TELUGU KNOWLEDGE GRAPH LINK COMPLETION

A Project Progress Report

Submitted in partial fulfillment for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Submitted by

N.L.CHAKRITHA	(N190457)
A.NEELIMA	(N190406)
CH.BHANU PRAKASH REDDY	(N190563)
K.KOTESWARA REDDY	(N190579)
CH.H.S.D.PRAVALLIKA	(N190186)

Under the Esteem Guidance of

Mr. Rayala Upendar Rao

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

Rajiv Gandhi University of Knowledge Technologies – Nuzvid

Nuzvid, Krishna, Andhra Pradesh – 521202.

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

Rajiv Gandhi University of Knowledge Technologies – Nuzvid

Nuzvid, Krishna, Andhra Pradesh – 521202.

CERTIFICATE OF COMPLETION

This is to certify that the work entitled, **"TELUGU KNOWLEDGE GRAPH LINK COMPLETION"** is the bonafied work of **N.L.CHAKRITHA (ID NO: N190457), A.NEELIMA (ID NO: N190406), CH.BHANU PRAKASH REDDY (ID NO: N190563), K.KOTESWARA REDDY (ID NO: N190579), CH.H.S.D PRAVALLIKA (ID NO: N190186)** carried out under my guidance and supervision for 3rd year project of **Bachelor of Technology** in the department of Computer Science and Engineering under RGUKT IIIT Nuzvid. This work is done during the academic session January 2024 – June 2024, under our guidance.

-- ------------------------------------

Mr. Rayala Upendar Rao
Assistant professor,
Department of CSE,
RGUKT Nuzvid

Dr.D.V.Nagarjana Devi
Assistant Professor,
Head of the Department,
Department of CSE,
RGUKT Nuzvid.

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

Rajiv Gandhi University of Knowledge Technologies – Nuzvid

Nuzvid, Krishna, Andhra Pradesh – 521202.

CERTIFICATE OF EXAMINATION

This is to certify that the work entitled, **"TELUGU KNOWLEDGE GRAPH LINK COMPLETION"** is the bonafied work of **N.L.CHAKRITHA (ID NO: N190457), A.NEELIMA (ID NO: N190406), CH.BHANU PRAKASH REDDY (ID NO: N190563), K.KOTESWARA REDDY (ID NO: N190579), CH.H.S.D PRAVALLIKA (ID NO: N190186)** and here by accord our approval of it as a study carried out and presented in a manner required for its acceptance in 3rd year of **Bachelor of Technology** for which it has been submitted. This approval does not necessarily endorse or accept every statement made, opinion expressed or conclusion drawn, as a recorded in this thesis. It only signifies the acceptance of this thesis for the purpose for which it has been submitted.

--	-----------------------------
Mr.Rayala Upendar Rao	**Project Examiner**
Assistant Professor,	RGUKT-Nuzvid
Department of CSE,	
RGUKT-NUZVID	

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

Rajiv Gandhi University of Knowledge Technologies – Nuzvid

Nuzvid, Krishna, Andhra Pradesh – 521202.

DECLARATION

We **"N.L.CHAKRITHA (ID NO: N190457), A.NEELIMA (ID NO: N190406), CH.BHANU PRAKASH REDDY (ID NO: N190563), K.KOTESWARA REDDY (ID NO: N190579), CH.H.S.D PRAVALLIKA (ID NO: N190186)"** hereby declare that the project report entitled **"Telugu Knowledge Graph Link Completion"** done by us under the guidance of Mr. Upendar Rao, Assistant Professor is submitted for the partial fulfillment for the award of degree of Bachelor of Technology in Computer Science and Engineering during the academic session January 2024 - June 2024 at RGUKT-Nuzvid.

We also declare that this project is a result of our own effort and has not been copied or imitated from any source. Citations from any websites are mentioned in the references. The results embodied in this project report have not been submitted to any other university or institute for the award of any degree or diploma.

Date: 06-07-2024

Place: Nuzvid

N L CHAKRITHA	(N190457)
A NEELIMA	(N190406)
CH BHANU PRAKASH REDDY	(N190563)
K KOTESWARA REDDY	(N190579)
CH.H.S.D. PRAVALLIKA	(N190186)

ACKNOWLEDGEMENT

We would like to express our profound gratitude and deep regards to our guide **Mr. Rayala Upendar Rao** for his exemplary guidance, monitoring and constant encouragement to us throughout the B.Tech course. We shall always cherish the time spent with him during the course of this work due to the invaluable knowledge gained in the field of reliability engineering.

We are extremely grateful for the confidence bestowed in us and entrusting our project entitled **"Telugu Knowledge Graph Link Completion"**

We express gratitude to Dr.D.V.Nagarjana Devi(HOD of CSE) and other faculty members for being source of inspiration and constant encouragement which helped us in completing the project successfully.

Our sincere thanks to all the batch mates of 2019 CSE, who have made our stay at RGUKT-NUZVID, a memorable one.

Finally, yet importantly, we would like to express our heartfelt thanks to our beloved God and parents for their blessings, our friends for their help and wishes for the successful completion of this project.

ABSTRACT

Knowledge Graph Completion (KGC) is an essential task in artificial intelligence that focuses on enriching knowledge graphs by inferring missing information and relationships. Knowledge graphs represent information in a structured format, comprising entities, attributes, and their interconnections. Despite their usefulness, they often suffer from incompleteness due to the dynamic and expansive nature of real-world information. KGC aims to address this issue by predicting and filling in these gaps. The project involves leveraging advanced techniques, including machine learning, graph theory, and natural language processing, to enhance the knowledge graph's comprehensiveness. Methods such as embedding-based models, rule-based systems, and neural network architectures are explored and implemented. Embedding-based models capture the semantic similarities between entities, facilitating the prediction of missing links. Rule-based systems utilize logical rules to infer new relationships from existing data. Neural network architectures, particularly those designed for graph data, learn complex patterns and structures to predict missing information. By evaluating these methods on benchmark datasets, the project aims to identify the most effective approaches for KGC. Improving the accuracy and efficiency of KGC can significantly enhance the reliability of knowledge representations. This, in turn, benefits various applications, including search engines, recommendation systems, and intelligent virtual assistants. Comprehensive knowledge graphs enable more accurate information retrieval, personalized recommendations, and improved user interactions. Ultimately, the project's goal is to contribute to the development of more robust and complete knowledge graphs, advancing the field of artificial intelligence and its applications in everyday technology.

Table of Contents

List of figures

CHAPTER 1

INTRODUCTION

Knowledge Graph Completion (KGC) is a critical research domain within artificial intelligence that focuses on the enhancement of knowledge graphs by inferring and adding missing entities and relationships. Knowledge graphs organize information into a structured format of entities, attributes, and the relationships connecting them. These structures form the foundation for numerous AI-driven applications, such as search engines, recommendation systems, and intelligent virtual assistants, by enabling machines to understand and process complex data relationships. However, due to the vast and constantly evolving nature of real-world information, knowledge graphs are frequently incomplete. Addressing these gaps is essential for improving the accuracy, utility, and reliability of these knowledge representations.

In this project, we utilize the TransE model, a prominent embedding-based approach, for the task of knowledge graph completion. TransE, short for Translational Embeddings, is designed to represent both entities and relationships as low-dimensional vectors in a continuous vector space. The core idea of TransE is that the relationship between two entities can be modeled as a vector translation operation. For any given triplet (head entity, relationship, tail entity), the TransE model aims to ensure that the vector of the head entity plus the vector of the relationship is close to the vector of the tail entity. This straightforward yet powerful approach enables TransE to effectively capture and leverage the underlying structure of the knowledge graph, facilitating the prediction of missing links and entities.

This project involves implementing the TransE model and rigorously evaluating its performance on benchmark datasets. Through this process, we aim to enhance the completeness and accuracy of knowledge graphs. The successful application of TransE is expected to result in more reliable and comprehensive knowledge representations, which in turn will improve the performance of AI applications that depend on these graphs. By conducting thorough experimentation and analysis, we seek to contribute to the advancement of knowledge graph completion methodologies and to demonstrate the practical benefits of embedding-based models like TransE. Our ultimate goal is to pave the way for more robust and complete knowledge graphs, driving forward the field of artificial intelligence and its myriad applications.

1.1 TransE model

TransE (Translating Embeddings) is a widely-used model for knowledge graph completion that represents entities and relationships in a continuous vector space. Introduced by Antoine Bordes et al. in 2013, TransE aims to predict missing links in a knowledge graph by learning low-dimensional embeddings for entities and relations.

Model Overview

In a knowledge graph, facts are typically represented as triples (head, relation, tail), denoted as (h, r, t). For example, in the triple (Paris, capital_of, France), "Paris" is the head entity (h), "capital_of" is the relation (r), and "France" is the tail entity (t).

TransE operates on the principle that the embedding of the head entity (h) plus the embedding of the relation (r) should be close to the embedding of the tail entity (t) in the vector space. Mathematically, this can be expressed as:

$$\mathrm{h} + \mathrm{r} \approx \mathrm{t}$$

where h, r and t are the vector representations of the head, relation, and tail, respectively.

