**DOMAIN-DEPENDENT KNOWLEDGE GRAPH COMPARISON THROUGH SELECTIONAL RESTRICTIONS**

**Knowledge Engineering**

**Projects 2022-2023**

Artificial Intelligence

Lsei [Li (lesi.li@studio.unibo.it)](mailto:Li(lesi.li@studio.unibo.it))

**Contents**

1. **[Introduction](#_30j0zll)** **3**
2. **Data acquisition and preparation** **3**
   1. Get knowledge graph data 3
   2. Data cleaning and preprocessing
3. **Parse RDF graph data** **4**
   1. parse data 4
   2. purposes and significance 5
4. **Analysing information** **6**
   1. Get a list of frame instances............................................................................6
   2. Analyze relationships between frame instances 6
   3. compare the attribute values and associated relationships 7
   4. most frequently frame\_occurrences 8
5. **Frequency information** **8**
   1. `frequency\_dict` and `metrics\_dict` 8
   2. explaination 9
   3. Find the frequency dictionary for a specific frame instance 10
   4. Find the frame instance with the highest subject frequency 11
6. **Types imforemation** **11**
   1. Frame Occurrence Type 11
   2. Number of occurrences of different types 11
   3. Type preference for framework instances 12
7. **Similarity calculation** **12**
   1. Simple similarity calculation method 12
   2. Improved similarity measure method 13
   3. Using Word2Vec model 13
8. **natural language explanation** **13**
   1. Generated natural language explanation 13
9. **[Conclusion](#_2u6wntf)** **14**

**1 Introduction**

The purpose of this project is to develop a pipeline to compare and analyze MusicBO knowledge graphs. Music BO is a polyphonic music project that automatically generates a framework-based Knowledge Graph (KG) about the role of music in the city of Bologna. Each knowledge graph is composed of text data from different authors by applying a text-to-KG pipeline.

This project will use knowledge graphs obtained from different authors to develop comparison pipelines for domain-specific selection constraints. The selection constraints are probabilistic constraints with respect to the frame parameters. Therefore, the analysis must employ knowledge graphs from different authors to highlight their differences based on frame constraints.

Analysis results must be leveraged to generate natural language explanations (e.g., proficiency questions) through the use of large pre-trained language models. The resulting explanations will provide additional insights into the topics discussed by the knowledge graph, as well as comparisons highlighting differences in topics discussed by different authors.

**2 Data acquisition and preparation**

**2.1 Get knowledge graph data**

Release\_190223\_EN\_ITA.zip from the MusicBO Knowledge Graph's GitHub repository https://github.com/polifonia-project/musicbo-knowledge-graph/tree/main/data/knowledge\_graphs\_releases

Downloaded the data set, which contains more than 7,000 nq files.

Since there are too many files, I extracted some of them.

**2.2 Data cleaning and preprocessing**

**1. Process Each Line of Data:**

Use the readlines() method to read the contents of the file and store each line in a list called ‘text’.

For each line in the ‘text’ list, perform the following operations:

- split the line by ‘>’using `split('>')` to extract the subject, predicate, object, and possibly content with multiple `<`.

- iterate through the split elements (index `i`) and perform different processing based on its position:

- If `i` is 2 and it contains double quotes ("):

- extract the content from the object by splitting and removing the last word, and then add it to a list called `line\_original`.

- add a period (.) to indicate the end of the object.

- Otherwise, check if the current element is a URL (validated using `validators.url()` function), and if it's not a URL, you add it to a list called `line\_parsed`.

**2. Recompose Line Data:**

- In the above processing steps, i split and store each line's subject, predicate, object, and period (if applicable) in the `line\_original` and `line\_parsed` lists.

- Next, reassemble these processed elements into a new string `new\_line\_str`, which includes the subject, predicate, object, and a period, forming a line in N-Quads format.

**3. Add New Line to `new\_text` List:**

-add each line's `new\_line\_str` to a list named `new\_text`, which will be used for further processing.

**4. Remove Extra Period:**

- After processing all lines, check if `new\_text` is not empty (to ensure it's not an empty list) and if the last line consists solely of a period.

