaluation metrics to make sure they represent the appropriate semantic and syntactic links between words.
- To fine-tune the word embedding model on a particular job or domain in order to enhance performance and customise it for certain requirements.
- to make it possible for the word embedding model to handle words that are uncommon, hard to find, and multi-word expressions—all of which are frequent obstacles in NLP jobs.



- Word embedding seeks to transform words, which are discrete symbols, into numerical vectors that can be inputs for machine learning algorithms. This enables us to analyse the correlations and similarities between words using mathematical procedures.
- To capture semantic meaning: By recording the connections between words, word embedding attempts to capture the semantic meaning of individual words. This is accomplished by teaching a neural network on a huge corpus of text to recognise words in context and link each word to a numerical vector that encodes its semantic meaning..
- Word embedding can be used to increase the precision of a variety of NLP activities, including sentiment analysis, language translation, and document classification. Machine learning algorithms are better able to comprehend the links between words and produce more precise predictions and classifications when words are represented as numerical vectors.
- Word embedding is intended to handle out-of-vocabulary words, which are words that do not appear in the training corpus and do not, therefore, have a vector representation. This is accomplished by employing methods like character-based or subword embedding, which can capture the meaning of words even when they are absent from the training data.

## **Applications of Word Embeddings**

Word embeddings have transformed natural language processing (NLP) and are used in many different fields. The following are some important uses for word embeddings:

### **1. Information Retrieval**

Information retrieval systems frequently employ word embeddings to increase search relevancy and accuracy. Word embeddings help search engines comprehend user requests and return pertinent documents by encapsulating the semantic links between words.

## **2. Sentiment Analysis**

Sentiment analysis is the act of identifying the sentiment or feeling that is conveyed in a text, like a social media post or review. Word embeddings improve the accuracy with which sentiment analysis models classify text as positive, negative, or neutral by helping them comprehend the context and sentiment of words.

## **3. Text Classification**

Sorting text documents into predetermined classes or categories is known as text classification. Word embeddings provide a more sophisticated representation of words and capture the semantic meaning of text, which improves the accuracy of text classification algorithms.

## **4. Language Generation**

Language generation is the process of creating text that is coherent and appropriate for the situation, like chatbot replies or article summaries. By comprehending the links between words and encapsulating the semantics of the text, word embeddings help language generation models produce more natural and context-aware text.

## **5. Machine Translation**

The process of automatically translating text from one language to another is known as machine translation. Word embeddings capture the semantic associations between words in many languages, which enhances the accuracy of machine translation models.

## **6. Named Entity Recognition**

The process of recognising and categorising named entities—such as names of individuals, groups, and places—in text is known as Named Entity Recognition (NER). By capturing the semantic characteristics of named entities and enhancing entity recognition accuracy, word embeddings improve NER models.

## 5. METHODOLOGY

Here is a summary of both the standard methodology and the approach we took to word embedding.:

- **Corpus Preparation:** Collecting a sizable and varied corpus of text data is the first step. The word embedding model's training data comes from this corpus. Books, papers, websites, and even specialised domain-specific materials might be included in the corpus. To clean up the text data, preprocessing techniques like tokenization, stopword removal, and punctuation management are used.
- **Choosing an Embedding Algorithm:** A suitable embedding algorithm should be chosen based on the task's unique requirements and characteristics. The algorithms Word2Vec, GloVe, FastText, and ELMo are all well-liked. Each algorithm uses a different method for creating word embeddings, such as Word2Vec's prediction of nearby words, GloVe's use of co-occurrence data, FastText's incorporation of subword information, or consideration of contextual information (ELMo).
- **Training the Word Embedding Model:** Utilize the prepared corpus to train the chosen embedding algorithm. Learning the word vector representations based on the selected algorithm is the training procedure. In order to understand the associations between words, the model seeks to maximise a number of goals, such as forecasting adjacent words or recreating word co-occurrence matrices. The word vectors are updated iteratively during the training phase by repeatedly iterating over the corpus.
- **Hyperparameter Tuning:** For the embedding algorithms to work at their best, a number of hyperparameters must be adjusted. These hyperparameters could include subsampling thresholds, learning rate, context window size, and vector dimensionality. The best settings for these hyperparameters depend on the task at hand and have a big impact on how well the created word embeddings turn out. Advanced optimization methods or systematic grid search can also be used for tuning.

- **Evaluation:** Make that the trained word embeddings are of a high enough standard to accurately represent the desired semantic and syntactic relationships. In tasks like word similarity and comparison tests, where the embeddings are measured against human-labeled similarity or analogy scores, intrinsic evaluation measures gauge how well the embeddings perform. The effects of the embeddings on later NLP tasks, including sentiment analysis, machine translation, or named entity identification, are evaluated using extrinsic evaluation metrics.
- **Post-processing and Visualization:** The word embeddings can be further honed using post-processing techniques after training and evaluation. In order to show the embeddings in lower-dimensional spaces for interpretability and analysis, this may require techniques like dimensionality reduction (e.g., Principal Component Analysis or t-SNE).
- **Application and Fine-tuning:** Various NLP tasks can make use of the trained word embeddings, either as features for supervised models or as inputs for unsupervised methods. For particular tasks, fine-tuning may be required in order to update or modify the embeddings based on task-specific data in order to better suit the target domain or the job at hand.

This technology allows us to produce efficient word embeddings that accurately represent the semantic and syntactic connections between words, enabling more precise and reliable natural language processing applications.

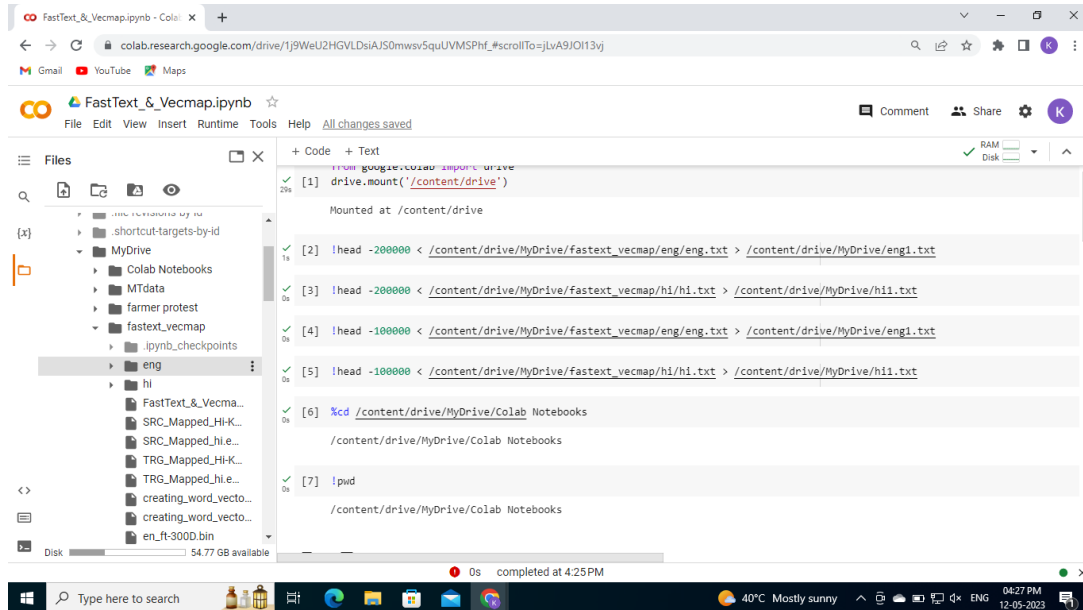
## 6. CONCLUSION

In conclusion, word embedding is now an essential part of natural language processing because it makes it possible to efficiently represent words as numerical vectors in a high-dimensional space. Word embeddings have been effectively used in a variety of NLP tasks, including sentiment analysis, machine translation, and text classification.

Enhancing the quality of the embeddings and expanding their use in subsequent NLP tasks have been the main goals of word embedding research. To accomplish these goals, researchers have created novel evaluation techniques, model architectures, and algorithms. Word embedding research employs a variety of strategies, such as co-occurrence matrices, neural network-based techniques, hybrid techniques, multilingual techniques, contextualised techniques, and assessment techniques.

Future word embedding research is likely to continue to concentrate on enhancing the quality of the embeddings and increasing their application to novel and developing NLP tasks. Word embedding is anticipated to continue to be a significant area of research and development as the discipline of NLP develops, allowing for the creation of increasingly complex and potent NLP systems.