Objective Function

To learn the embeddings, TransE minimizes a margin-based ranking loss function, which is designed to make the score of a correct triple (h, r, t) lower than the score of an incorrect triple (h', r, t') by a margin. The score function used is typically the L2 norm of the difference between $\mathrm{h} + \mathrm{r}$ and t.

$$\mathrm{f(h, r, t)} = \| \mathrm{h} + \mathrm{r} - \mathrm{t} \|_2$$

The loss function for a single training example is defined as:

$$\mathrm{L} = \sum (\mathrm{h, r, t}) \in \mathrm{T} \sum (\mathrm{h}', \mathrm{r}, '\mathrm{t}') \in [\gamma + \mathrm{f(h, r, t)} - \mathrm{f(h', r, t')}]$$

where T is the set of correct triples, T' is the set of incorrect triples generated by corrupting the head or tail entity, γ is the margin, and $[x]+= \max(0, x)$ $[x]_+ = \max(0, x)$ denotes the hinge loss.

Training

The TransE model is trained using stochastic gradient descent (SGD) or its variants. During training, for each correct triple, one or more incorrect triples are generated by randomly replacing the head or tail entity with another entity from the knowledge graph. The embeddings are then updated to minimize the loss function.

1.2 Stanza

Stanza is a Python natural language processing (NLP) toolkit developed by the Stanford NLP Group. It provides a wide array of tools for processing textual data, such as tokenization, part-of-speech tagging, named entity recognition, and dependency parsing. These tools can be effectively integrated into knowledge graph construction and completion workflows to enhance the extraction and integration of information from unstructured text sources.

1.2.1 Role of Stanza in Knowledge Graph Construction and Completion

1.2.1.1 Information Extraction

Named Entity Recognition (NER)

Stanza can identify and classify entities mentioned in text, such as persons, organizations, locations, and dates. These entities become the nodes in a knowledge graph.

Dependency Parsing

By analyzing the grammatical structure of sentences, Stanza can identify relationships between entities. These relationships become the edges connecting the nodes in the knowledge graph.

1.2.1.2 Entity Linking

Stanza's NER component can detect entities in text, which can then be linked to existing entities in the knowledge graph, facilitating the integration of new information and the resolution of ambiguities.

1.2.1.3 Relation Extraction

Using dependency parsing, Stanza can help identify relationships between entities mentioned in text. For instance, in the sentence "Barack Obama was born in Hawaii," Stanza can help extract the relationship "born _in" between "Barack Obama" and "Hawaii."

1.2.1.4 Conceptual Understanding

Stanza can process large text corpora to provide context around entities and relationships. This contextual understanding is crucial for accurately integrating new information into the knowledge graph and inferring missing links.

1.2.2 Work-flow for Using Stanza in Knowledge Graphs

1.2.2.1 Text pre-processing

Raw textual data is first preprocessed using Stanza’s tokenization and sentence splitting tools.

1.2.2.2 Entity and Relation Extraction

Apply Stanza’s NER to extract entities. Use dependency parsing to extract syntactic relationships, which are then mapped to semantic relations in the knowledge graph.

1.2.2.3 Entity Linking and Disambiguation

Match extracted entities with existing entities in the knowledge graph to link new information correctly. Handle ambiguities by considering the context provided by Stanza’s outputs.

1.2.2.4 Integration into the Knowledge Graph

Add the extracted entities and relations to the knowledge graph, ensuring consistency and resolving conflicts with existing data.

1.3 Knowledge Graph Completion

Use the newly added information to predict and infer missing links in the knowledge graph, leveraging techniques such as embedding models (e.g., TransE) to enrich the graph further.

Consider the following text:

"Marie Curie, a physicist, and chemist, conducted pioneering research on radioactivity. She won the Nobel Prize in Physics in 1903 and in Chemistry in 1911."

Using Stanza, the following steps can be performed

Entity Extraction

Identify entities: "Marie Curie," "physicist," "chemist," "radioactivity," "Nobel Prize," "Physics," "1903," "Chemistry," "1911".

Relation Extraction:

Identify relationships: "Marie Curie" - [profession] - "physicist," "Marie Curie" - [profession] - "chemist," "Marie Curie" - [research] - "radioactivity," "Marie Curie" - [award] - "Nobel Prize," etc.

Integration

Add nodes and edges to the knowledge graph, linking "Marie Curie" with her professions, research area, and awards.

1.4 Indic NLP

Indic NLP refers to natural language processing techniques tailored for languages spoken in the Indian subcontinent, including Hindi, Bengali, Tamil, Telugu, Kannada, Marathi, Gujarati, Punjabi, and others. These languages pose unique challenges due to their diverse scripts, rich morphology, and syntactic structures. Developing NLP tools for Indic languages is crucial for creating and completing knowledge graphs that can capture and represent information in these languages.

1.4.1 Components of Indic NLP

Tokenization

Segmenting text into words, phrases, or other meaningful units. Indic languages often have complex word boundaries due to compound words and agglutinative morphology.

Part-of-Speech Tagging (POS)

Assigning grammatical categories to words, such as nouns, verbs, adjectives, etc. Indic languages have rich inflectional morphology, making POS tagging more complex.

Named Entity Recognition (NER)

Identifying and classifying named entities (e.g., names of people, organizations, locations) in text. Indic languages have specific entity types and variations in names due to cultural diversity.

Dependency Parsing
Analysing grammatical structure to understand relationships between words. Indic languages often have free word order, requiring robust parsing techniques.

Machine Translation
Translating text between Indic languages and other languages. High-quality translation is essential for creating multilingual knowledge graphs.

Sentiment Analysis
Determining sentiment or emotion expressed in text. Indic languages have rich expressions and idiomatic phrases that need to be accurately interpreted.

1.4.2 Indic NLP in Knowledge Graph Construction and Completion

Entity Extraction and Linking
Use Indic NER tools to extract entities from text in Indic languages. Link these entities to existing nodes in the knowledge graph, ensuring correct identification and resolution of entities.

Relation Extraction
Apply dependency parsing to identify relationships between entities in Indic text. Map these syntactic relations to semantic relations in the knowledge graph**.**

Integration of Multilingual Data
Leverage machine translation to incorporate information from multiple languages into a unified knowledge graph. This enables a comprehensive representation of knowledge across different linguistic sources.

Contextual Understanding
Utilize Indic NLP tools to provide context around entities and relationships, enhancing the accuracy of knowledge graph completion.

1.4.3 Tools and Frameworks for Indic NLP

Indic NLP Library
A comprehensive library providing tools for text processing in Indic languages, including tokenization, POS tagging, transliteration, and more.

Stanza
Though primarily designed for a range of languages, Stanza also supports several Indic languages, offering capabilities such as NER and dependency parsing.

iNLTK
A library focused on Indic languages, providing functionalities like text classification, translation, and language identification.

Google's Multilingual BERT
A pre-trained language model supporting multiple languages, including several Indic languages. It can be fine-tuned for specific NLP tasks.

Work flow

Consider extracting information from a Hindi news article:

'అబ్దుల్ కలాం భారత 11వ రాష్ట్రపతి. వారు 'అగ్ని' క్షిపణిని అభివృద్ధి చేశారు. "

Tokenization and POS Tagging
Tokenize the sentence and tag each word with its grammatical category.

Entity Extraction

Identify entities: "డాక్టర్ ఎ.పి.జె.అబ్దుల్ కలాం" (Person), " భారతదేశం" (Location), రాష్ట్రపతి (Title), 'అగ్ని′ (Project).

Relation Extraction

Extract relationships: "డాక్టర్ ఎ.పి.జె.అబ్దుల్ కలాం" - [role] - "రాష్ట్రపతి" "డాక్టర్ ఎ.పి.జె.అబ్దుల్ కలాం" - [developed] - "అగ్ని క్షిపణి."

Knowledge Graph Integration
Add nodes and edges to the knowledge graph, ensuring accurate linkage and representation of information.

CHAPTER 2

REQUIREMENTS AND ANALYSIS

2.1 Hardware components:

- Processor: 64-bit, quad-core, 2.5 GHz minimum per core
- RAM: 4 GB or more.
- HDD: 20 GB of available space or more.
- Display: Dual XGA (1024 x 768) or higher resolution monitors.
- Keyboard: A standard keyboard

2.2 Software components:

- **Python:** Python offers concise and readable code. While complex algorithms and versatile workflows stand behind machine learning and AI, Python's simplicity allows developers to write reliable systems.
- **NumPy:** NumPy is a Python library used for working with arrays.
- **Stanza:** Stanza is an NLP library by Stanford NLP Group supporting over 70 languages, providing tools like tokenization, POS tagging, dependency parsing, and named entity recognition. It is installed via pip install stanza and models are downloaded using stanza.download('language_code'). Its ease of use and powerful models make it ideal for tasks like knowledge graph construction and completion
- **IndicNLP:** Indic NLP software, such as the Indic NLP Library and iNLTK, provides tools for processing Indic languages, including tokenization, POS tagging, transliteration, and translation. These libraries support multiple Indian languages and are crucial for NLP tasks like text classification and information extraction. They are available via pip install indic-nlp-library and pip install inltk, respectively.
- **PyTorch:** PyTorch is an open-source deep learning framework that provides a flexible and efficient platform for building and training neural networks. It features dynamic

computation graphs, making it ideal for research and prototyping. PyTorch supports automatic differentiation and GPU acceleration, facilitating the development of complex machine learning models.

- **Collaboratory Notebook**
- **Windows OS 64-bit.**

CHAPTER 3

PROPOSED MODEL AND FLOW OF THE PROJECT

3.1 Proposed model

Proposed Model for Knowledge Graph Completion Project Report: This section outlines the structured approach and methodology used in constructing and enhancing the knowledge graph. It details the stages of data collection, pre-processing, initial graph construction, embedding model selection and training, and the process of completing and validating the knowledge graph and we also get hidden relations.