- If the last line is just a period, you remove this period from `new\_text` to ensure there are no extra periods at the end of the N-Quads formatted file.

**5. Rewrite to Original File:**

- Finally, you reassemble the processed text in `new\_text` into a string called `new\_text\_str`.

- use the `open()` function to open the original file in write mode and write `new\_text\_str` back to the original file, overwriting the existing content, completing the data preprocessing.

The main purpose of this script is to clean and format N-Quads formatted knowledge graph data to ensure it adheres to the expected format requirements, making it suitable for subsequent analysis and processing. The processing steps include splitting, extraction, URL validation, reassembly, and writing back to the original file.

**3 Parse RDF graph data.**

**3.1 parse data.**

Class Initialization:

My `Parser` class is designed for RDF graph data analysis. During initialization, I create a `Namespace` object for handling various namespaces. I also specify a directory path containing RDF graph data files to be loaded and processed later.

Creating and Returning an RDFLib Graph Object (graph\_init):

This method takes a file path as a parameter and is used to create and parse an RDFLib graph (`Graph`) object. I use the `bind` method on the graph to associate different namespaces with it, facilitating their use in subsequent queries.

Recursively Getting Frame Instance Lists (get\_frame\_occ\_list):

This method is employed to recursively search for frame instances (frame occurrences) in the graph. It locates instances related to a specified class or object and returns a list containing all found instances.

Converting URI References to Names (uriref2name):

This method transforms URI references (URIRed) into names and returns the result. It parses URIs and finds the corresponding namespace, returning the namespace prefix followed by the name.

Obtaining Frequencies of Different Frame Instances, Predicates, and Objects (get\_frequencies\_dict):

This method is used to tally the occurrences of different frame instances, predicates, and objects in the graph. I iterate through RDF graph files in the specified directory and record the occurrences of instances and predicates.

Calculating Support and Confidence (calculate\_metrics):

This method calculates support and confidence. Support reflects the frequency of an event, while confidence represents the conditional probability of an event.

Getting Metrics Dictionary for Different Elements (get\_metrics\_dict):

This method calculates various metrics, such as support and confidence, for different elements. I compute metrics based on the frequency of occurrences and conditional probabilities and add these values to a metrics dictionary.

Generalizing Different Types (generalize\_common):

This method generalizes different types (objects) into types with a common ancestor by recursively accessing the WordNet hierarchy. I use SPARQL queries to find superclasses in the RDF graph and generalize the type hierarchy to a common ancestor.

'''

WordNet alignment extraction: For each type in the types list, the code tries to find an alignment with an equivalent class in WordNet (via the owl.equivalentClass relationship). If alignment is found, this WordNet type is added to the wn\_types list.

Hierarchical path extraction: For each type in wn\_types, the code performs a SPARQL query to obtain its path in the WordNet hierarchy, specifically by looking for its hypernyms (wn.hyponymOf relationship). This process is repeated k times, or until no more hypernyms can be found.

Finding common ancestors: By comparing WordNet hierarchical paths of different types, the code attempts to find a common ancestor type. This is done by checking each path in the generalizations list, looking for a type that appears in all paths.

'''

Generalizing Different Types (generalize):

This method generalizes different types (objects) to multiple generalized types by recursively accessing the WordNet hierarchy, but with a lower degree of generalization. I look for and record the generalized types for each type.

Cleaning the Dictionary (clean\_dictionary):

This method is used to clean entries in the metrics dictionary, especially those that do not meet specific criteria, particularly those that are not generalized.

Printing Statistics (print\_statistics):

This method prints statistics about the most common types and combinations. It can be used to analyze the most frequently occurring elements and combinations.

Getting Frequency Information for Different Types (get\_types):

This method retrieves frequency information for different types from the metrics dictionary, with an option to include zero frequencies.

Calculating Type Preference for Specific Frame Instances (get\_preference):

This method computes the type preference for a specific frame instance, measured by calculating the entropy of the type distribution.