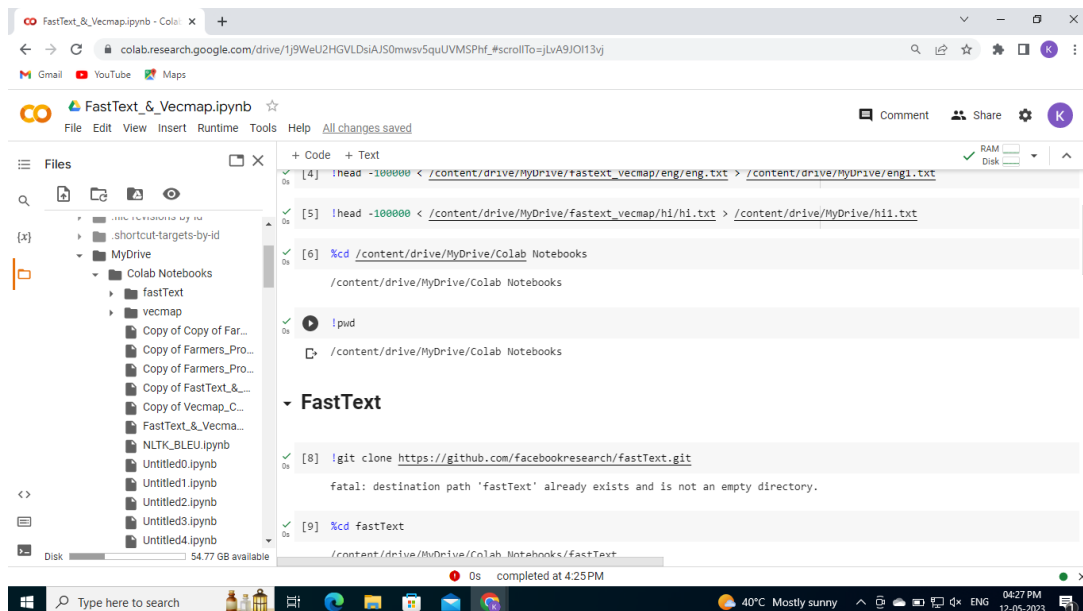
## 7. OUTPUTS



```
from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive

[2] !head -200000 < /content/drive/MyDrive/fasttext_vecmap/eng/eng.txt > /content/drive/MyDrive/engi.txt
[3] !head -200000 < /content/drive/MyDrive/fasttext_vecmap/hi/hi.txt > /content/drive/MyDrive/hii.txt
[4] !head -100000 < /content/drive/MyDrive/fasttext_vecmap/eng/eng.txt > /content/drive/MyDrive/engi.txt
[5] !head -100000 < /content/drive/MyDrive/fasttext_vecmap/hi/hi.txt > /content/drive/MyDrive/hii.txt
[6] %cd /content/drive/MyDrive/Colab Notebooks
/content/drive/MyDrive/Colab Notebooks
[7] !pwd
/content/drive/MyDrive/Colab Notebooks
```



```
[4] !head -100000 < /content/drive/MyDrive/fasttext_vecmap/eng/eng.txt > /content/drive/MyDrive/engi.txt
[5] !head -100000 < /content/drive/MyDrive/fasttext_vecmap/hi/hi.txt > /content/drive/MyDrive/hii.txt
[6] %cd /content/drive/MyDrive/Colab Notebooks
/content/drive/MyDrive/Colab Notebooks
[7] !pwd
/content/drive/MyDrive/Colab Notebooks

FastText

[8] !git clone https://github.com/facebookresearch/fastText.git
fatal: destination path 'fastText' already exists and is not an empty directory.
[9] %cd fastText
/content/drive/MyDrive/Colab Notebooks/fastText
```

```
[8] !git clone https://github.com/facebookresearch/fastText.git
fatal: destination path 'fastText' already exists and is not an empty directory.

[9] %cd fastText
/content/drive/MyDrive/Colab Notebooks/fastText

[10] %cd /content/drive/MyDrive/Colab Notebooks/fastText
/content/drive/MyDrive/Colab Notebooks/fastText

[11] ## Build using make
!make
make: Nothing to be done for 'opt'.

[12] ## Build using Cmake
!mkdir build && cd build && cmake ..
```

```
[15] #! ./fasttext skipgram -input input.txt -output output.txt -dim 300 -epoch 10
! ./fasttext skipgram -input /content/drive/MyDrive/fasttext_vecmap/eng/eng.txt -output /content/drive/MyDrive/fasttext_vecmap/eng/eng.txt -dim 300 -epoch 10

[16] txt -output output.txt -dim 300 -epoch 10
/content/drive/MyDrive/fasttext_vecmap/hi/hi.txt -output /content/drive/MyDrive/fasttext_vecmap/hi_ft-3000 -dim 300 -epoch 10

[17] %cd ..
/content/drive/MyDrive/Colab Notebooks

[18] !git clone https://github.com/artetxem/vecmap
fatal: destination path 'vecmap' already exists and is not an empty directory.

[19] %cd vecmap
/content/drive/MyDrive/Colab Notebooks/vecmap
```

```
[18] !git clone https://github.com/artetxem/vecmap
fatal: destination path 'vecmap' already exists and is not an empty directory.

[19] %cd vecmap
/content/drive/MyDrive/Colab Notebooks/vecmap

[21] ## Supervised
#python3 map_embeddings.py --supervised TRAIN.DICT SRC.EMB TRG.EMB SRC_MAPPED.EMB TRG_MAPPED.EMB

[22] ## Semi-supervised
#python3 map_embeddings.py --semi_supervised TRAIN.DICT SRC.EMB TRG.EMB SRC_MAPPED.EMB TRG_MAPPED.EMB

[23] ## Identical
#python3 map_embeddings.py --identical SRC.EMB TRG.EMB SRC_MAPPED.EMB TRG_MAPPED.EMB
```

```
[18] !git clone https://github.com/artetxem/vecmap
fatal: destination path 'vecmap' already exists and is not an empty directory.

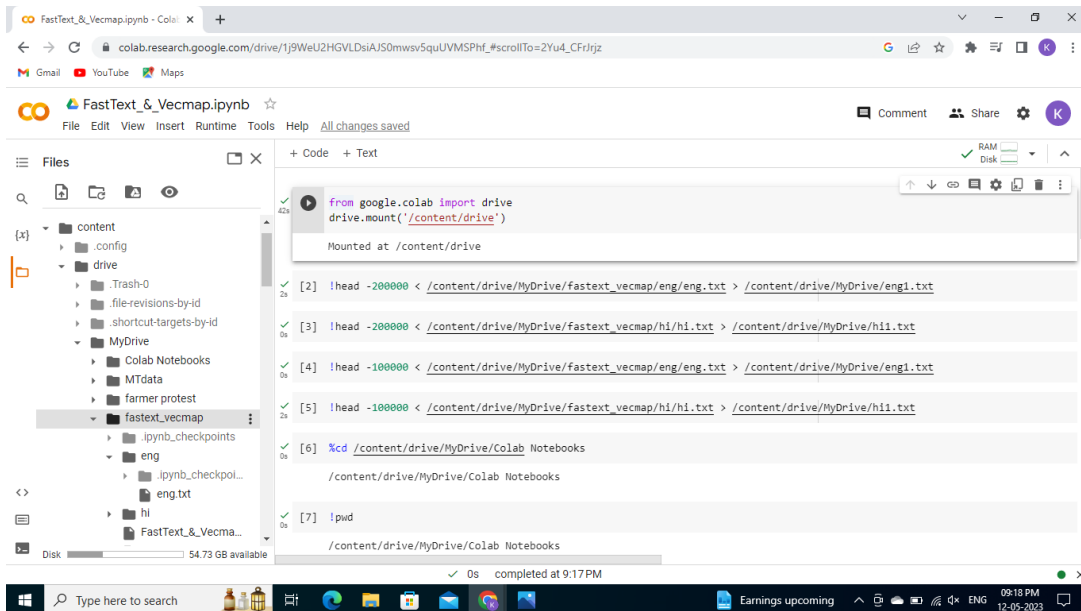
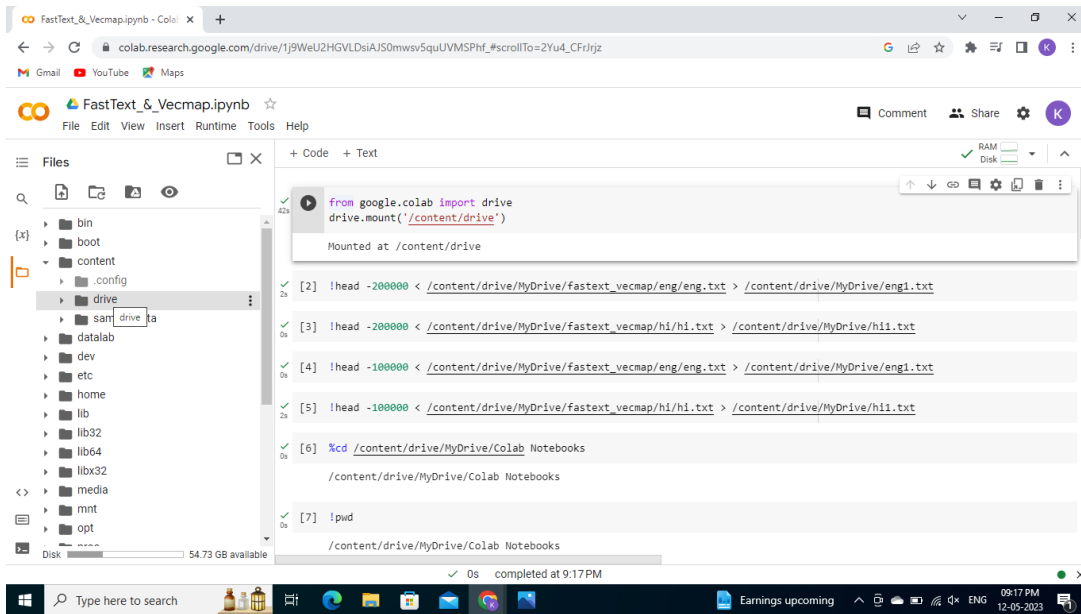
[19] %cd vecmap
/content/drive/MyDrive/Colab Notebooks/vecmap

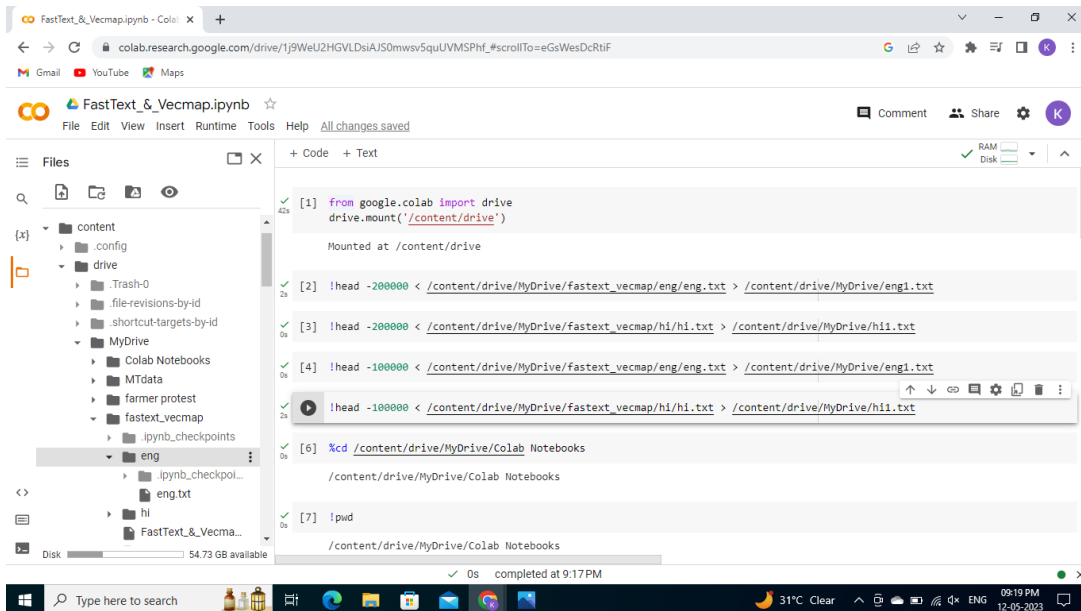
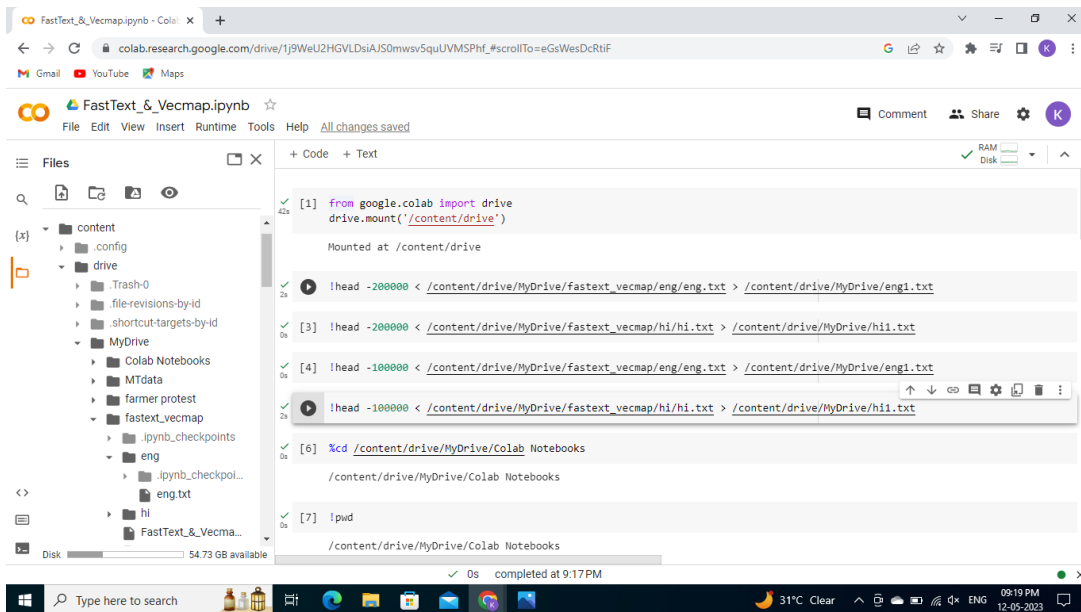
[21] ## Supervised
#python3 map_embeddings.py --supervised TRAIN.DICT SRC.EMB TRG.EMB SRC_MAPPED.EMB TRG_MAPPED.EMB

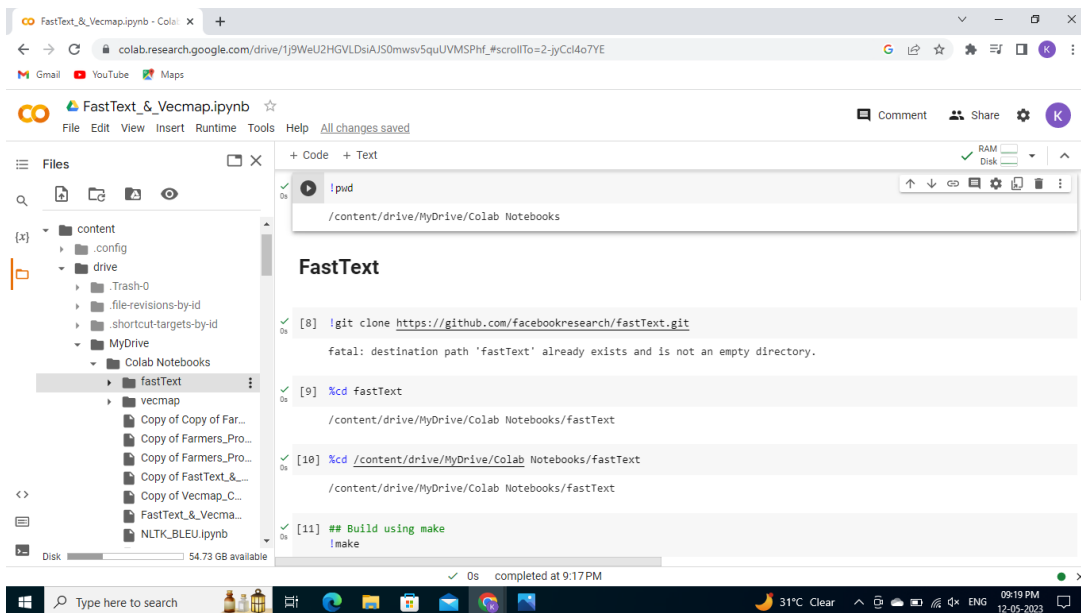
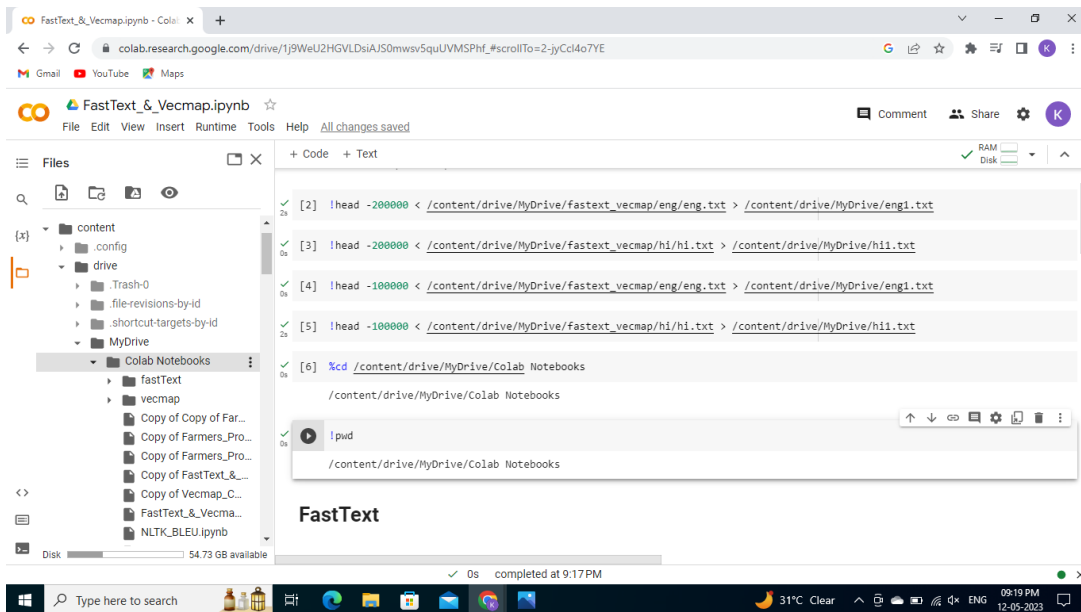
[22] ## Semi-supervised