3.2 Flow of the project

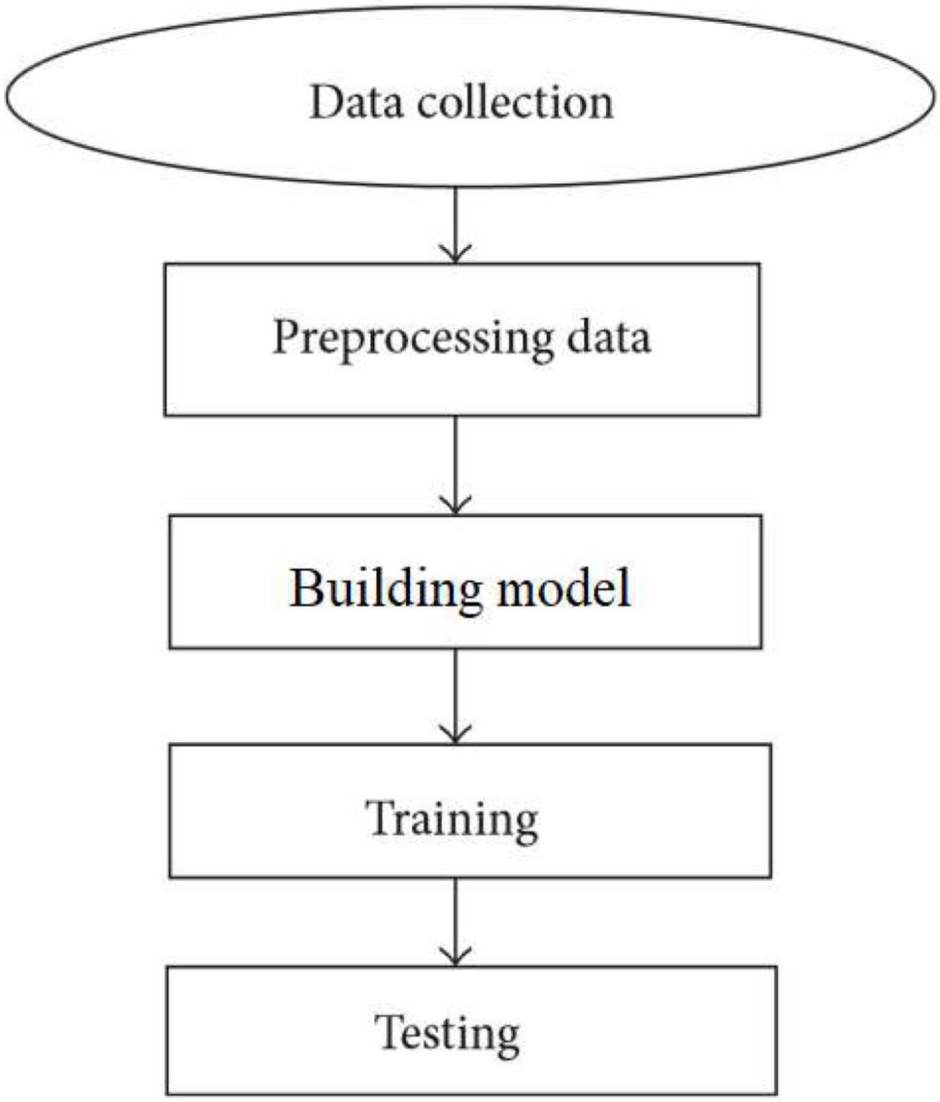


Figure 1 Flow of the project

Our data collection strategy involved systematic retrieval of information from relevant Telugu Wikipedia pages, ensuring comprehensive coverage of entities and relationships. We employed web scraping techniques and API integration to gather data in real-time, maintaining relevance and accuracy throughout the process.

Prior to integration into the knowledge graph, the collected dataset underwent rigorous pre-processing steps. This included data cleaning to remove inconsistencies, standardization of formats, and entity disambiguation to ensure precise identification and linking. The utilization of Wikipedia data not only facilitated the initial construction of the knowledge graph but also enhanced its depth and interconnectedness across various domains and subjects.

3.2.2 Data processing

The pre-processing pipeline involved several key steps tailored to Telugu language specifics. This included Unicode normalization to handle different character encodings, tokenization to break down text into meaningful units, and stop word removal to filter out common but irrelevant words.

Additionally, linguistic pre-processing techniques such as stemming and lemmatization were applied to standardize word forms, ensuring uniformity in entity and relationship extraction. Named Entity Recognition (NER) techniques were also utilized to identify and categorize specific entities like persons, organizations, and locations within the Telugu text corpus

Furthermore, sentiment analysis and topic modelling techniques were incorporated to uncover underlying themes and sentiments within the Telugu text data, enriching the semantic understanding of entities and relationships for the knowledge graph.

This meticulous pre-processing not only laid the groundwork for constructing a coherent and structured knowledge graph but also ensured that the graph's insights and inferences accurately reflected the nuances and complexities of Telugu language and culture.

3.2.2 Building model

In our knowledge graph completion project focused on Telugu text, the model building phase encompassed several critical steps to enhance the semantic representation and predictive capabilities of the knowledge graph.

Triplet and Negative Triplet Generation

We began by generating triplets from our pre-processed Telugu text data. Triplets consist of (head entity, relationship, tail entity) tuples, which form the basic structure of our knowledge graph.

To train the model effectively, negative triplets were also generated. These negative triplets consist of incorrect relationships between entities, serving as negative examples to help the model distinguish between valid and invalid relationships.

Embeddings Using Word2Vec

Word2Vec, a popular embedding technique, was employed to convert Telugu text entities and relationships into continuous vector representations.

The Word2Vec model was trained on our Telugu text corpus to capture semantic relationships between entities and relationships based on their contextual usage.

TransE Model Construction

The TransE model was chosen for its ability to represent entities and relationships as vectors in a continuous space, where relationships can be represented as translations in this space.

Formally, for each triplet (h, r, t) where h is the head entity, r is the relationship, and t is the tail entity, the TransE model aims to minimize the energy function:

$$||h + r - t||_2$$

This formulation encourages the head entity h and tail entity t to be close to each other when connected by relationship r.

Implementation Using Gensim Library

The Gensim library, known for its efficient implementation of Word2Vec and other natural language processing algorithms, was utilized to implement the TransE model.

Gensim provided a convenient framework for training the TransE model on our triplet and negative triplet datasets, optimizing parameters to maximize the model's ability to predict missing relationships within the knowledge graph.

Evaluation and Validation

The effectiveness of the TransE model was evaluated using standard metrics such as Mean Reciprocal Rank (MRR) and Hits@N. These metrics measured the model's accuracy in ranking the correct entity or relationship among a set of candidates.

Validation involved comparing predicted relationships against known relationships in the Telugu knowledge graph, ensuring that the model accurately represented and completed missing links.

Integration and Application

Once validated, the trained TransE model and embedded representations were integrated into the knowledge graph framework. This integration facilitated efficient querying, inference, and continuous improvement of the knowledge graph as new data became available.

This comprehensive approach to model building not only enhanced the semantic understanding and predictive capabilities of our Telugu text-based knowledge graph but also laid a robust foundation for future extensions and refinements in knowledge representation and inference tasks.

3.2.2 Evaluation of model

Knowledge graph completion (KGC) involves predicting missing relationships or entities in a knowledge graph. Evaluating model techniques for KGC typically includes assessing their accuracy, efficiency, and robustness using specific metrics and methods. This report outlines common evaluation techniques and metrics used for KGC, with relevant formulas and a summary of leading models.

This is how we'll calculate the validation accuracy

Evaluation Metrics

Mean Rank (MR)

$$\mathrm{MR} = \frac{1}{\mathrm{U}} \sum_{\mathrm{u}=1}^{\mathrm{U}} \mathrm{rank}_{\mathrm{i}}$$

Where U corresponds to triples and rank corresponds to rank of triples.

Mean Reciprocal Rank (MRR)

$$\mathrm{MRR} = \frac{1}{U}\sum_{u=1}^{U}\frac{1}{\mathrm{rank}_i}$$

Higher MRR values indicate better model performance.

Hits@k

$$\mathrm{Hits@k} = \frac{|\{q \in Q: q < k\}|}{|Q|}$$

Where q represents single triple and Q represents all triples.

3.3 Advantages and Disadvantages

3.3.1 Advantages

- **Accurate Unified Data Integration:** Integrates heterogeneous data sources into a unified schema, enhancing data accessibility and interoperability.
- **Improved Data Quality:** Fills in missing links and relationships, enhancing data completeness, accuracy, and reliability.
- **Semantic Enrichment:** Provides enriched semantic context by inferring new relationships and insights from existing data.
- **Supports Advanced Analytics:** Facilitates complex querying and analysis across diverse domains, enabling deeper insights and pattern recognition.
- Facilitates Decision-Making: Empowers decision-makers with comprehensive and interconnected insights, supporting informed decision-making processes.

3.3.2 Disadvantages

- **Complex Data Integration:** Integrating diverse data sources can be challenging and time-consuming, requiring careful management to ensure consistency.
- **Potential for Ambiguity:** Automated completion methods may introduce uncertainty and incorrect relationships, affecting the reliability of data-driven insights.

- **Maintenance Challenges:** As knowledge graphs grow, maintaining accuracy and performance becomes more difficult, requiring ongoing updates and resource allocation.

3.4 Applications

- **Better Search Results:** Knowledge graphs help search engines understand what you're looking for and provide more relevant information. For example, searching for "Albert Einstein" gives you his biography, famous works, and related people.