**3.2 purposes and significance**

Data Preparation: I parse RDF-formatted knowledge graph data to help prepare the data for subsequent analysis. It parses MusicBO knowledge graphs generated by different authors and loads them into an RDFLib graph.

Data Statistics and Probability Calculation: calculate the frequency and probability of different types of instances (e.g., different frame types, predicate types, object types, etc.) within the knowledge graph. This is valuable for understanding the frequency of elements and their probabilistic relationships in the knowledge graph, revealing differences and commonalities among knowledge graphs from different authors.

Data Generalization: I provide methods for generalizing different types (objects) into types with a common ancestor by accessing the WordNet hierarchy. This helps in better understanding and comparing relationships between different types in the analysis.

Statistics and Analysis: I offer functions to print statistical information about the most common types and combinations, aiding in understanding the primary elements and patterns within the knowledge graph. Analyzing this statistical information provides insights into themes and differences among knowledge graphs from different authors.

Generating Explanations: Ultimately, I assist in generating natural language explanations. These explanations can be used for a deeper understanding and discussion of the content within the knowledge graph. They provide insights into the topics discussed within the knowledge graph and highlight differences in topics discussed by different authors.

In summary, to assist in processing, summarizing, and analyzing MusicBO knowledge graph data.

**4 Analysing information**

**4.1 Get a list of frame instances**

Calls the get\_frame\_occ\_list method of the Parser object to extract a list of frame instances from the graph.

The frame\_occurrences list represents different concepts or entities within the MusicBO knowledge graph. These concepts can include music-related entities like events, musicians, musical works, and more. By extracting these frame occurrences from different knowledge graphs, we can compare and analyze identical or similar musical concepts across different graphs. This comparison can help us identify commonalities and differences in how these concepts are represented by different authors or sources.

**4.2 Analyze relationships between frame instances**

For example,the relationship between two frames:

1.The relationship between [https://w3id.org/polifonia/resource/MusicBO\_5\_410\_amr/manage\_1](https://w3id.org/polifonia/resource/MusicBO_5_410_amr/manage_1" \t "/Users/a123/Documents\\x/_blank) and [https://w3id.org/polifonia/resource/MusicBO\_5\_410\_amr/merge\_1](https://w3id.org/polifonia/resource/MusicBO_5_410_amr/merge_1" \t "/Users/a123/Documents\\x/_blank) is [https://w3id.org/framester/pb/localrole/deed\_accomplished](https://w3id.org/framester/pb/localrole/deed_accomplished" \t "/Users/a123/Documents\\x/_blank)

The name of this relationship indicates that there may be some kind of "behavior completion" relationship between the two frame instances, which may mean that one frame instance manages or controls another frame instance and completes a certain task.

Furthermore,Relationships between frames and other entities

2.The relationship between [https://w3id.org/polifonia/resource/MusicBO\_5\_410\_amr/general\_1](https://w3id.org/polifonia/resource/MusicBO_5_410_amr/general_1" \t "/Users/a123/Documents\\x/_blank) and [https://w3id.org/polifonia/resource/MusicBO\_5\_410\_amr/character\_2](https://w3id.org/polifonia/resource/MusicBO_5_410_amr/character_2" \t "/Users/a123/Documents\\x/_blank) is [https://w3id.org/framester/pb/localrole/common\_thing](https://w3id.org/framester/pb/localrole/common_thing" \t "/Users/a123/Documents\\x/_blank)

there may be something "in common" or "common characteristics" between the two frame instances. It might mean that the two frame instances share some common property or characteristic.

3.The relationship between [https://w3id.org/polifonia/resource/MusicBO\_5\_410\_amr/general\_1](https://w3id.org/polifonia/resource/MusicBO_5_410_amr/general_1" \t "/Users/a123/Documents\\x/_blank) and [https://w3id.org/polifonia/resource/MusicBO\_5\_410\_amr/piece\_1](https://w3id.org/polifonia/resource/MusicBO_5_410_amr/piece_1" \t "/Users/a123/Documents\\x/_blank) is [https://w3id.org/framester/pb/localrole/entities\_sharing\_arg\_1](https://w3id.org/framester/pb/localrole/entities_sharing_arg_1" \t "/Users/a123/Documents\\x/_blank)

The name of this relationship indicates that some parameters or entities may be shared between the two frame instances. This may indicate that there is some kind of parameter passing or sharing of an entity between the two frame instances.