#python3 map_embeddings.py --semi_supervised TRAIN.DICT SRC.EMB TRG.EMB SRC_MAPPED.EMB TRG_MAPPED.EMB

[23] ## Identical
#python3 map_embeddings.py --identical SRC.EMB TRG.EMB SRC_MAPPED.EMB TRG_MAPPED.EMB
```









```
[12] ## Build using Cmake
[12] #mkdir build && cd build && cmake ..
[12] #make && make install

[13] |chmod 755 fasttext

[14] pwd

/content/drive/MyDrive/Colab Notebooks/fastText

[15] #! ./fasttext skipgram -input input.txt -output output.txt -dim 300 -epoch 10
[15] ! ./fasttext skipgram -input /content/drive/MyDrive/fasttext_vecmap/eng/eng.txt -output /content/drive/MyDrive/fasttext_vecmap/eng/eng.txt

[16] #! ./fasttext skipgram -input input.txt -output output.txt -dim 300 -epoch 10
[16] ! ./fasttext skipgram -input /content/drive/MyDrive/fasttext_vecmap/hi/hi.txt -output /content/drive/MyDrive/fasttext_vecmap/hi/hi.txt

[17] %cd ..

/content/drive/MyDrive/Colab Notebooks
```

```
[12] ## Build using Cmake
    ##mkdir build && cd build && cmake ..
    ##make && make install

[13] !chmod 755 fasttext

[14] pwd

/content/drive/MyDrive/Colab Notebooks/fastText'

[15] #! ./fasttext skipgram -input input.txt -output output.txt -dim 300 -epoch 10
    ! ./fasttext skipgram -input /content/drive/MyDrive/fasttext_vecmap/eng/eng.txt -output /content/drive/MyDrive/fasttext_vecmap/eng/eng.txt

[16] txt -output output.txt -dim 300 -epoch 10
    t/drive/MyDrive/fasttext_vecmap/hi/hi.txt -output /content/drive/MyDrive/fasttext_vecmap/hi_ft-3000 -dim 300 -epoch 10

[17] %cd ..

/content/drive/MyDrive/Colab Notebooks
```

```
[1] from google.colab import drive
    drive.mount('/content/drive')

Mounted at /content/drive

[2] !head -200000 < /content/drive/MyDrive/fasttext_vecmap/eng/eng.txt > /content/drive/MyDrive/eng1.txt

[3] !head -200000 < /content/drive/MyDrive/fasttext_vecmap/hi/hi.txt > /content/drive/MyDrive/hi1.txt

[4] !head -100000 < /content/drive/MyDrive/fasttext_vecmap/eng/eng.txt > /content/drive/MyDrive/eng1.txt

[5] !head -100000 < /content/drive/MyDrive/fasttext_vecmap/hi/hi.txt > /content/drive/MyDrive/hi1.txt

[6] %cd /content/drive/MyDrive/Colab Notebooks

/content/drive/MyDrive/Colab Notebooks

[7] !pwd

/content/drive/MyDrive/Colab Notebooks
```

```
[19] %cd vecmap
/content/drive/MyDrive/Colab Notebooks/vecmap

[20] ## Supervised
#python3 map_embeddings.py --supervised TRAIN.DICT SRC.EMB TRG.EMB SRC_MAPPED.EMB TRG_MAPPED.EMB

[21] ## Semi-supervised
#python3 map_embeddings.py --semi_supervised TRAIN.DICT SRC.EMB TRG.EMB SRC_MAPPED.EMB TRG_MAPPED.EMB

[22] ## Identical
#python3 map_embeddings.py --identical SRC.EMB TRG.EMB SRC_MAPPED.EMB TRG_MAPPED.EMB

[23]
unsupervised_src_emb.vec trg_emb.vec SRC_MAPPED.EMB TRG_MAPPED.EMB --cuda
se files are generated using fasttext)
B (these will be generated.... just provide the path/to/location/of/empty/file)
unsupervised /content/drive/MyDrive/fasttext_vecmap/en_ft-300D.vec /content/drive/MyDrive/fasttext_vecmap/en_ft-300D.vec /c
```

```
[19] %cd vecmap
/content/drive/MyDrive/Colab Notebooks/vecmap

[20] ## Supervised
#python3 map_embeddings.py --supervised TRAIN.DICT SRC.EMB TRG.EMB SRC_MAPPED.EMB TRG_MAPPED.EMB

[21] ## Semi-supervised
#python3 map_embeddings.py --semi_supervised TRAIN.DICT SRC.EMB TRG.EMB SRC_MAPPED.EMB TRG_MAPPED.EMB

[22] ## Identical
#python3 map_embeddings.py --identical SRC.EMB TRG.EMB SRC_MAPPED.EMB TRG_MAPPED.EMB

[23]
/content/drive/MyDrive/fasttext_vecmap/SRC_Mapped_hi.emb /content/drive/MyDrive/fasttext_vecmap/TRG_Mapped_hi.emb --cuda
```

