SDM Project 2

Master's Degree in Data Science (MDS)

Facultat d'Informàtica de Barcelona (FIB), UPC, 08034 Barcelona, Spain

Authors:
Daniel Weroński Falcó and Mateja Zatezalo

June 2025

Contents

1 Ontology Creation

The foundation of any knowledge graph is its ontology. An ontology serves as a formal specification of a domain, defining the concepts, their properties, and the relationships that exist between them. For this project, our goal was to model the domain of academic publications. The process of creating our ontology was divided into two fundamental stages: first, the design and implementation of the TBox, which acts as the schema or vocabulary; and second, the generation of the ABox, which contains the instance data that populates the knowledge graph according to the rules defined in the TBox.

1.1 TBOX Creation (B.1)

The TBox, or terminological box, provides the vocabulary for our knowledge graph. It is the schema that defines the classes of objects we are interested in and the properties that describe their attributes and interconnections. The design of our TBox was carefully planned to capture the key entities and the lifecycle of an academic paper, from authorship to publication.

Our model includes central classes such as `ex:Paper`, `ex:Person`, `ex:Journal`, and `ex:Event` (which further specializes into `ex:Conference` and `ex:Workshop`). The relationships between these classes are defined by properties. We distinguish between two types of properties:

- **Object Properties**, which link an individual of one class to an individual of another class. For example, the property `ex:isAuthor` links an individual of the class `ex:Person` to an individual of the class `ex:Paper`.
- **Datatype Properties**, which link an individual to a literal value, such as a string or a number. For example, `ex:hasName` links a `ex:Paper` individual to its title, which is a string.

Figure 1 presents a visual representation of our TBox schema. This diagram uses a clear visual language to distinguish between the different components of the ontology, which aids in its interpretation. Classes are shown as boxes with a black outline. Inside these boxes, datatype properties relevant to that class are listed in a light blue color. The arrows connecting the boxes represent object properties, and their labels are coloured green.

A key feature of RDFS, the language used to define our schema, is the ability to create class hierarchies. We represent this with a special property, `rdfs:subClassOf`, shown with a distinct blue arrow. This allows us to model specialization. For instance, both `ex:Conference` and `ex:Workshop` are defined as subclasses of `ex:Event`. This hierarchical structure is very powerful, as it enables queries at different levels of generality. One could query for all `ex:Event` publications and retrieve results from both conferences and workshops.

To maintain clarity and readability, the diagram does not represent properties as separate nodes, which is a common practice in formal ontology representations. Such an approach would significantly increase the visual complexity of the diagram. Instead, we have adopted a UML-style convention where properties are shown as labels on the connections, which makes the schema more intuitive and compact.