1. The relationship between [https://w3id.org/polifonia/resource/MusicBO\_5\_410\_amr/general\_1](https://w3id.org/polifonia/resource/MusicBO_5_410_amr/general_1" \t "/Users/a123/Documents\\x/_blank) and [https://w3id.org/framester/pb/data/general-02](https://w3id.org/framester/pb/data/general-02" \t "/Users/a123/Documents\\x/_blank) is

[http://www.w3.org/1999/02/22-rdf-syntax-ns#type](http://www.w3.org/1999/02/22-rdf-syntax-ns" \l "type" \t "/Users/a123/Documents\\x/_blank)

This relationship means that the frame instance "https://w3id.org/polifonia/resource/MusicBO\_5\_410\_amr/general\_1" is an instance of type "https://w3id.org/framester/pb/data/general-02". This may indicate that "https://w3id.org/polifonia/resource/MusicBO\_5\_410\_amr/general\_1" belongs to a specific category or type, and the frame instance can be classified or tagged based on this information.

**4.3 compare the attribute values and associated relationships**

Common Entities between 0\_7\_EN.nq and 0\_9\_EN.nq:

Common Entities between 0\_7\_EN.nq and 0\_6\_ITA.nq:

Common Entities between 0\_7\_EN.nq and 0\_8\_EN.nq:

Common Entities between 0\_7\_EN.nq and 0\_8\_ITA.nq:

Common Entities between 0\_7\_EN.nq and 0\_4\_ITA.nq:

Common Entities between 0\_3\_EN.nq and 0\_9\_ITA.nq:

Common Entities between 0\_3\_EN.nq and 0\_5\_ITA.nq:

Common Entities between 0\_3\_EN.nq and 0\_1\_EN.nq:

Common Entities between 0\_3\_EN.nq and 0\_2\_ITA.nq:

Common Entities between 0\_3\_EN.nq and 0\_9\_EN.nq:

Common Entities between 0\_3\_EN.nq and 0\_6\_ITA.nq:

Entity URI: [https://w3id.org/framester/pb/data/include-91](https://w3id.org/framester/pb/data/include-91" \t "/Users/a123/Documents\\x/_blank)

Entity URI: [https://w3id.org/framester/pb/data/include-91](https://w3id.org/framester/pb/data/include-91" \t "/Users/a123/Documents\\x/_new)" is the URI of a common entity found in both graphs.

Same [http://www.w3.org/2000/01/rdf-schema#subClassOf:](http://www.w3.org/2000/01/rdf-schema" \l "subClassOf:" \t "/Users/a123/Documents\\x/_blank) [http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#Event](http://www.ontologydesignpatterns.org/ont/dul/DUL.owl" \l "Event" \t "/Users/a123/Documents\\x/_blank)

the property "subClassOf" has the same value "[http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#Event](http://www.ontologydesignpatterns.org/ont/dul/DUL.owl" \l "Event" \t "/Users/a123/Documents\\x/_new)" in both graphs for this entity.

Common Relations: {rdflib.term.URIRef('http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#Event')}

there is a common relation associated with this entity, which is also "[http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#Event](http://www.ontologydesignpatterns.org/ont/dul/DUL.owl" \l "Event" \t "/Users/a123/Documents\\x/_new)."

**4.4 most frequently frame\_occurrences**

a. Frequently occurring frame instances usually represent core concepts or entities in the project

b. Certain frame instances may represent key actions, events, or relationships in the project. These actions or relationships are important for understanding the processes and interactions within the project.

c. Frequently occurring frame instances can be used for information extraction tasks to help us extract information about specific entities or relationships from the knowledge graph.

**5 Frequency information**

**5.1 `frequency\_dict` and `metrics\_dict`**

`frequency\_dict` and `metrics\_dict` are two dictionaries used to process and store information related to RDF graph data.