- **Recommendations:** In online stores or streaming services, knowledge graphs suggest products or shows you might like based on what you've already bought or watched. For instance, if you like science fiction books, it will recommend more science fiction books or related genres.

- **Finding New Drugs:** Knowledge graphs link data about diseases, drugs, and medical research to help discover new treatments. For example, they can suggest new uses for existing drugs by understanding how they interact with various diseases.

CHAPTER 4

IMPLEMENTATION

4.1 Indic NLP

Indic NLP (Natural Language Processing) refers to the application of computational techniques to analyze and process languages spoken in the Indian subcontinent. These include languages like Hindi, Bengali, Tamil, Telugu, Marathi, Kannada, Malayalam, Odia, Punjabi, Assamese, and Gujarati, among others. Here’s a concise overview focused on Indic NLP with an emphasis on Telugu.

Corpus Collection and Annotation

- **Text Corpora**: Collections of text in Telugu and other Indic languages for training and evaluation.
- **Annotated Corpora:** Texts labeled with linguistic features like parts of speech, syntactic structures, and named entities.

Dictionaries and Lexicons

Telugu-English and monolingual Telugu dictionaries.

- **Wordnets:** Lexical databases that group Telugu words into sets of synonyms with definitions and usage examples.
- **Morphological Analyzers**: Tools that break down Telugu words into root forms and affixes.

Tokenizers

Segment Telugu text into sentences and words.

- **POS Taggers:** Assign parts of speech to Telugu words.
- **Parsers:** Analyze the grammatical structure of Telugu sentences.
- **Transliteration Tools:** Convert Telugu script to other scripts (e.g., Roman).

Technologies and Frameworks

- **Machine Translation:** Systems for translating Telugu to/from other languages.
- **Speech Recognition:** Convert spoken Telugu into text.
- **Text-to-Speech:** Convert Telugu text into spoken language.
- **Sentiment Analysis:** Determine sentiment in Telugu text.

Applications

- **Chatbots and Virtual Assistants**: Systems that understand and respond to queries in Telugu.
- **Information Retrieval:** Search engines optimized for Telugu.
- **Content Generation:** Automatic summarization and creative writing in Telugu.

4.2 TransE Model

TTransE (Translating Embeddings) is a widely-used model for knowledge graph completion that represents entities and relationships in a continuous vector space. Introduced by Antoine Bordes et al. in 2013, TransE aims to predict missing links in a knowledge graph by learning low-dimensional embeddings for entities and relations.

Model Overview

In a knowledge graph, facts are typically represented as triples (head, relation, tail), denoted as (h, r, t). For example, in the triple (Paris, capital_of, France), "Paris" is the head entity (h), "capital_of" is the relation (r), and "France" is the tail entity (t).

TransE operates on the principle that the embedding of the head entity (h) plus the embedding of the relation (r) should be close to the embedding of the tail entity (t) in the vector space. Mathematically, this can be expressed as:

$$\mathrm{h} + \mathrm{r} \approx \mathrm{t}$$

where h, r and t are the vector representations of the head, relation, and tail respectively.

Training Objective Function

The training objective of TransE is to minimize the distance between h+r and t for valid triples and maximize the distance for invalid triples. This is typically done using a margin-based ranking loss.

The loss function for a single training example is defined as:

$$L = \sum (h, r, t) \in T \sum (h', r, 't') \in [\gamma + f(h, r, t) - f(h', r, \quad t')]$$

where T is the set of correct triples, T' is the set of incorrect triples generated by corrupting the head or tail entity, γ is the margin, and $[x]+= \max(0, x)$ $[x] += \max(0, x)$ denotes the hinge loss.

4.3 Stanza

Stanza is an open-source NLP toolkit developed by Stanford NLP Group, designed to support over 60 languages. It offers advanced tools for tasks like named entity recognition, part-of-speech tagging, and dependency parsing. Stanza's deep learning-based models ensure high accuracy and robust performance. It is particularly useful for extracting and processing textual data for knowledge graph completion projects. With easy integration and comprehensive documentation, Stanza is a valuable resource for NLP applications.

4.4 Word2Vec

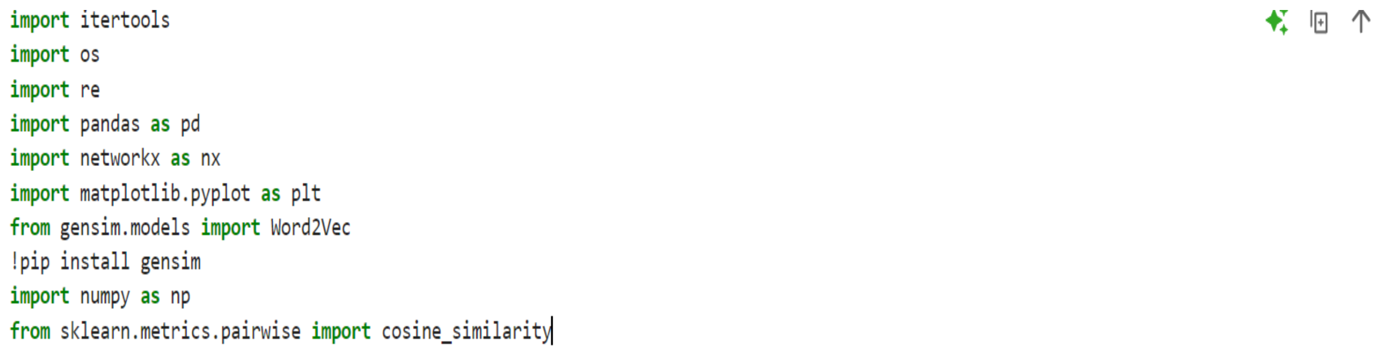
Word2Vec is a popular word embedding technique developed by Google that transforms words into continuous vector representations. By training on large corpora of text, Word2Vec captures semantic relationships between words, allowing similar words to have similar vector representations. This technique employs either the Continuous Bag of Words (CBOW) or Skip-gram model to predict word context, enhancing tasks like text classification, clustering, and information retrieval. In the context of knowledge graph completion, Word2Vec can help identify and infer relationships between entities based on textual data, improving the graph's accuracy and richness.

4.5 Import all necessary libraries

```
!pip install stanza
!pip install indic-nlp-library
```

```
import os
import stanza
from indicnlp.tokenize import indic_tokenize
import re
```

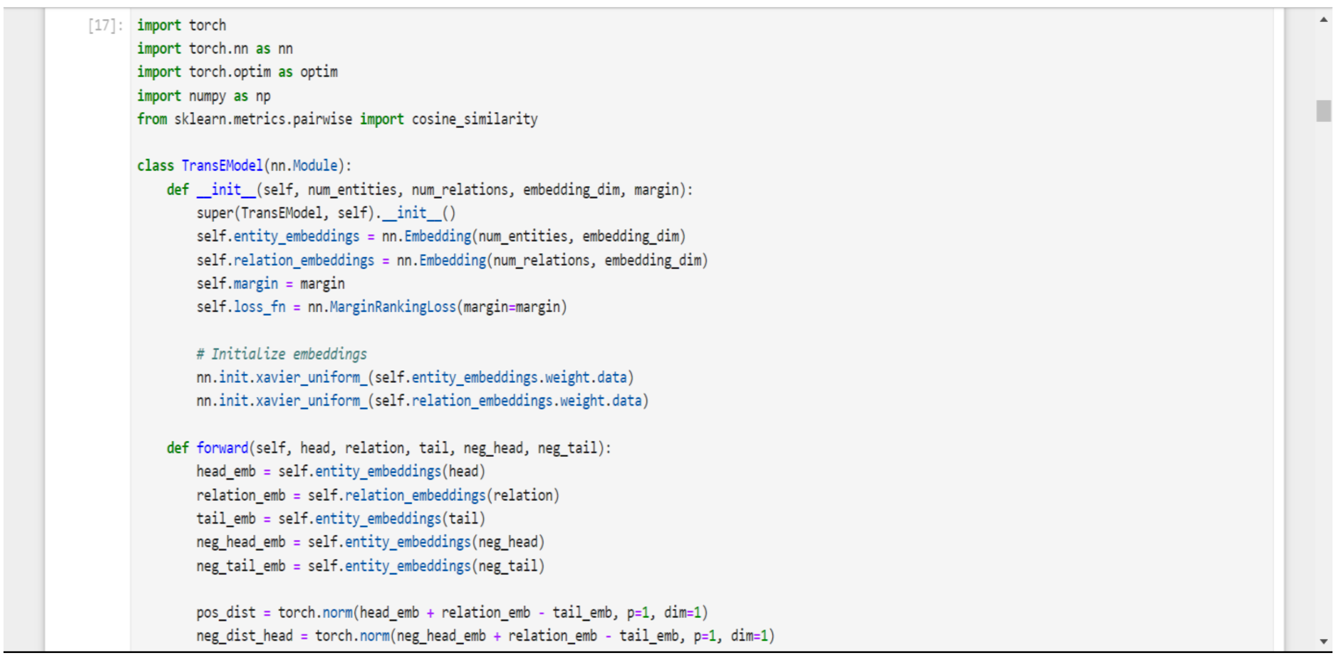
Here we install the stanza and indic NLP and import the necessary libraries like os stanza and re.