FastText\_Vecmap.ipynb - Colab

colab.research.google.com/drive/1j9WeU2HGVLdSiAJ50mwsV5quUVMSPhf\_#scrollTo=jLvA9JO13vj

FastText\_Vecmap.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Files

- bin
- boot
- content
  - .config
  - drive
    - Trash-0
    - file-revisions-by-id
    - shortcut-targets-by-id
    - MyDrive
    - sample\_data
  - datalab
  - dev
  - etc
  - home
  - lib
  - lib32
  - lib64
- Disk 54.77 GB available

Code

```
[1] from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive

[2] !head -200000 < /content/drive/MyDrive/fasttext_vecmap/eng/eng.txt > /content/drive/MyDrive/eng1.txt

[3] !head -200000 < /content/drive/MyDrive/fasttext_vecmap/hi/hi.txt > /content/drive/MyDrive/hi1.txt

[4] !head -100000 < /content/drive/MyDrive/fasttext_vecmap/eng/eng.txt > /content/drive/MyDrive/eng1.txt

[5] !head -100000 < /content/drive/MyDrive/fasttext_vecmap/hi/hi.txt > /content/drive/MyDrive/hi1.txt

%cd /content/drive/MyDrive/Colab Notebooks

/content/drive/MyDrive/Colab Notebooks

[7] !pwd

/content/drive/MyDrive/Colab Notebooks
```

completed at 4:25 PM

40°C Mostly sunny

04:26 PM 12-05-2023

retrieval\_techniques.ipynb - Colab

colab.research.google.com/drive/1vx8dqMuutQhG-qDAW70JRT9ocLmJEDHC

retrieval\_techniques.ipynb

File Edit View Insert Runtime Tools Help Last saved at 9:38 PM

Table of contents

- Word whose NN to retrieve
- Nearest Neighbour Technique
- Inverted Nearest Neighbour
- Inverted Softmax
- CSLS
- New Section
- Section

Code

```
cd /content/drive/My Drive/Embeddings/Techniques/vecmap/vecmap

/content/drive/My Drive/Embeddings/Techniques/vecmap/vecmap

[ ] import embeddings
import numpy as np
from cupy_utils import *

[ ] '''src_embeddings = '/content/drive/My Drive/Embeddings/ENG-HIN/VECMAP/SRC_MAPPED.EMB'
trg_embeddings = '/content/drive/My Drive/Embeddings/ENG-HIN/VECMAP/TRG_MAPPED.EMB'''

src_embeddings = '/content/drive/My Drive/VSD/english_sense_mapped_sup_big'
trg_embeddings = '/content/drive/My Drive/VSD/hindi_sense_mapped_sup_big'

srcfile = open(src_embeddings, encoding='utf-8', errors='surrogateescape')
trgfile = open(trg_embeddings, encoding='utf-8', errors='surrogateescape')
src_words, x = embeddings.read(srcfile, dtype='float32')
trg_words, z = embeddings.read(trgfile, dtype='float32')

[ ] len(src_words)
```

31°C Clear

09:38 PM 12-05-2023

retrieval\_techniques.ipynb - Colab

colab.research.google.com/drive/1vx8dqMuutQhG-qDAW70RT9ocLmJEDHC#scrollTo=TT6z6gH0mRbC

retrieval\_techniques.ipynb

File Edit View Insert Runtime Tools Help Last saved at 9:38 PM

Table of contents

- Word whose NN to retrieve
- Nearest Neighbour Technique
- Inverted Nearest Neighbour
- Inverted Softmax
- CSLS
- New Section
- Section

```
/content/drive/.shortcut-targets-by-id/120/Embeddings/Techniques/vecmap/vecmap

[ ] import embeddings
import numpy as np
from cupy_utils import *

[ ] '''src_embeddings = '/content/drive/My Drive/Embeddings/ENG-HIN/VECHAP/SRC_MAPPED.EMB'
trg_embeddings = '/content/drive/My Drive/Embeddings/ENG-HIN/VECHAP/TRG_MAPPED.EMB'''

src_embeddings = '/content/drive/My Drive/USD/english_sense_mapped_sup_big'
trg_embeddings = '/content/drive/My Drive/USD/hindi_sense_mapped_sup_big'

srcfile = open(src_embeddings, encoding='utf-8', errors='surrogateescape')
trgfile = open(trg_embeddings, encoding='utf-8', errors='surrogateescape')
src_words, x = embeddings.read(srcfile, dtype='float32')
trg_words, z = embeddings.read(trgfile, dtype='float32')

[ ] len(src_words)

531181
```

Type here to search

31°C Clear

09:39 PM 12-05-2023

retrieval\_techniques.ipynb - Colab

colab.research.google.com/drive/1vx8dqMuutQhG-qDAW70RT9ocLmJEDHC#scrollTo=i6Sk333GCUv7

retrieval\_techniques.ipynb

File Edit View Insert Runtime Tools Help Last saved at 9:38 PM

Table of contents

- Word whose NN to retrieve
- Nearest Neighbour Technique
- Inverted Nearest Neighbour
- Inverted Softmax
- CSLS
- New Section
- Section

```
src_embeddings = '/content/drive/My Drive/USD/english_sense_mapped_sup_big'
trg_embeddings = '/content/drive/My Drive/USD/hindi_sense_mapped_sup_big'

[ ] srcfile = open(src_embeddings, encoding='utf-8', errors='surrogateescape')
trgfile = open(trg_embeddings, encoding='utf-8', errors='surrogateescape')
src_words, x = embeddings.read(srcfile, dtype='float32')
trg_words, z = embeddings.read(trgfile, dtype='float32')

[ ] len(src_words)

531181

[ ] len(trg_words)

38768

[ ] embeddings.length_normalize(x)
embeddings.length_normalize(z)

xp = get_cupy()
x = cupy.asarray(x)
z = cupy.asarray(z)
```

Type here to search

31°C Clear

09:39 PM 12-05-2023



retrieval\_techniques.ipynb - Colab

colab.research.google.com/drive/1vx8dqMuutQhG-qDAW70/RT9ocLmJEDHC#scrollTo=T9JdpCQeQm4L

retrieval\_techniques.ipynb

File Edit View Insert Runtime Tools Help Last saved at 9:38 PM

Table of contents

- Word whose NN to retrieve
- Nearest Neighbour Technique
- Inverted Nearest Neighbour
- Inverted Softmax
- CSLS
- New Section
- Section