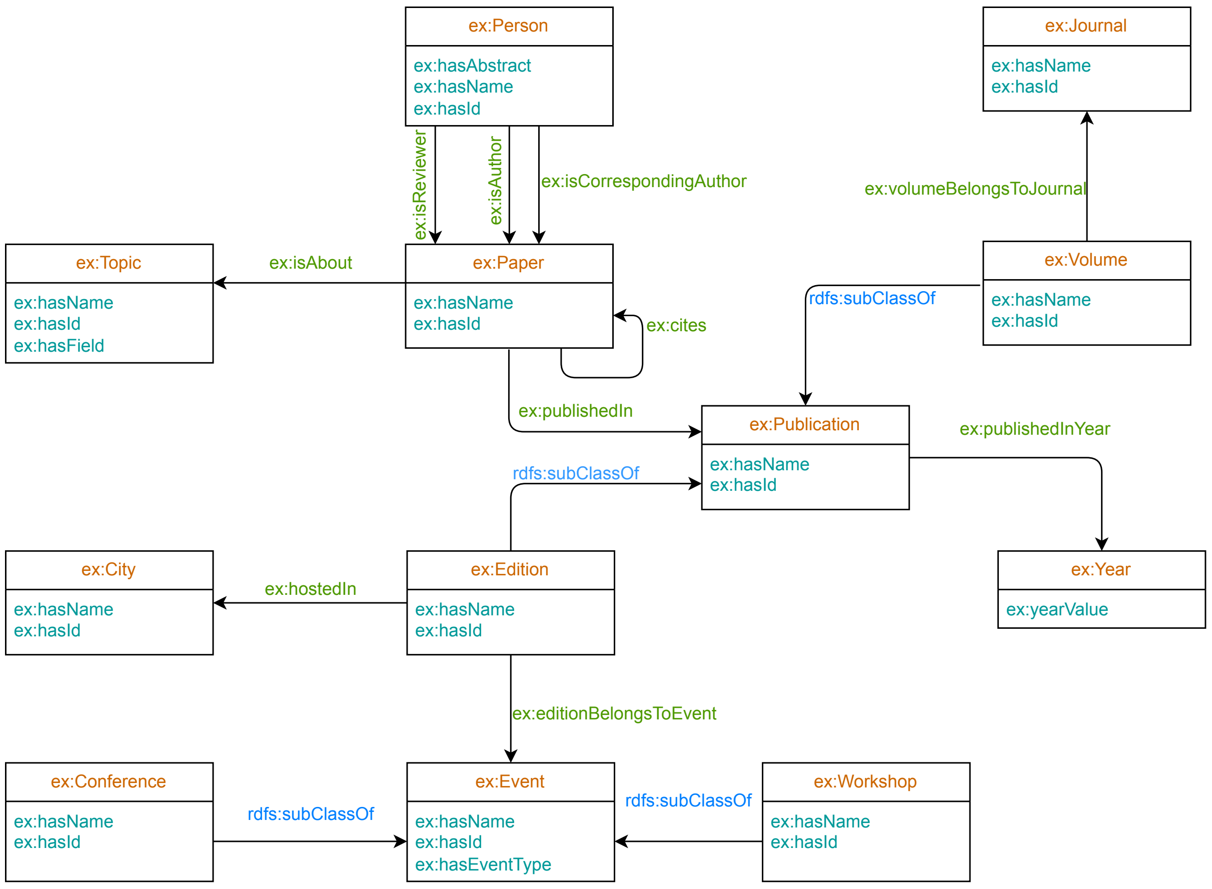


Figure 1: The TBox schema diagram for the publications knowledge graph. It shows classes (orange), datatype properties (blue), object properties (green), and subclass relationships (red).

1.2 ABOX Definition (B.2)

The ABox, or assertional box, contains the actual data instances that constitute the knowledge graph. While the TBox defines the rules, the ABox contains the individuals that conform to those rules. The source data for our ABox is the Semantic Scholar Academic Graph Dataset, a large-scale corpus of academic literature. This data is provided in a set of JSON files, which are well-structured for data exchange but lack the formal, explicit semantics necessary for a knowledge graph.

Our primary task was to “lift” this non-semantic data into RDF triples that adhere to our TBox schema. For this, we developed a Python script using the `rdflib` library. The methodology was systematic and designed to be robust and scalable.

1.2.1 Data Transformation Methodology

1. **URI Minting Strategy:** In RDF, every unique entity must be identified by a Unique Resource Identifier (URI). Establishing a consistent strategy for creating these URIs, known as “URI minting”, is critical to prevent data duplication and ensure data integrity. We created a rule-based approach to generate URIs from the unique identifiers present in the source JSON. For example:

- A paper with 'paperId: "e530..."' becomes the resource `ex:paper/e530...`.
- An author with 'authorId: "1694612"' becomes `ex:person/1694612`.
- A journal with 'id: "93be..."' becomes `ex:journal/93be...`.

This strategy guarantees that every time the same paper or author appears in the data, it is mapped to the exact same URI.

2. **Mapping and Triplification:** The core of the script iterates through each paper in the JSON files and maps its key-value pairs to RDF triples. For example, the 'title' key is mapped to the `ex:hasName` property, and the 'abstract' key to the `ex:hasAbstract` property. The logic also handles more complex structures. For instance, the 'publicationVenue.type' field is used as a switch to determine whether to create an instance of `ex:Journal` or `ex:Conference`. The script is also built to be resilient, gracefully handling cases where data is missing or 'null' by simply omitting the corresponding triple.

1.2.2 Inference and Entailment Considerations

Our system was designed with the RDFS Entailment regime in mind. This means that we expect an RDF-aware database, such as GraphDB, to use the rules in our TBox to logically infer new facts from the asserted data. This has important implications for which triples we need to explicitly create.