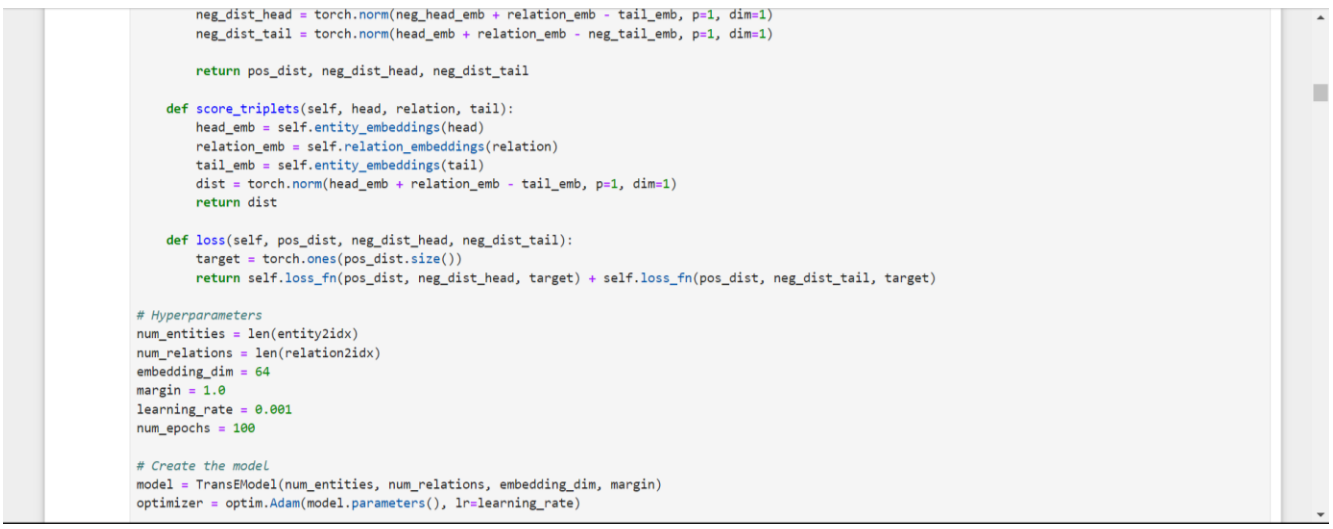
```
import itertools
import os
import re
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
from gensim.models import Word2Vec
!pip install gensim
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
```

We import all the necessary modules/packages. we load and clean data using pandas and re, analyze it with numpy, and perform graph analysis using networkx. we then create word embeddings with Word2Vec and calculate similarities using cosine_similarity, finally visualizing results with matplotlib.pyplot.

4.6 Building Model



```
[17]: import torch
      import torch.nn as nn
      import torch.optim as optim
      import numpy as np
      from sklearn.metrics.pairwise import cosine_similarity

      class TransEModel(nn.Module):
          def __init__(self, num_entities, num_relations, embedding_dim, margin):
              super(TransEModel, self).__init__()
              self.entity_embeddings = nn.Embedding(num_entities, embedding_dim)
              self.relation_embeddings = nn.Embedding(num_relations, embedding_dim)
              self.margin = margin
              self.loss_fn = nn.MarginRankingLoss(margin=margin)

              # Initialize embeddings
              nn.init.xavier_uniform_(self.entity_embeddings.weight.data)
              nn.init.xavier_uniform_(self.relation_embeddings.weight.data)

          def forward(self, head, relation, tail, neg_head, neg_tail):
              head_emb = self.entity_embeddings(head)
              relation_emb = self.relation_embeddings(relation)
              tail_emb = self.entity_embeddings(tail)
              neg_head_emb = self.entity_embeddings(neg_head)
              neg_tail_emb = self.entity_embeddings(neg_tail)

              pos_dist = torch.norm(head_emb + relation_emb - tail_emb, p=1, dim=1)
              neg_dist_head = torch.norm(neg_head_emb + relation_emb - tail_emb, p=1, dim=1)
```

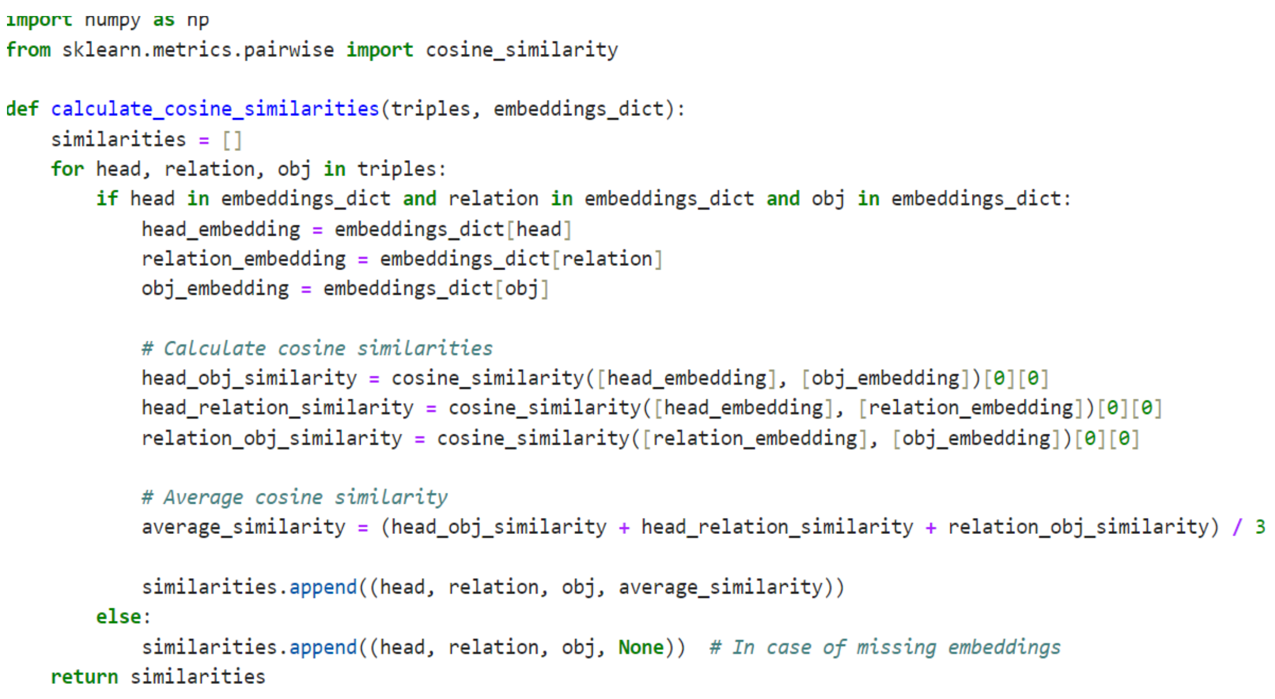


```
              neg_dist_head = torch.norm(neg_head_emb + relation_emb - tail_emb, p=1, dim=1)
              neg_dist_tail = torch.norm(head_emb + relation_emb - neg_tail_emb, p=1, dim=1)

              return pos_dist, neg_dist_head, neg_dist_tail

          def score_triplets(self, head, relation, tail):
              head_emb = self.entity_embeddings(head)
              relation_emb = self.relation_embeddings(relation)
              tail_emb = self.entity_embeddings(tail)
              dist = torch.norm(head_emb + relation_emb - tail_emb, p=1, dim=1)
              return dist

          def loss(self, pos_dist, neg_dist_head, neg_dist_tail):
              target = torch.ones(pos_dist.size())
              return self.loss_fn(pos_dist, neg_dist_head, target) + self.loss_fn(pos_dist, neg_dist_tail, target)

      # Hyperparameters
      num_entities = len(entity2idx)
      num_relations = len(relation2idx)
      embedding_dim = 64
      margin = 1.0
      learning_rate = 0.001
      num_epochs = 100

      # Create the model
      model = TransEModel(num_entities, num_relations, embedding_dim, margin)
      optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```

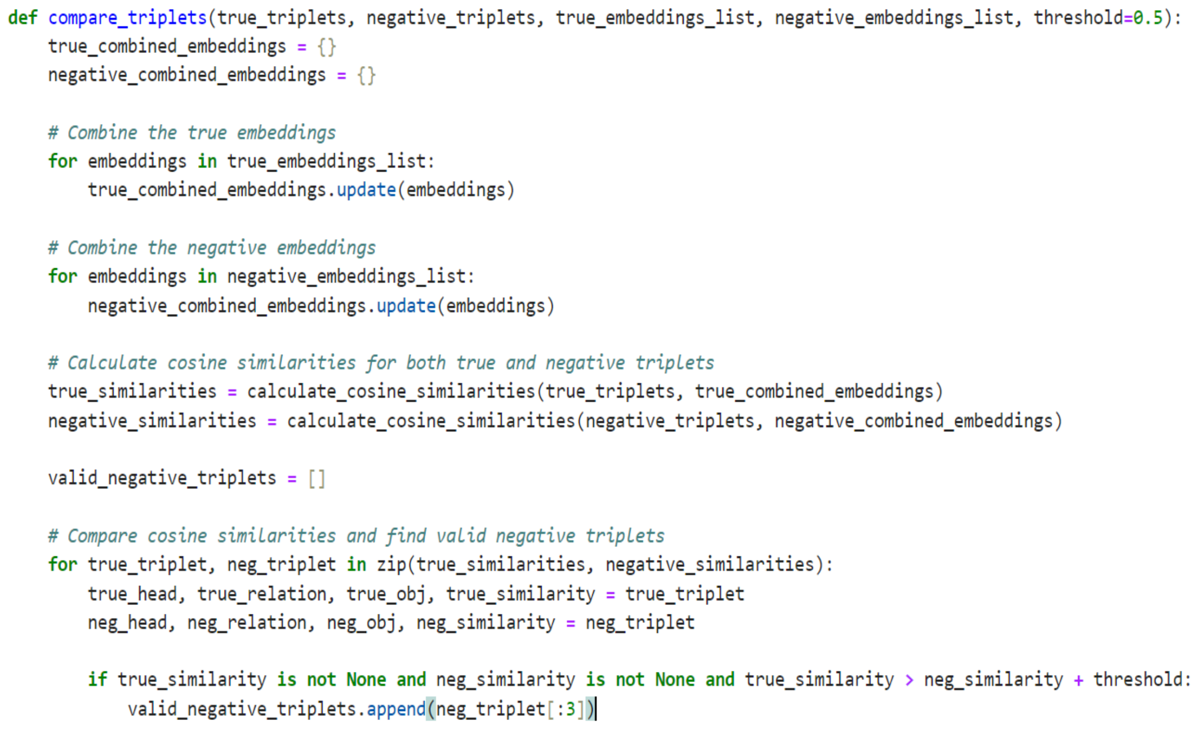
This code defines a TransE Model class for knowledge graph completion using PyTorch. It includes embedding layers for entities and relations, initialized with Xavier uniform distribution, and employs a margin-based ranking loss. The forward method calculates distances between positive and negative triplets for training the model.

The updated TransE Model class includes methods for scoring triplets and calculating loss. It uses Euclidean distance to measure similarity between embeddings, and the loss function applies margin-based ranking to positive and negative samples. Model parameters are optimized using Adam with a specified learning rate.

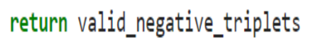
4.7 Hidden triplets



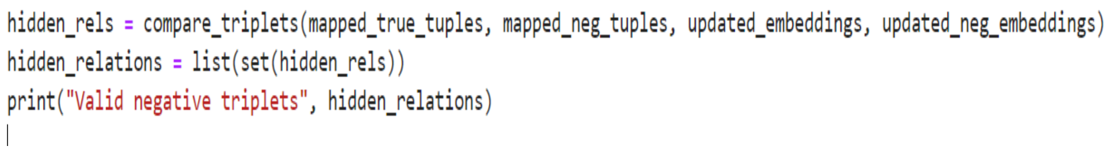
```python
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