+ Code + Text

```
len(trg_words)

38760

[ ] embeddings.length_normalize(x)
    embeddings.length_normalize(z)

[ ] xp = get_cupy()
    x = cupy.asarray(x)
    z = cupy.asarray(z)

[ ] sr_word2emb = {}
    for word, vector in zip(src_words, x):
        sr_word2emb[word] = vector

    tr_word2emb = {}
    for word, vector in zip(trg_words, z):
        tr_word2emb[word] = vector

[ ] sr_word2emb['at']
```

31°C Clear 09:39 PM 12-05-2023

retrieval\_techniques.ipynb - Colab

colab.research.google.com/drive/1vx8dqMuutQhG-qDAW70/RT9ocLmJEDHC#scrollTo=qqHfudjYEUIN

retrieval\_techniques.ipynb

File Edit View Insert Runtime Tools Help Last saved at 9:38 PM

Table of contents

- Word whose NN to retrieve
- Nearest Neighbour Technique
- Inverted Nearest Neighbour
- Inverted Softmax
- CSLS
- New Section
- Section

+ Code + Text

```
len(src_words)

531181

[ ] len(trg_words)

38760

[ ] embeddings.length_normalize(x)
    embeddings.length_normalize(z)

[ ] xp = get_cupy()
    x = cupy.asarray(x)
    z = cupy.asarray(z)

[ ] sr_word2emb = {}
    for word, vector in zip(src_words, x):
        sr_word2emb[word] = vector

    tr_word2emb = {}
    for word, vector in zip(trg_words, z):
        tr_word2emb[word] = vector
```

31°C Clear 09:39 PM 12-05-2023

retrieval\_techniques.ipynb - Colab

colab.research.google.com/drive/1vx8dqMuutQhG-qDAW70/RT9ocLmJEDHC#scrollTo=bDJHVM6A8Qe

retrieval\_techniques.ipynb

File Edit View Insert Runtime Tools Help Last saved at 9:38 PM

Table of contents

- Word whose NN to retrieve
  - Nearest Neighbour Technique
  - Inverted Nearest Neighbour
- Inverted Softmax
- CSLS
- New Section
- Section

Word whose NN to retrieve

```
[ ] testWord = 'at#1'
```

1. Nearest Neighbour Technique

```
[ ] similarities = sr_word2emb[testWord].dot(z.T)
nn = similarities.argmax().tolist()
```

Nearest Neighbour

```
[ ] print(len(similarities))
trg_words[nn]
```

```
38760
'समतापूर्ण'
```

Type here to search

31°C Clear 09:40 PM 12-05-2023

retrieval\_techniques.ipynb - Colab

colab.research.google.com/drive/1vx8dqMuutQhG-qDAW70/RT9ocLmJEDHC#scrollTo=XxaBOZc3B\_Yg

retrieval\_techniques.ipynb

File Edit View Insert Runtime Tools Help Last saved at 9:38 PM

Table of contents

- Word whose NN to retrieve
- Nearest Neighbour Technique
  - Inverted Nearest Neighbour
- Inverted Softmax
- CSLS
- New Section
- Section

Nearest Neighbour Technique

```
[ ] print(len(similarities))
trg_words[nn]
```

```
38760
'समतापूर्ण'
```

k Nearest Neighbours

```
K = 10
knn = (-similarities).argsort().tolist()[1:K]
for kn in knn :
    print(trg_words[kn])
```

```
समतापूर्ण
सहयोगपूर्ण
दक्षतापूर्ण
एकको
क्रियायत
अनुकूलता
संचदको
सुगम्य
कार्यकुशल
दाची
```

Type here to search

31°C Clear 09:40 PM 12-05-2023

retrieval\_techniques.ipynb - Colab

colab.research.google.com/drive/1vx8dqMuutQhG-qDAW70JRT9ocLmJEDHC#scrollTo=BRTboX9JnBdb

retrieval\_techniques.ipynb

File Edit View Insert Runtime Tools Help Last saved at 9:38 PM

Table of contents

- Word whose NN to retrieve
- Nearest Neighbour Technique
- Inverted Nearest Neighbour**
- Inverted Softmax
- CSLS
- New Section
- Section

## 2. Inverted Nearest Neighbour

```

queue = []
K=10
best_rank = x.shape[0]
best_sim = -100
for i in range(0, z.shape[0], 500):
    j = min(i + 500, z.shape[0])
    similarities = z[i:j].dot(sr_word2emb[testWord])
    ind = (-similarities).argsort()
    ranks = ind.argsort()
    sims = similarities
    for k in range(1, j):
        rank = ranks[k-1]
        sim = sims[k-1]
        if rank < best_rank or (rank == best_rank and sim > best_sim):
            best_rank = rank
            best_sim = sim
            ivnn = k
            if len(queue) > K:
                queue.pop(0)
            queue.append(k)

```

31°C Clear 09:40 PM 12-05-2023

retrieval\_techniques.ipynb - Colab

colab.research.google.com/drive/1vx8dqMuutQhG-qDAW70JRT9ocLmJEDHC#scrollTo=Sf91\_0wDDb66

retrieval\_techniques.ipynb

File Edit View Insert Runtime Tools Help Last saved at 9:38 PM

Table of contents

- Word whose NN to retrieve
- Nearest Neighbour Technique
- Inverted Nearest Neighbour
- Inverted Softmax**
- CSLS
- New Section
- Section

```

[ ] if len(queue) > K:
    queue.pop(0)
    queue.append(k)

```

Nearest Neighbour

```

[ ] trg_words[ivnn]
</s>

```

K Nearest Neighbours

```

[ ] queue.reverse()
for i in queue:
    print(trg_words[i])
</s>

```

## 3. Inverted Softmax

31°C Clear 09:40 PM 12-05-2023

```
[19] %cd vecmap
/content/drive/MyDrive/Colab Notebooks/vecmap

[20] ## Supervised
#python3 map_embeddings.py --supervised TRAIN.DICT SRC.EMB TRG.EMB SRC_MAPPED.EMB TRG_MAPPED.EMB

[21] ## Semi-supervised
#python3 map_embeddings.py --semi_supervised TRAIN.DICT SRC.EMB TRG.EMB SRC_MAPPED.EMB TRG_MAPPED.EMB

[22] ## Identical
#python3 map_embeddings.py --identical SRC.EMB TRG.EMB SRC_MAPPED.EMB TRG_MAPPED.EMB

[23]
```