- **Inference with `rdfs:subClassOf`:** Our TBox states that `ex:Edition rdfs:subClassOf ex:Publication`. When our script processes a paper from a conference edition, it generates the triple `<myEdition> rdf:type ex:Edition`. We do not need to also add `<myEdition> rdf:type ex:Publication`. An RDFS reasoner will automatically deduce this second triple. This makes our data loading process more efficient and keeps our explicit data concise.

- **Inference with `rdfs:domain` and `rdfs:range`:** These constructs also enable inference. For example, the property `ex:isAuthor` has a domain of `ex:Person` and a range of `ex:Paper`. When we assert the triple `<janeDoe> ex:isAuthor <paper1>`, a reasoner can infer that `<janeDoe>` is an instance of `ex:Person` and `<paper1>` is an instance of `ex:Paper`.

Despite the power of inference, we adopted a best-practice approach of being explicit where possible. Our script explicitly asserts the `rdf:type` for all major entities (papers, persons, journals, etc.) when this information is clearly available in the source JSON. This makes our ABox data more self-contained and portable, ensuring it can be correctly interpreted even by systems that do not perform RDFS reasoning. This programmatic approach allows us to transform a large, non-semantic dataset into a rich, interconnected, and logically consistent knowledge graph.

1.3 Knowledge Graph Statistics and Discussion

After populating our triplestore with the data generated from the Semantic Scholar JSON files, we performed a statistical analysis to understand the size and structure of the resulting knowledge graph. We executed a series of SPARQL queries directly within the GraphDB Workbench to compute these metrics. The results provide valuable insights into both our schema (TBox) and the instance data (ABox).

1.3.1 Summary of Statistics

Table 1 presents a summary of the key statistics for the knowledge graph. The table is divided into three parts: schema-level statistics, counts of instances for our main classes, and counts of triple patterns for our main properties.

Table 1: Summary Statistics of the Populated Knowledge Graph (see Appendix)

Category	Item	Count
TBox (Schema) Statistics		
Schema	Number of Classes	12
Schema	Number of Properties	17
ABox (Instance) Statistics		
Class Instances	Papers (`ex:Paper`)	80541
Class Instances	Persons (`ex:Person`)	4238
Class Instances	Journals (`ex:Journal`)	84
Class Instances	Volumes (`ex:Volume`)	185
Class Instances	Topics (`ex:Topic`)	15
ABox (Property Usage) Statistics		
Property Triples	isAuthor (`ex:isAuthor`)	3897
Property Triples	cites (`ex:cites`)	96372
Property Triples	isAbout (`ex:isAbout`)	1042
Property Triples	publishedIn (`ex:publishedIn`)	587

1.3.2 Analysis of Results

The statistics in Table 1 confirm that our data transformation was successful, and they reveal important characteristics of our dataset.

Schema Validation: The counts for classes (12) and properties (17) perfectly match the number of entities defined in our TBox schema. This serves as a basic validation that our ontology was loaded correctly into the triplestore.

Core vs. Network Papers: A key observation is the large number of `ex:Paper` instances (80541), even though our source data consisted of exactly 1000 fully described papers. This is not an error, but rather a feature of our graph-based approach. Our data lifting script correctly creates an `ex:Paper` resource for every unique paper ID it encounters. This includes the 1000 "core" papers from the source files, as well as the thousands of other papers that were mentioned in the 'citations' list of these core papers. This result shows that our knowledge graph represents not just a collection of documents, but a densely interconnected citation network. The high count for the `ex:cites` property (96372) further supports this conclusion.

Author and Publication Data: The number of `ex:Person` instances (4238) represents the authors and reviewers associated with the 1000 core papers. The `ex:isAuthor` count (3897) is slightly lower, which is expected as some of the people entries in the source JSON only be present as reviewers. Similarly, the count for `ex:publishedIn` triples (587) indicates that just over half of the core papers contained complete publication venue information, which is a reflection of the completeness of the source dataset itself. The numbers for Journals (84) and Volumes (185) show a healthy ratio, indicating that our data contains multiple volumes from the same journals, as expected.