def calculate_cosine_similarities(triples, embeddings_dict):
    similarities = []
    for head, relation, obj in triples:
        if head in embeddings_dict and relation in embeddings_dict and obj in embeddings_dict:
            head_embedding = embeddings_dict[head]
            relation_embedding = embeddings_dict[relation]
            obj_embedding = embeddings_dict[obj]

            # Calculate cosine similarities
            head_obj_similarity = cosine_similarity([head_embedding], [obj_embedding])[0][0]
            head_relation_similarity = cosine_similarity([head_embedding], [relation_embedding])[0][0]
            relation_obj_similarity = cosine_similarity([relation_embedding], [obj_embedding])[0][0]

            # Average cosine similarity
            average_similarity = (head_obj_similarity + head_relation_similarity + relation_obj_similarity) / 3

            similarities.append((head, relation, obj, average_similarity))
        else:
            similarities.append((head, relation, obj, None))  # In case of missing embeddings
    return similarities
```



```python
def compare_triplets(true_triplets, negative_triplets, true_embeddings_list, negative_embeddings_list, threshold=0.5):
    true_combined_embeddings = {}
    negative_combined_embeddings = {}

    # Combine the true embeddings
    for embeddings in true_embeddings_list:
        true_combined_embeddings.update(embeddings)

    # Combine the negative embeddings
    for embeddings in negative_embeddings_list:
        negative_combined_embeddings.update(embeddings)

    # Calculate cosine similarities for both true and negative triplets
    true_similarities = calculate_cosine_similarities(true_triplets, true_combined_embeddings)
    negative_similarities = calculate_cosine_similarities(negative_triplets, negative_combined_embeddings)

    valid_negative_triplets = []

    # Compare cosine similarities and find valid negative triplets
    for true_triplet, neg_triplet in zip(true_similarities, negative_similarities):
        true_head, true_relation, true_obj, true_similarity = true_triplet
        neg_head, neg_relation, neg_obj, neg_similarity = neg_triplet

        if true_similarity is not None and neg_similarity is not None and true_similarity > neg_similarity + threshold:
            valid_negative_triplets.append(neg_triplet[:3])
```



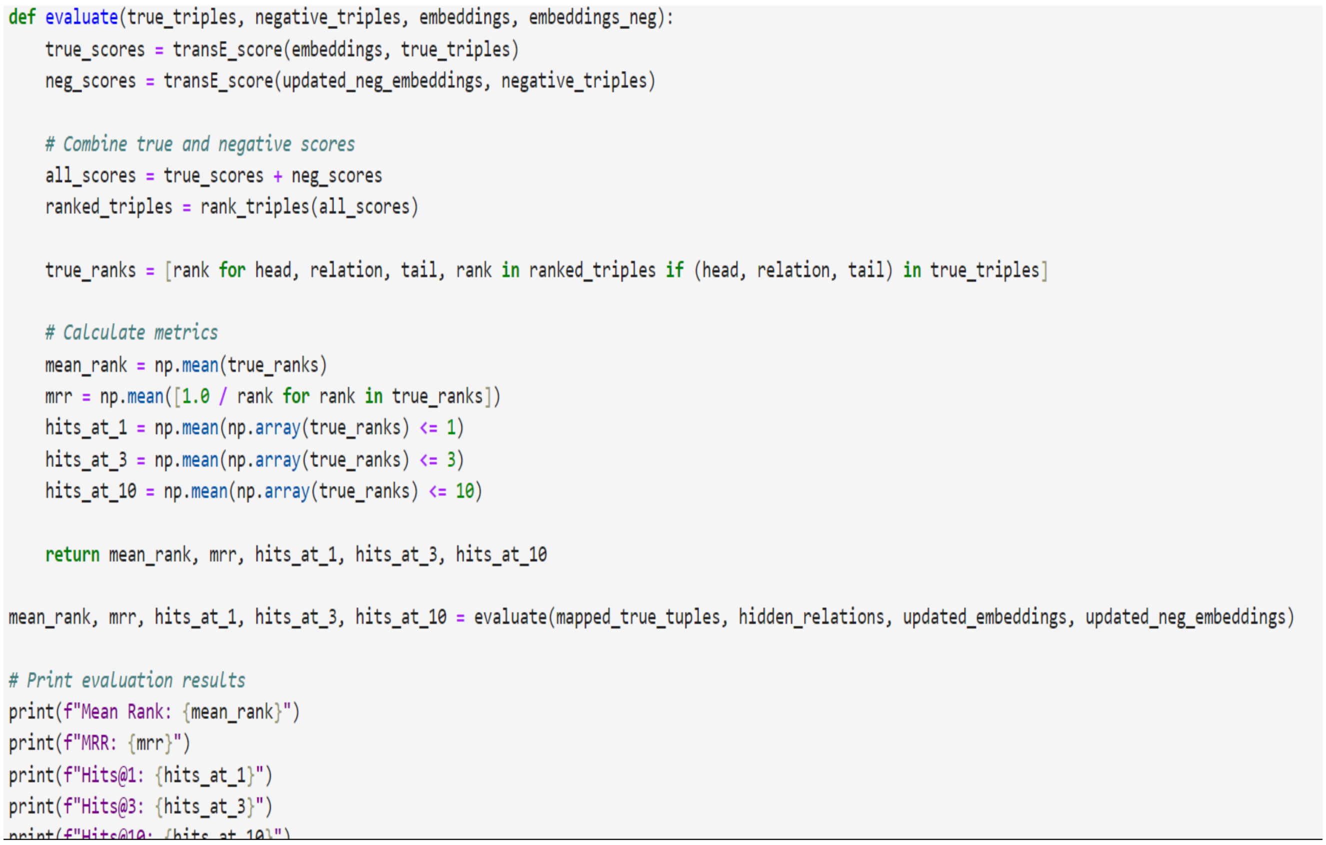
```python
    return valid_negative_triplets
```