1. frequency\_dict(frequency dictionary):

Role: `frequency\_dict` is used to store occurrence frequency information of different framework instances, predicates and objects. It is a multi-level dictionary that helps count the occurrences of different elements and provides data support for subsequent calculations.

Details: `frequency\_dict` includes multiple levels for storing different types of frequency information, such as the following example:

The frequency of frame occurrences.

Frequency of occurrence of a combination of frame instance and predicate.

Frequency of occurrence of combinations of framework instances, predicates, and object types.

2. metrics\_dict (metrics dictionary):

Function: `metrics\_dict` is used to calculate metrics associated with data such as support and confidence. These metrics are often used to evaluate the strength of association rules or the importance of association relationships.

Details: `metrics\_dict` generates support and confidence associated with each element (such as framework instance, predicate, object type) by calculating the frequency information in `frequency\_dict`. It is also a multi-level dictionary with a similar structure to `frequency\_dict`, but contains metric information such as support and confidence.

In short, `frequency\_dict` is used to store the original frequency of occurrence information, while `metrics\_dict` is used to calculate different metrics based on frequency information in order to analyze and understand the correlation and patterns in RDF graph data. These metrics can be used to identify important patterns, rules, or characteristics in the data.

**5.2 explaination**

https://w3id.org/framester/pb/data/express-01': This is a framework instance, appearing 1 time. It has an attribute called 'https://w3id.org/framester/pb/localrole/speaker' which appears 1 time. The attribute has information about the attribute value (object)

'https://w3id.org/polifonia/resource/MusicBO\_105\_6991\_amr-ml/Person', which appears 1 time. Some probability information is also provided, such as total probability and probability, etc.

'https://w3id.org/framester/pb/data/treat-01': This is another frame instance, appearing 2 times. It has two attributes, 'https://w3id.org/framester/pb/localrole/assumer\_of\_attribute' and 'https://w3id.org/framester/pb/localrole/entity\_treated'.

Both attributes have information about the attribute value (object) and probability.

'https://w3id.org/framester/pb/data/have-org-role-91': This is the third frame instance, appearing 4 times. It has two attributes, namely 'https://w3id.org/framester/pb/localrole/office\_holder' and 'https://w3id.org/framester/pb/localrole/title\_of\_office\_held'. Each attribute has information about Information about attribute values ​​(objects) and probabilities.

**5.3 Find the frequency dictionary for a specific frame instance**

frame\_occurrences: [https://w3id.org/framester/pb/data/treat-01](https://w3id.org/framester/pb/data/treat-01" \t "/Users/a123/Documents\\x/_blank)

subj\_frequency: 2

[https://w3id.org/framester/pb/localrole/assumer\_of\_attribute:](https://w3id.org/framester/pb/localrole/assumer_of_attribute:" \t "/Users/a123/Documents\\x/_blank)

subj\_pred\_frequency: 1

[https://w3id.org/polifonia/resource/MusicBO\_105\_6991\_amr-ml/Person:](https://w3id.org/polifonia/resource/MusicBO_105_6991_amr-ml/Person:" \t "/Users/a123/Documents\\x/_blank)

subj\_pred\_obj\_frequency: 1

subj\_pred\_obj\_total\_probability: 0.007692307692307693

subj\_pred\_obj\_probability: 1.0

None:

subj\_pred\_obj\_frequency: 0

subj\_pred\_obj\_total\_probability: 0.0

subj\_pred\_obj\_probability: 0.0

subj\_pred\_total\_probability: 0.007692307692307693

subj\_pred\_probability: 0.5

[https://w3id.org/framester/pb/localrole/entity\_treated:](https://w3id.org/framester/pb/localrole/entity_treated:" \t "/Users/a123/Documents\\x/_blank)

subj\_pred\_frequency: 1

[https://w3id.org/polifonia/resource/MusicBO\_105\_6991\_amr-ml/Situation:](https://w3id.org/polifonia/resource/MusicBO_105_6991_amr-ml/Situation:" \t "/Users/a123/Documents\\x/_blank)