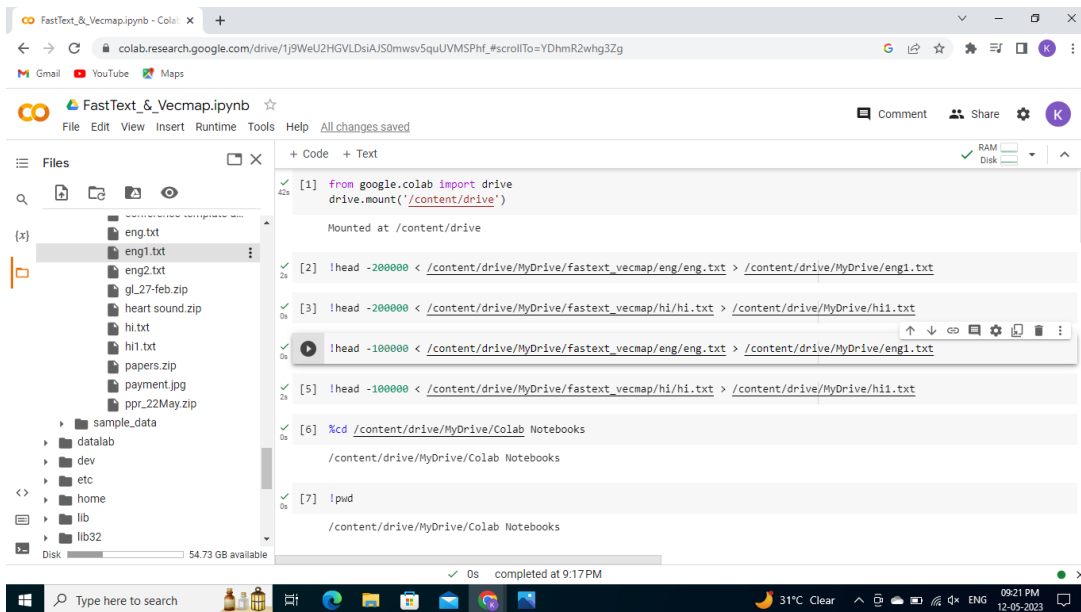
```
[19] %cd vecmap
/content/drive/MyDrive/Colab Notebooks/vecmap

[20] ## Supervised
#python3 map_embeddings.py --supervised TRAIN.DICT SRC.EMB TRG.EMB SRC_MAPPED.EMB TRG_MAPPED.EMB

[21] ## Semi-supervised
#python3 map_embeddings.py --semi_supervised TRAIN.DICT SRC.EMB TRG.EMB SRC_MAPPED.EMB TRG_MAPPED.EMB

[22] ## Identical
#python3 map_embeddings.py --identical SRC.EMB TRG.EMB SRC_MAPPED.EMB TRG_MAPPED.EMB

[23] python3 map_embeddings.py --unsupervised SRC.EMB TRG.EMB SRC_MAPPED.EMB TRG_MAPPED.EMB
```



```
[1] from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive

[2] !head -200000 < /content/drive/MyDrive/fasttext_vecmap/eng/eng.txt > /content/drive/MyDrive/eng1.txt

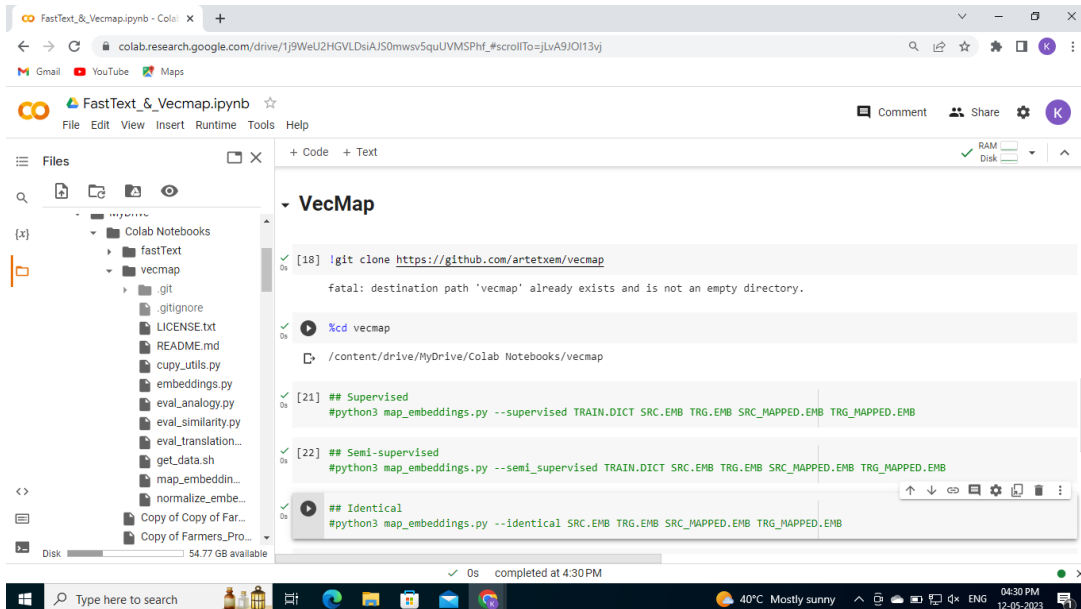
[3] !head -200000 < /content/drive/MyDrive/fasttext_vecmap/hi/hi.txt > /content/drive/MyDrive/hi1.txt

[4] !head -100000 < /content/drive/MyDrive/fasttext_vecmap/eng/eng.txt > /content/drive/MyDrive/eng1.txt

[5] !head -100000 < /content/drive/MyDrive/fasttext_vecmap/hi/hi.txt > /content/drive/MyDrive/hi1.txt

[6] %cd /content/drive/MyDrive/Colab Notebooks
/content/drive/MyDrive/Colab Notebooks

[7] !pwd
/content/drive/MyDrive/Colab Notebooks
```



```
[18] !git clone https://github.com/artetxem/vecmap

fatal: destination path 'vecmap' already exists and is not an empty directory.

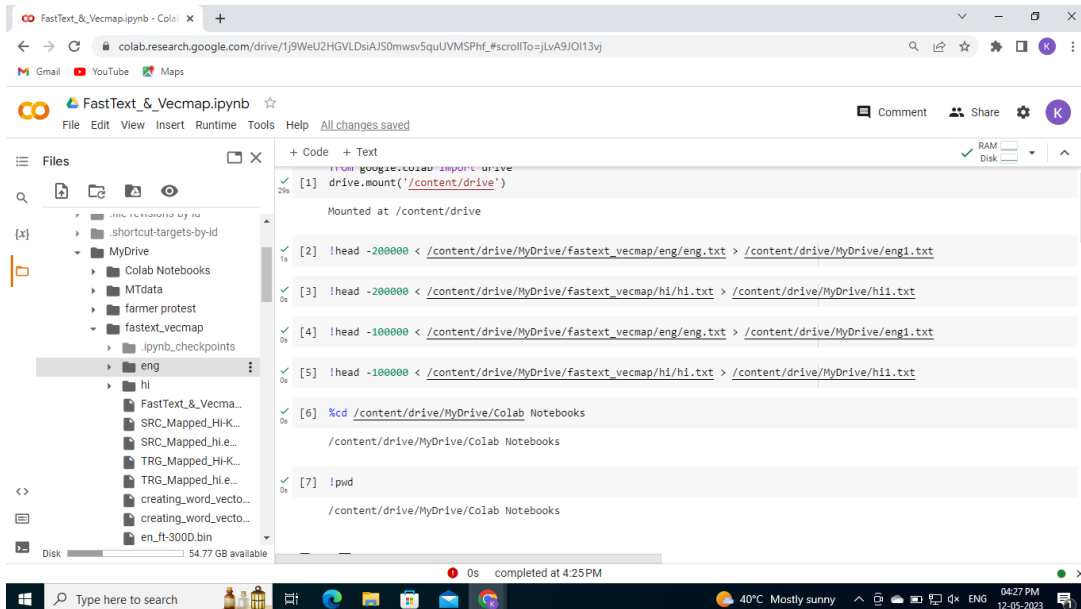
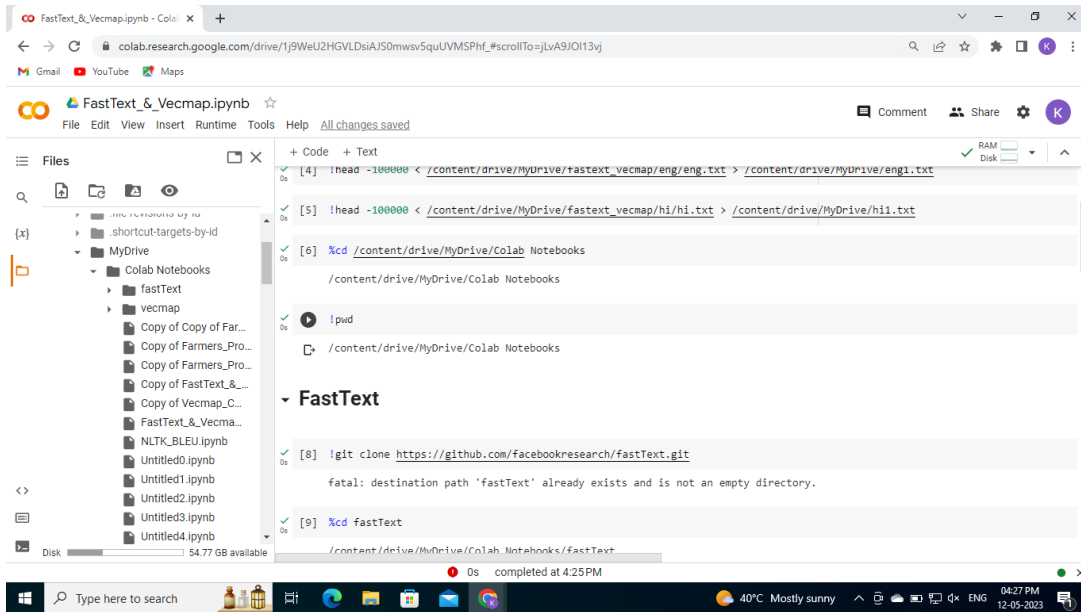
[19] %cd vecmap
/content/drive/MyDrive/Colab Notebooks/vecmap

[21] ## Supervised
#python3 map_embeddings.py --supervised TRAIN.DICT SRC.EMB TRG.EMB SRC_MAPPED.EMB TRG_MAPPED.EMB

[22] ## Semi-supervised
#python3 map_embeddings.py --semi_supervised TRAIN.DICT SRC.EMB TRG.EMB SRC_MAPPED.EMB TRG_MAPPED.EMB

[23] ## Identical
#python3 map_embeddings.py --identical SRC.EMB TRG.EMB SRC_MAPPED.EMB TRG_MAPPED.EMB
```





FastText\_Vecmap.ipynb - Colab

colab.research.google.com/drive/1j9WeU2HGVLdSiAJ50mwsV5quUVMSPhf\_#scrollTo=jLvA9JO13vj

FastText\_Vecmap.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Files

- bin
- boot
- content
  - content
    - .config
    - drive
      - Trash-0
      - file-revisions-by-id
      - shortcut-targets-by-id
      - MyDrive
      - sample\_data
    - datalab
    - dev
    - etc
    - home
    - lib
    - lib32
    - lib64

54.77 GB available

Code

```
[1] from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive

[2] !head -200000 < /content/drive/MyDrive/fasttext_vecmap/eng/eng.txt > /content/drive/MyDrive/eng1.txt

[3] !head -200000 < /content/drive/MyDrive/fasttext_vecmap/hi/hi.txt > /content/drive/MyDrive/hi1.txt

[4] !head -100000 < /content/drive/MyDrive/fasttext_vecmap/eng/eng.txt > /content/drive/MyDrive/eng1.txt

[5] !head -100000 < /content/drive/MyDrive/fasttext_vecmap/hi/hi.txt > /content/drive/MyDrive/hi1.txt

%cd /content/drive/MyDrive/Colab Notebooks

/content/drive/MyDrive/Colab Notebooks

[7] !pwd

/content/drive/MyDrive/Colab Notebooks
```

completed at 4:25 PM

40°C Mostly sunny

04:26 PM 12-05-2023

retrieval\_techniques.ipynb - Colab

colab.research.google.com/drive/1vx8dQMuutQhG-qDAW70jRT9ocLmJEDHC#scrollTo=kf4zzLTgC6NM

retrieval\_techniques.ipynb

File Edit View Insert Runtime Tools Help Last saved at 9:38 PM

Table of contents

- Word whose NN to retrieve
- Nearest Neighbour Technique
- Inverted Nearest Neighbour
- Inverted Softmax
  - CSLS
  - New Section
  - Section

Code

```
[ ] print(trg_words[nn])

अभिनेता

k Nearest Neighbours

K = 10
knn=(-p).argsort().tolist()[0:K]
for kn in knn :
    print(trg_words[kn])

अभिनेता
,अभिनेता
अभिनेता।
सह-अभिनेता
अभिनेतागण
अभिनेता/अभिनेत्री
अभिनेताओ
अभिनेत्री
निर्देशक
अभिनेत्री,
```