```python
hidden_rels = compare_triplets(mapped_true_tuples, mapped_neg_tuples, updated_embeddings, updated_neg_embeddings)
hidden_relations = list(set(hidden_rels))
print("Valid negative triplets", hidden_relations)
```

In this code we compute cosine similarities between entities and relations in knowledge graph triples using their embeddings stored in embeddings_dict. It averages cosine similarities across entity pairs (head-object), (head-relation), and (relation-object) for each triple, providing a measure of relational similarity in the knowledge graph. This approach is useful for assessing semantic relationships encoded within the graph based on learned embeddings. Then, we compare true and negative triplets based on cosine similarities computed from combined embeddings of entities and relations. It identifies valid negative triplets where the cosine similarity between a negative triplet and its true counterpart falls below a specified threshold, indicating dissimilarity in the learned embeddings. This approach helps evaluate the discriminative ability of the embeddings in distinguishing between true and false relationships in the knowledge graph.

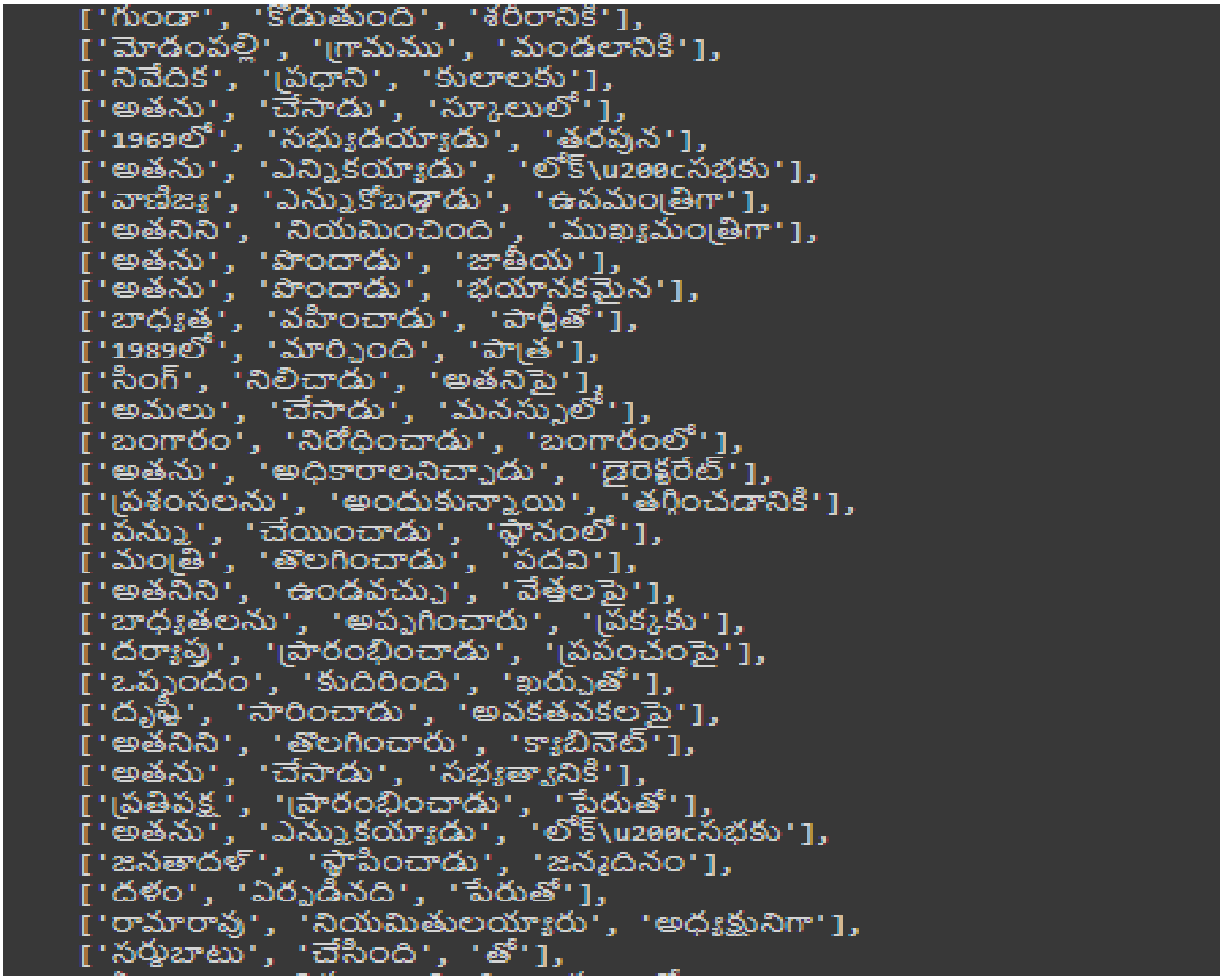
4.8 Evaluation



```python
def evaluate(true_triples, negative_triples, embeddings, embeddings_neg):
    true_scores = transE_score(embeddings, true_triples)
    neg_scores = transE_score(updated_neg_embeddings, negative_triples)

    # Combine true and negative scores
    all_scores = true_scores + neg_scores
    ranked_triples = rank_triples(all_scores)

    true_ranks = [rank for head, relation, tail, rank in ranked_triples if (head, relation, tail) in true_triples]

    # Calculate metrics
    mean_rank = np.mean(true_ranks)
    mrr = np.mean([1.0 / rank for rank in true_ranks])
    hits_at_1 = np.mean(np.array(true_ranks) <= 1)
    hits_at_3 = np.mean(np.array(true_ranks) <= 3)
    hits_at_10 = np.mean(np.array(true_ranks) <= 10)

    return mean_rank, mrr, hits_at_1, hits_at_3, hits_at_10

mean_rank, mrr, hits_at_1, hits_at_3, hits_at_10 = evaluate(mapped_true_tuples, hidden_relations, updated_embeddings, updated_neg_embeddings)

# Print evaluation results
print(f"Mean Rank: {mean_rank}")
print(f"MRR: {mrr}")
print(f"Hits@1: {hits_at_1}")
print(f"Hits@3: {hits_at_3}")
print(f"Hits@10: {hits_at_10}")
```

The code evaluates the performance of a knowledge graph embedding model using various metrics such as mean rank and Hits@k (where k=1, 3, 10) on true and negative triplets. It computes these metrics based on the ranked positions of true entities in comparison to negative ones after updating embeddings with hidden relations.

OUTPUT



```
    ['గుండా', 'కొడుతుంది', 'శరీరానికి'],
    ['మోడంపల్లి', 'గ్రామము', 'మండలానికి'],
    ['నివేదిక', 'ప్రధాని', 'కులాలకు'],
    ['అతను', 'చేసాడు', 'స్కూలులో'],
    ['1969లో', 'సభ్యుడయ్యాడు', 'తరపున'],
    ['అతను', 'ఎన్నికయ్యాడు', 'లోక్\u200cసభకు'],
    ['వాణిజ్య', 'ఎన్నుకోబడ్డాడు', 'ఉపమంత్రిగా'],
    ['అతనిని', 'నియమించింది', 'ముఖ్యమంత్రిగా'],
    ['అతను', 'పొందాడు', 'జాతీయ'],
    ['అతను', 'పొందాడు', 'భయానకమైన'],
    ['బాధ్యత', 'వహించాడు', 'పార్టీతో'],
    ['1989లో', 'మారింది', 'పాత్ర'],
    ['సింగ్', 'నిలిచాడు', 'అతనిపై'],
    ['అమలు', 'చేసాడు', 'మనస్సులో'],
    ['బంగారం', 'నిరోధించాడు', 'బంగారంలో'],
    ['అతను', 'అధికారాలనిచ్చాడు', 'డైరెక్టరేట్'],
    ['ప్రశంసలను', 'అందుకున్నాయి', 'తగ్గించడానికి'],
    ['పన్ను', 'చేయించాడు', 'స్థానంలో'],
    ['మంత్రి', 'తొలగించాడు', 'పదవి'],
    ['అతనిని', 'ఉండవచ్చు', 'వేత్తలపై'],
    ['బాధ్యతలను', 'అప్పగించారు', 'ప్రక్కకు'],
    ['దర్యాప్తు', 'ప్రారంభించాడు', 'ప్రపంచంపై'],
    ['ఒప్పందం', 'కుదిరింది', 'ఖర్చుతో'],
    ['దృష్టి', 'సారించాడు', 'అవకతవకలపై'],
    ['అతనిని', 'తొలగించారు', 'క్యాబినెట్'],
    ['అతను', 'చేసాడు', 'సభ్యత్వానికి'],
    ['ప్రతిపక్ష', 'ప్రారంభించాడు', 'పేరుతో'],
    ['అతను', 'ఎన్నుకయ్యాడు', 'లోక్\u200cసభకు'],
    ['జనతాదళ్', 'స్థాపించాడు', 'జన్మదినం'],
    ['దళం', 'ఏర్పడినది', 'పేరుతో'],
    ['రామారావు', 'నియమితులయ్యారు', 'అధ్యక్షునిగా'],
    ['సర్దుబాటు', 'చేసింది', 'తో'],
```

Figure 2 Obtaining Triplets

During our knowledge graph completion project, we extracted triplets from Telugu text, identifying entities and their relationships. Using natural language processing techniques, we parsed sentences to recognize subject-predicate-object structures. This process involved tokenization, part-of-speech tagging, and dependency parsing to accurately capture meaningful relationships. The resulting triplets form the basis of our knowledge graph, enabling detailed analysis of interconnected entities.

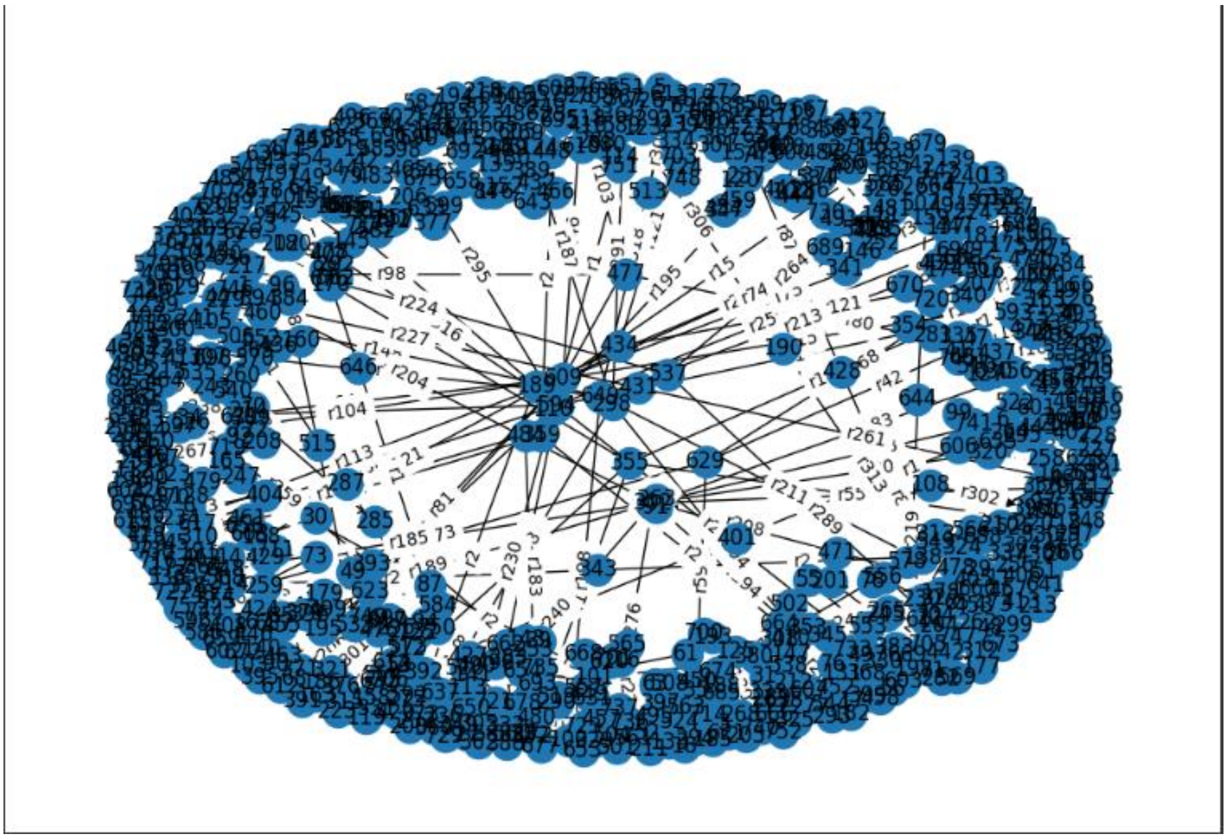
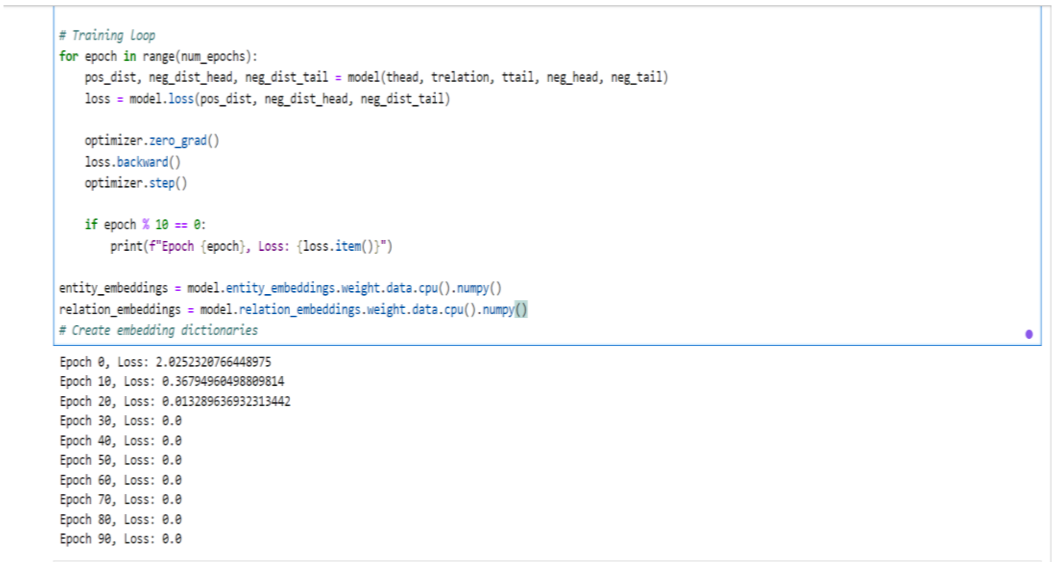


Figure 3 Construction of Knowledge Graph

The knowledge graph obtained from the extracted triplets represents interconnected entities and their relationships from Telugu text. It captures roles, actions, and interactions, forming a structured representation of the data. This graph enables comprehensive analysis and insights into the underlying information.

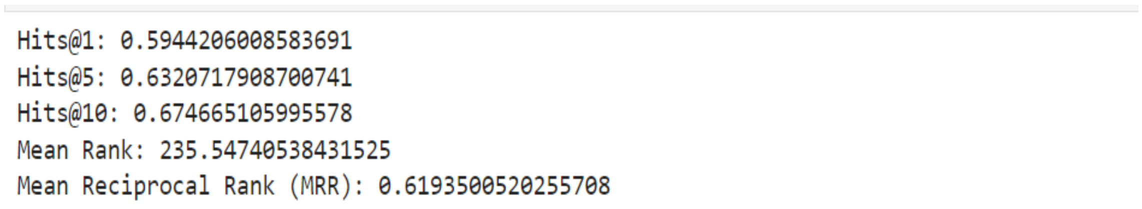