subj\_pred\_obj\_frequency: 1

subj\_pred\_obj\_total\_probability: 0.007692307692307693

subj\_pred\_obj\_probability: 1.0

None:

subj\_pred\_obj\_frequency: 0

subj\_pred\_obj\_total\_probability: 0.0

subj\_pred\_obj\_probability: 0.0

subj\_pred\_total\_probability: 0.007692307692307693

subj\_pred\_probability: 0.5

subj\_total\_probability: 0.015384615384615385

subj\_probability: 0.015384615384615385

Detailed information about a specific frame instance (`https://w3id.org/framester/pb/data/treat-01`), including frequencies and predicates associated with them and their frequency and probability information:

`subj\_frequency`: The frequency of the frame instance, that is, the number of times it appears in the data, here it is 2 times.

Then, each predicate of the framework instance has related information, including:

`https://w3id.org/framester/pb/localrole/assumer\_of\_attribute`: This is a predicate related to the framework instance. Its frequency is 1, which means that the association of the framework instance with this predicate occurred 1 time.

`https://w3id.org/polifonia/resource/MusicBO\_105\_6991\_amr-ml/Person`: This is the object type associated with the predicate `https://w3id.org/framester/pb/localrole/assumer\_of\_attribute`. Its frequency is 1, indicating that the association of a framework instance with an object of this type occurred 1 time.

`None`: This is a special case, indicating that the object associated with the predicate `https://w3id.org/framester/pb/localrole/assumer\_of\_attribute` has no specific type. Its frequency is 0, indicating that there are no objects of unspecified type associated with the framework instance.

`subj\_pred\_total\_probability`: The total probability of frame instances and predicates, here is 0.0077.

`subj\_pred\_probability`: The probability of the frame instance and predicate, here is 0.5.

The same information applies to other predicates related to frame instances, such as `https://w3id.org/framester/pb/localrole/entity\_treated`.

We can use this information to understand the degree, frequency, and probability of associations between framework instances and different predicate and object types.

**5.4 Find the frame instance with the highest subject frequency**

The frame instance with the highest subject frequency generally represents the situation that occurs most frequently in the data. By identifying these framework instances, important topics, events, or entities in the data set can be found, which is very helpful for the overall understanding of the data and the determination of focus.

**6 Types imforemation**

**6.1 Frame Occurrence Type**

Different frame occurrence have different types of Uniform Resource Identifiers (URIs). Each frame occurrence can be identified by its URI, which usually contains information about the category or type to which the framework instance belongs.

Role: Analysis of different types of URIs can help identify and classify frame occurrenceto better understand different concepts, topics, or entities in RDF data. This is crucial for semantic understanding of data and knowledge graph construction.

**6.2 Number of occurrences of different types**

This refers to the number of occurrences of different types of resources (such as categories, attributes, instances, etc.) in RDF data. It reflects the distribution of various types in the data set.

What it does: Analyzing the number of occurrences of different types can help identify popular concepts or topics in a data set, and understand which concepts are more common and which may be rarer. This helps determine the importance and focus of the data.

**6.3 Type preference for framework instances**

This refers to the degree to which each framework instance prefers different types, usually expressed as probabilities or weights. It reflects the strength of the relationship between a framework instance and its associated type.

Purpose: Analyzing the type preference of framework instances helps to understand which types are more influential and which types are more relevant to specific framework instances. This can be used to infer semantic associations of framework instances and establish more accurate relationships in knowledge graphs.

**7 Similarity calculation**

**7.1 Simple similarity calculation method**



Parse the data from two different RDF data files and then use the NetworkX library to build the corresponding graph. In both graphs, nodes represent topics and objects in RDF triples, and edges represent relationships between topics and objects.

Calculate the similarity score between two knowledge graphs. The similarity score is based on a comparison of the number of nodes and the number of edges. The function calculates the number of nodes and edges for each graph and then calculates the similarity score between the number of nodes and the number of edges. Finally, the final similarity score is calculated by taking the average of the two scores.