4 CSLS

31°C Clear

09:40 PM 12-05-2023



retrieval\_techniques.ipynb - Colab

colab.research.google.com/drive/1vx8dqMuutQhG-qDAW70jRT9ocLmJEDHC#scrollTo=2OPI-WNQ8zcv

retrieval\_techniques.ipynb

File Edit View Insert Runtime Tools Help Last saved at 9:38 PM

Table of contents

- Word whose NN to retrieve
- Nearest Neighbour Technique
- Inverted Nearest Neighbour
- Inverted Softmax
- CSLS
- New Section
- Section

## 4. CSLS

### New Section

```
def topk_mean(m, k, inplace=False): # TODO Assuming that axis is 1
    xp = get_array_module(m)
    n = m.shape[0]
    ans = xp.zeros(n, dtype=m.dtype)
    if k <= 0:
        return ans
    if not inplace:
        m = xp.array(m)
    ind0 = xp.arange(n)
    ind1 = xp.empty(n, dtype=int)
    minimum = m.min()
    for i in range(k):
        m_argmax = m.argmax(axis=-1, out=ind1)
        ans += m[ind0, ind1]
        m[ind0, ind1] = minimum
    return ans / k
```

Type here to search

31°C Clear 09:40 PM 12-05-2023

retrieval\_techniques.ipynb - Colab

colab.research.google.com/drive/1vx8dqMuutQhG-qDAW70jRT9ocLmJEDHC#scrollTo=vPU\_kaAchPgY

retrieval\_techniques.ipynb

File Edit View Insert Runtime Tools Help Last saved at 9:38 PM

Table of contents

- Word whose NN to retrieve
- Nearest Neighbour Technique
- Inverted Nearest Neighbour
- Inverted Softmax
- CSLS
- New Section
- Section

```
knn_sim_bwd[1:] = topk_mean(z[1:], dot(x.T), k=10, inplace=True)
similarities = 2*sr_word2emb[testword].dot(z.T) - knn_sim_bwd # Equivalent to the real CSLS scores for NN
nn = similarities.argmax()

print(trg_words[nn.tolist()])
```

अभिनेता

```
K = 10
knn = (-similarities).argsort().tolist()[1:K]
for kn in knn:
    print(trg_words[kn])
```

अभिनेता  
अभिनेता  
सह-अभिनेता  
अभिनेता  
अभिनेतागण  
अभिनेताओ  
अभिनेता/अभिनेत्री  
हास्यकलाकार  
अभिनेताओ  
अभिनेय

Type here to search

31°C Clear 09:40 PM 12-05-2023

FastText\_Vecmap.ipynb - Colab

colab.research.google.com/drive/1j9WeU2HGVLdsiAJ50mwsV5quUVMSPHf\_#scrollTo=jLvA9JOI13vj

FastText\_Vecmap.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

Files

- bin
- boot
- content
  - .config
  - drive
    - Trash-0
    - file-revisions-by-id
    - shortcut-targets-by-id
    - MyDrive
  - sample\_data
- datalab
- dev
- etc
- home
- lib
- lib32
- lib64

Disk 54.77 GB available

```
[1] from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive

[2] !head -200000 < /content/drive/MyDrive/fasttext_vecmap/eng/eng.txt > /content/drive/MyDrive/eng1.txt

[3] !head -200000 < /content/drive/MyDrive/fasttext_vecmap/h1/h1.txt > /content/drive/MyDrive/h11.txt

[4] !head -100000 < /content/drive/MyDrive/fasttext_vecmap/eng/eng.txt > /content/drive/MyDrive/eng1.txt

[5] !head -100000 < /content/drive/MyDrive/fasttext_vecmap/h1/h1.txt > /content/drive/MyDrive/h11.txt

%cd /content/drive/MyDrive/Colab Notebooks

/content/drive/MyDrive/Colab Notebooks

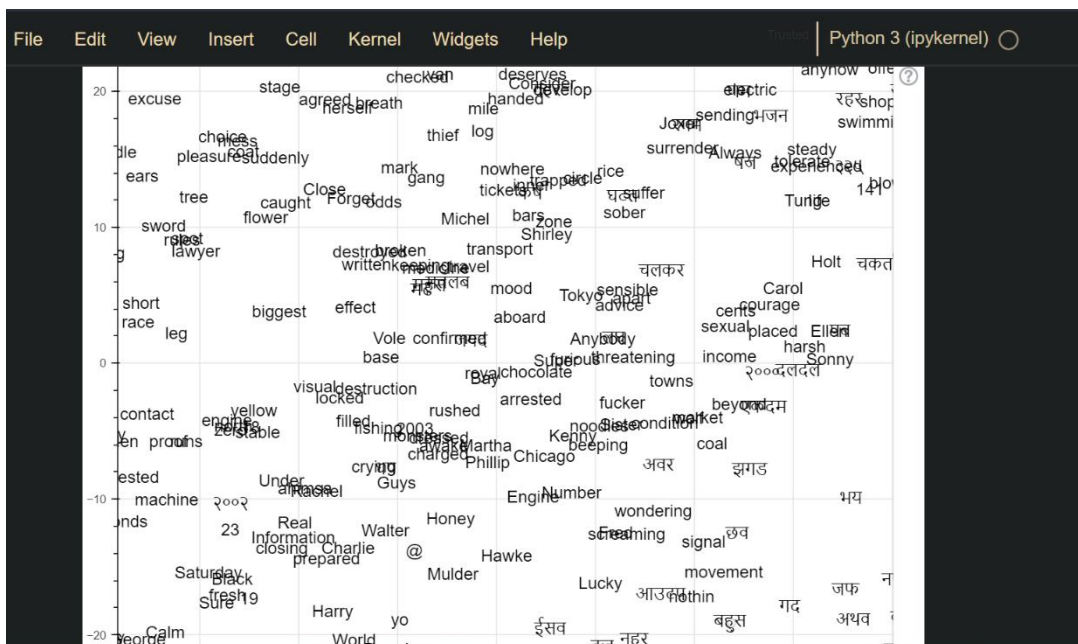
[7] !pwd

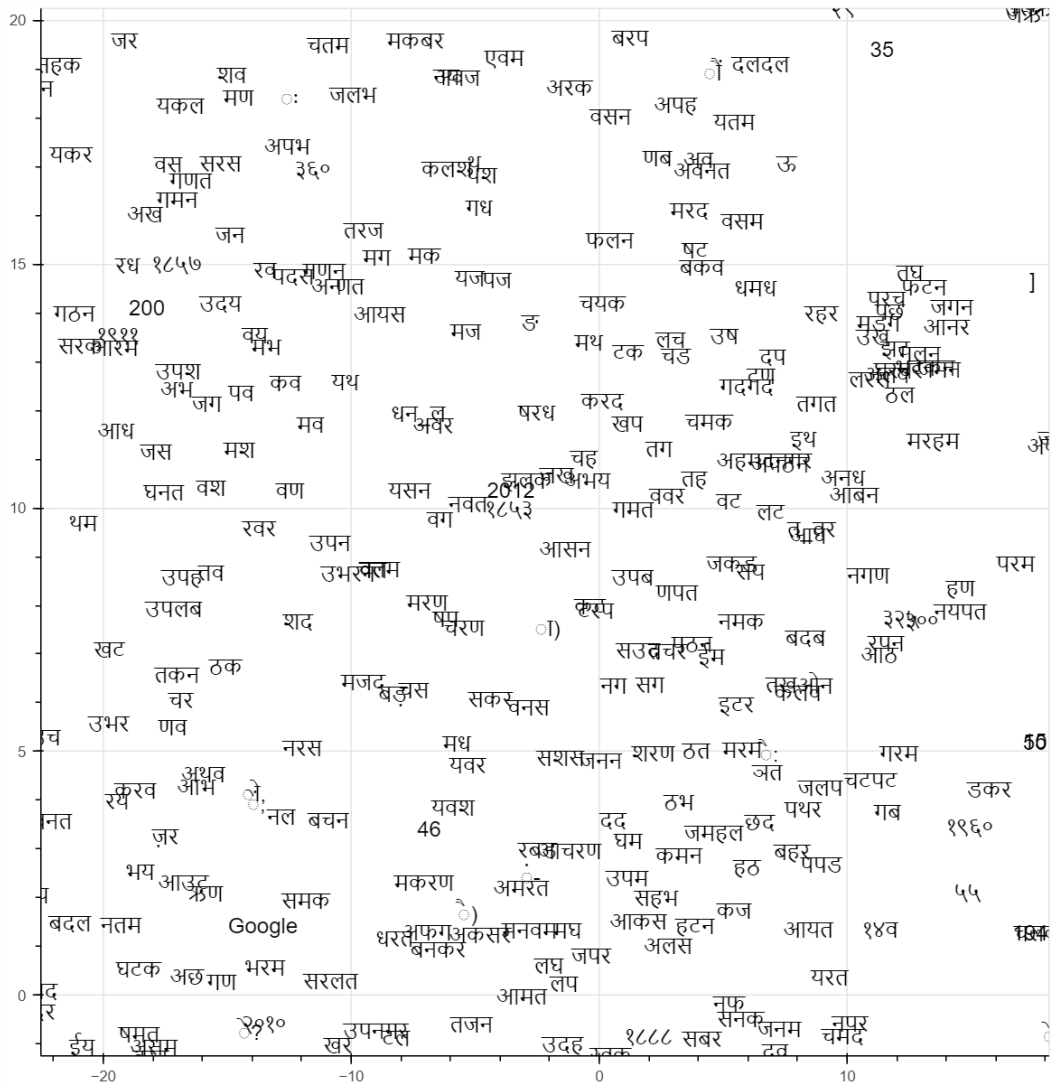
/content/drive/MyDrive/Colab Notebooks
```