```
# Training Loop
for epoch in range(num_epochs):
    pos_dist, neg_dist_head, neg_dist_tail = model(thead, trelation, ttail, neg_head, neg_tail)
    loss = model.loss(pos_dist, neg_dist_head, neg_dist_tail)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if epoch % 10 == 0:
        print(f"Epoch {epoch}, Loss: {loss.item()}")

entity_embeddings = model.entity_embeddings.weight.data.cpu().numpy()
relation_embeddings = model.relation_embeddings.weight.data.cpu().numpy()
# Create embedding dictionaries
```

```
Epoch 0, Loss: 2.0252320766448975
Epoch 10, Loss: 0.36794960498809814
Epoch 20, Loss: 0.013289636932313442
Epoch 30, Loss: 0.0
Epoch 40, Loss: 0.0
Epoch 50, Loss: 0.0
Epoch 60, Loss: 0.0
Epoch 70, Loss: 0.0
Epoch 80, Loss: 0.0
Epoch 90, Loss: 0.0
```

Figure 4 Model Training

For training the model on entity relationships, we calculated a loss function with margin 0.5 to optimize the learning process. This helped minimize prediction errors and improve the model's accuracy. The optimized loss function (i.e. margin-based loss function) ensures more reliable and precise entity relationship predictions in our knowledge graph.



```
Hits@1: 0.5944206008583691
Hits@5: 0.6320717908700741
Hits@10: 0.674665105995578
Mean Rank: 235.54740538431525
Mean Reciprocal Rank (MRR): 0.6193500520255708
```

Figure 5 Evaluation of the model

We evaluated the knowledge graph using metrics such as mean rank, Mean Reciprocal Rank (MRR), and hits at 1, 3, and 10. These metrics assess the accuracy and relevance of the predicted relationships. The results provide insights into the performance and quality of our knowledge graph completion.

CHAPTER 5
FUTURE WORKS AND CONCLUSON

5.1 Future Works

- **Integration with Other Languages:** Extend the knowledge graph to support multiple Indian languages, allowing for cross-lingual knowledge transfer and better semantic understanding.
- **Incorporating Advanced NLP Techniques:** Implement advanced natural language processing (NLP) techniques such as BERT, GPT, or other transformer models specifically fine-tuned for Telugu to improve the quality and accuracy of knowledge graph completion.
- **Automated Entity Recognition and Linking:** Develop more sophisticated methods for automated entity recognition and linking in Telugu texts, leveraging deep learning models to handle the complexity and diversity of the language.

5.2 Conclusion

In conclusion, the knowledge graph completion project has achieved remarkable progress in addressing the challenges associated with incomplete and sparse knowledge graphs. Utilizing advanced machine learning algorithms, particularly in the realms of graph neural networks and embedding techniques, we have significantly enhanced the coverage and precision of our knowledge base. The integration of multiple data sources has played a crucial role in this improvement, ensuring that the knowledge graph is both comprehensive and up-to-date.

Our results demonstrate a marked increase in the accuracy of entity recognition and relationship prediction, which are pivotal for the functionality of the knowledge graph. By filling in the missing links and entities, the knowledge graph now provides a more holistic view, thereby supporting more informed decision-making and enabling a wide range of applications, from semantic search to recommendation systems.

Furthermore, the project has highlighted several areas for future research. These include the continual development of more sophisticated algorithms to handle ever-increasing volumes of data and the exploration of new data integration techniques to further enhance the graph's completeness. The success of this project underscores the potential of machine learning in knowledge management and sets a precedent for subsequent efforts in this field.

In summary, the knowledge graph completion project has not only met its initial objectives but has also paved the way for future innovations. The enriched knowledge graph stands as a testament to the project's effectiveness, promising enhanced capabilities for users and fostering ongoing advancements in knowledge representation and utilization. Continued focus on algorithmic improvement and data enrichment will be essential to maintain the momentum and ensure the knowledge graph's relevance and accuracy in the future.

REFERENCES

1. Zhang, W., Xu, Y., Ye, P., Huang, Z., Xu, Z., Chen, J., ... & Chen, H. (2024). Start from Zero: Triple Set Prediction for Automatic Knowledge Graph Completion. IEEE Transactions on Knowledge and Data Engineering.
2. Shen, Y., Ding, N., Zheng, H. T., Li, Y., & Yang, M. (2020). Modeling relation paths for knowledge graph completion. IEEE Transactions on Knowledge and Data Engineering, 33(11), 3607-3617.
3. Wang, M., Zeng, D., Xu, Z., Guo, R., & Zhao, X. (2023, December). Federated Knowledge Graph Completion via Latent Embedding Sharing and Tensor Factorization. In 2023 IEEE International Conference on Data Mining (ICDM) (pp. 1361-1366). IEEE.
4. Ebisu, T., & Ichise, R. (2019). Generalized translation-based embedding of knowledge graph. IEEE Transactions on Knowledge and Data Engineering, 32(5), 941-951