**7.2 Improved similarity measure method**

Improved similarity measure method, taking into account the similarity of nodes, edges and predicates

Obtain the node, edge and predicate sets of two knowledge graphs.

Calculate node similarity: Determine the similarity of nodes by calculating the intersection and union of nodes.

Calculate edge similarity: Determine edge similarity by calculating the intersection and union of edges.

Calculate predicate similarity: Determine the similarity of predicates by calculating their intersection and union.

Comprehensive calculation of the overall similarity: the node, edge and predicate similarities are weighted and averaged to obtain the overall similarity score.

**7.3 Using Word2Vec model to obtain embedding representation of node labels**

This method takes into account the semantic information of node labels and can more accurately compare the similarities between knowledge graphs.

Node embedding representation: Using the Word2Vec model to obtain the embedding representation of node labels is a good method, which takes into account the semantic information of node labels.

Word segmentation processing: Use NLTK’s word segmenter to segment the text.

Word2Vec parameters: Parameters of Word2Vec model, such as vector\_size, window and min\_count

Exception handling

**8 natural language explanation**

**8.1 Generated natural language explanation**

The GPT-2 model in the Hugging Face Transformers library is used to generate natural language explanations based on the input knowledge graph difference description.

The differences in the knowledge graphs reflect: After comparative analysis of the two knowledge graphs, we found that their similarity score at the semantic level of node labels is 0.56. This score shows that, although there is a certain degree of similarity, the two graphs also exhibit clear differences in expressing certain concepts and entities in the music domain. In particular, this may reflect different authors' unique perspectives and interpretations of musical history, styles, or events. We wanted to dig deeper into the meaning behind this similarity score: how does it reflect the commonalities and divergences between the two knowledge graphs in describing musical elements? What cultural or historical contextual differences might these similarities and differences reveal? Please provide a natural language explanation based on this similarity score to help us better understand the connections and differences between the two knowledge graphs.. This implies that the similarity scores are not just a reflection of differences, but also reflect a commonality.

The similarity of knowledge graph

. The similarity graph is the most common and most important of all the information graphs. It is used to describe the relationship between a given concept and its associated entity. For example, a concept is related to a specific entity by its similarity to that concept. A concept can be related by a number of different factors, such as the number and type of a word, its meaning, and the type and content of its content. However, there are also many other factors that can affect the quality of an entity's knowledge. These factors include:

(1) the degree to which the concept or entity is associated with a particular concept, (2) its type, content, meaning and meaning of content (e.g., the word "tongue" is linked to the "Tongues" concept), (3) whether the entity has a "synthetic" meaning (i.e., a person who is not a human being), and (4) how the concepts are related. (The "Synthesis" and "Synchronicity" concepts have similar meanings, as do the words "the" or "s" in "The Song of Songs").

,, and. The concept/entity is also related in some way to its entity, e. g., by the fact that it is connected to an idea or concept by an attribute. Examples of this attribute are the name of one's own concept (the "Song of Song"), the title of another concept's concept ("The Artist"), and a...

**9 Conclusions**

The project involves obtaining, cleaning, and preprocessing data from the MusicBO Knowledge Graph's GitHub repository, then parsing the RDF graph data and analyzing framework instances to understand relationships and attribute values. The analysis content extends to frequency information, type information, similarity calculation, and natural language explanation generation. The goal is to highlight the differences in musical concept representation in different knowledge graphs and generate insights by using natural language explanations generated by pre-trained models.

The project report covers an in-depth analysis of the MusicBO knowledge graph, including extraction and comparison of framework instances, analysis of entity attributes and relationships in RDF graphs, frequency and similarity calculations, and generation of natural language explanations. The research focus includes the relationship between different frame instances, common frame parameters, construction of frequency dictionary, and analysis of type URI and number of occurrences. In addition, the report explores improvements in similarity measurement methods, including considering the similarity of nodes, edges, and predicates, and using the Word2Vec model to obtain embedding representations of node labels. Finally, the report transforms technical analysis into easy-to-understand explanations through natural language generation technology to improve the accessibility and understandability of knowledge graph data.