0s completed at 4:25 PM

40°C Mostly sunny 04:25 PM 12-05-2023











```
In [27]: lower_bigram = Phraser(Phrases(lower_sents))
```

```
In [28]: lower_bigram.phrasegrams
```

```
{'two_daughters': 11.080894472729938,
 'her_sister': 16.93985297075414,
 'very_early': 10.517085686126077,
 'her_mother': 10.708214678014947,
 'long_ago': 59.22693146255729,
 'more_than': 28.53016238333433,
 'had_been': 21.58337149316804,
 'an_excellent': 37.41890603576534,
 'sixteen_years': 131.43021613989356,
 'miss_taylor': 420.4375727213963,
 'mr_woodhouse': 104.1999395195192,
 'very_fond': 24.18592621729314,
 'passed_away': 11.75157033589844,
 'too_much': 30.363341094363623,
 'did_not': 10.846285856844784,
 'any_means': 14.29426622702947,
 'after_dinner': 18.607525034015455,
 'mr_weston': 91.63366549621855,
 'five_years': 37.66459722425521,
 'years_old': 48.5994906090284,
 'seven_years': 50.334976394001785,
 'each_other': 71.31335962645632,
 'well_informed': 14.185145241835276,
 ...}
```

```
In [30]: lower_bigram["john lives in new york city"].split()
```

```
['john', 'lives', 'in', 'new_york', 'city']
```

```
In [ ]:
```

```
In [17]: bigram = Phraser(phrases) # create a more efficient Phraser object for transforming sentences
```

```
In [20]: bigram.phrasegrams # output count and score of each bigram
```

```
{'two_daughters': 11.966987886528118,
 'her_sister': 17.796341912611076,
 "'_s': 31.066694850762417,
 'very_early': 11.01230173457644,
 'her_mother': 13.529621959045564,
 'long_ago': 63.22435639270114,
 'more_than': 29.024006819814797,
 'had_been': 22.306349272800997,
 'an_excellent': 39.064443355850045,
 'Miss_Taylor': 453.76578390553544,
 'very_fond': 24.134631699685762,
 'passed_away': 12.359716162995981,
 'too_much': 31.376458659431253,
 'did_not': 11.72858690304441,
 'any_means': 14.097169263925728,
 'wedding_': 17.469774011299435,
 'her_father': 13.129762639674155,
 'after_dinner': 21.528861425991604,
 'self_-': 47.79007603091109,
 'sixteen_years': 107.04772502472798,
 'five_years': 48.129339674923365,
 'years_old': 54.73622181125361,
 'seven_years': 52.59487691468612,
 ...}
```

```
In [21]: tokenized_sentence = "John lives in New York City".split()
```

```
In [22]: tokenized_sentence
```

```
['John', 'lives', 'in', 'New', 'York', 'City']
```

```
In [23]: bigram[tokenized_sentence]
```

```
['John', 'lives', 'in', 'New_York', 'City']
```



```

1 x,y,token
2 -9.551004,37.971066,.
3 -13.105887,30.385374,"","
4 -51.048424,15.304509,'
5 -27.199919,28.372332,I
6 -12.993579,-41.98631,the
7 -60.43505,8.889997,you
8 9.657712,48.194504,-
9 -53.457455,-6.6015577,to
10 -9.646741,30.096273,?
11 -35.330105,-26.849537,a
12 -36.603592,3.2311103,s
13 -18.281649,-38.653362,of
14 -56.27987,11.472625,t
15 -37.87144,15.723879,it
16 -17.988829,-27.116003,and
17 -12.99112,30.093119,...
18 -9.5341835,37.979416,!
19 -34.256542,1.3751588,that
20 -8.7836685,-42.954662,in
21 -36.596146,20.60377,You
22 -40.211143,22.98699,is
23 -64.505646,-9.393259,me
24 -29.903452,-25.594872,for
25 -49.59596,-2.385206,have
26 -33.877434,1.0073704,this
27 -30.201143,-14.562382,my
28 -57.70048,3.49553,be

```

```

1 x,y,token
2 -31.267056,14.435461,ा
3 -31.665245,15.432334,ू
4 -31.27835,14.201794,ि
5 -31.094366,13.949045,क
6 -32.40465,13.936923,े
7 -31.162615,14.414888,ी
8 -31.094072,12.998591,र
9 -31.065664,15.196336,स
10 -31.232944,13.057977,न
11 -29.536158,15.066569,त
12 -15.330632,2.5334463,म
13 -19.272697,-17.183802,ह
14 -31.864832,14.452369,ो
15 -30.430475,14.166613,य
16 -13.551956,1.8157904,ल
17 -27.778118,13.641204,द
18 8.32831,-15.2411,ु
19 -31.625868,16.183027,प
20 -19.37373,22.870945,व
21 45.337692,-17.876661,-
22 -4.1345124,6.093872,ं
23 -32.708755,11.3839035,ज
24 -5.5682516,-31.076607,ग
25 -7.5022063,4.893635,ँ
26 -22.339146,-24.659296,े
27 5.3071213,19.335367,्र
28 -14.406739,2.5929267,ब

```

## 8. CONCLUSION

In this project, we investigated and compared various word embedding learning methods, such as Word2Vec, GloVe, and FastText. Their performance on various NLP tasks was to be assessed for effectiveness, efficiency, and performance. Following extensive investigation and analysis, we have reached the following conclusions:

1. *Effectiveness of Word Embedding:* Word2Vec, GloVe, and FastText all showed the capacity to record syntactic and semantic links between words. They created word embeddings that allowed computational models to better comprehend and analyse spoken language by representing words as dense vectors in a continuous vector space.
2. *Performance Variations:* The methods varied in how well they performed when put to use on various evaluation tasks. Word2Vec demonstrated superior training efficiency, giving it a great option for datasets with a lot of data. GloVe performed exceptionally well at identifying word similarity, making it appropriate for uses in which semantic connections are important. For morphologically complex languages, FastText's subword information modelling proven to be especially useful.
3. *Computing Requirements:* When compared to Word2Vec and FastText, GloVe requires greater memory and processing power. GloVe embedding training may take longer, especially when working with huge datasets. FastText, due to its subword information approach, required additional preprocessing steps and incurred longer training times.
4. *Generalization and Resilience:* Word2Vec and GloVe showed robustness when processing words that were not part of their respective corpora, whereas FastText's subword modelling allowed it to produce representations for unheard words based on their character n-grams. This

skill improves word embeddings' ability to handle words that aren't in the training set.

5. *Application Specificity*: Depending on the particular requirements of the NLP work at hand, the word embedding learning technique is selected. For jobs that call for effective training and all-purpose semantic representations, Word2Vec excels. GloVe performs remarkably well on tests that emphasise analogy and resemblance between words. In situations involving morphologically difficult languages and uncommon words, FastText excels.

By NLP practitioners can choose the best word embedding learning method for their particular applications by studying the advantages and disadvantages of each strategy.

Overall, word embedding learning techniques have transformed the area of NLP by making it possible for robots to efficiently comprehend and interpret human language. These methods have produced continuous vector representations that have proven useful for a variety of NLP tasks, including sentiment analysis, machine translation, information retrieval, and more.

Contextualized embeddings and transformer-based models, which have demonstrated promising results in capturing contextual information and enhancing the performance of NLP systems, could be the focus of future research in word embeddings.

In conclusion, this project has offered insightful information on the various word embedding learning algorithms and their relative advantages and disadvantages. This study's findings will lay the groundwork for utilising word embeddings in a variety of NLP applications and will act as an inspiration for subsequent developments in the area.

## 9.REFERENCES

[1] Dean, J., Chen, K., Corrado, G., and Mikolov, T. (2013). word representations in vector space are efficiently estimated. Throughout the International Conference on Learning Representations's Proceedings (ICLR).

[2] J. Pennington, R. Socher, and C. Manning (2014). GloVe stands for "global vectors for word representation." Empirical Methods in Natural Language Processing Conference Proceedings (EMNLP).

[3] A. Joulin, P. Bojanowski, E. Grave, T. Mikolov, and A. (2017). incorporating subword data to improve word vectors. 135–146 are included in Transactions of the Association for Computational Linguistics, 5, 2015.

[4] Chang, M. W., Lee, K., and Toutanova, K.; Devlin, J. (2018). BERT: Deep bidirectional transformers that have been pre-trained for language comprehension. Human Language Technologies: Proceedings of the Conference of the Association for Computational Linguistics' North American Chapter (NAACL-HLT).

[5] Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep contextualized word representations. In Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)

[6] Ludovic Denoyer, Herve J. egou, Marc'Aurelio Ranzato, Guillaume Lample, and Alexis Conneau (2018). Without Parallel Data, Word Translation ICLR conference paper that was published.

[7] Shweta Chauhan, Umesh Pant , Mustafa , Philemon Daniel. (2020). A Robust Unsupervised Word by Word Translation for Morphological Rich Languages using different Retrieval Techniques. Journal of Critical Reviews ISSN - 2394 - 5125 Vol 7, Issue 17.

[8] Shweta Chauhan, Shefali Saxena, Philemon Daniel. (2021). Fully unsupervised word translation from cross-lingual word embeddings especially for healthcare professionals. Int J Syst Assur Eng Manag (March 2022) 13(Suppl. 1):S28–S37.

[9] Guillaume Lample, Alexis Conneau, Ludovic Denoyer, Marc’Aurelio Ranzato. (2018). Unsupervised Machine Translation using Monolingual Corpora only. Published as a conference paper at ICLR.