Topic Distribution: The data shows a small number of unique topics (15) but a high number of links to them (1042). This suggests that the fields of study in the source data are drawn from a limited set of general categories, and these categories are frequently reused across many papers.

2 Knowledge Graph Querying and Analysis (B.3)

The primary motivation for constructing a knowledge graph is to enable complex queries and analyses that are difficult to perform with traditional database systems like SQL. Using the SPARQL query language, we can leverage the rich structure and formal semantics of our ontology to uncover insightful relationships within the data. This section presents two such queries that demonstrate the unique capabilities of our graph-based approach, including the use of RDFS reasoning and the calculation of graph-based metrics.

2.1 Querying with RDFS Reasoning: Finding Papers in Events

One of the key benefits of defining a TBox is the ability to perform reasoning over the class hierarchy. The following query was designed to exploit this capability.

2.1.1 The Question and Results

The query answers the following question: "Show me the titles of all papers published in any kind of academic 'Event' during the year 2020."

This query is interesting because our TBox defines `ex:Event` as a superclass of both `ex:Conference` and `ex:Workshop`. Therefore, the query should return papers from both types of venues without needing to specify them individually. The results from running this query against our dataset are shown in Table 2.

Table 2: Papers Published in Events During 2020 (extract)

Paper Title
ChainLink: Indexing Big Time Series Data For Long Subsequence Matching
FALKON: Large-Scale Content-Based Video Retrieval Utilizing Deep-Features ...
JUST: JD Urban Spatio-Temporal Data Engine
Efficient spatial data partitioning for distributed Managing Heterogeneous Data ...

2.1.2 Analysis and Query

The power of this query lies in the line `?event a ex:Event`. When the GraphDB reasoner encounters this pattern, it uses the `rdfs:subClassOf` relationships defined in our TBox.

It understands that any individual that is an instance of `ex:Conference` or `ex:Workshop` is also, by inference, an instance of `ex:Event`.

This provides a significant advantage over a relational SQL database. In SQL, conference and workshop data would likely be in separate tables. Answering the same question would require querying both tables and combining the results with a `UNION` operation. Our SPARQL query (Listing 1) is not only more concise but also more maintainable. If we were to add a new subclass to `ex:Event`, such as `ex:Symposium`, the query would work immediately without any modification.

```sparql
PREFIX ex: <http://example.org/ontology/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT DISTINCT ?paperTitle
WHERE {
  # 1. Find a publication edition that belongs to any "Event".
  #    Reasoning will include conferences and workshops here.
  ?edition ex:editionBelongsToEvent ?event .
  ?event a ex:Event .

  # 2. Filter these events to only those from the year 2020.
  ?edition ex:publishedInYear ?year_uri .
  ?year_uri ex:yearValue "2020"^^<http://www.w3.org/2001/XMLSchema#
   gYear> .

  # 3. Find the papers published in these editions.
  ?paper ex:publishedIn ?edition .
  ?paper ex:hasName ?paperTitle .
}
ORDER BY ?paperTitle
```

Listing 1: Query to find papers published in any event in 2020.

2.2 Graph Analytics: Identifying Collaborative Impact

Our second query moves beyond simple data retrieval to perform graph analytics. It calculates a new metric for each author to identify those with the highest collaborative impact.

2.2.1 The Question and Results

The query was designed to answer: “Who are the top 10 most collaborative authors, ranked by their average number of unique co-authors per paper?”

This metric provides a more nuanced measure of collaboration than simply counting papers or total co-authors. It highlights researchers who consistently work with larger and more diverse teams. Table 3 shows the results from our dataset. While the scores are uniform due to the limited size of the test data, the methodology demonstrates a powerful analytical capability.

2.2.2 Analysis and Query

This query (Listing 2) demonstrates the ability to perform complex aggregations based on graph patterns. The inner subquery first identifies the pattern of co-authorship (`author`

Table 3: Top Authors by Collaborative Impact Score (extract)

Author Name	Paper Count	Total Co-authors	Impact Score
D. Misev	3	40	13.33
Vlad Merticariu	3	38	12.67
Hai Jin	4	32	8
V. Borkar	3	24	8

`-> paper <- coauthor`). It then uses `COUNT(DISTINCT)` to calculate the total number of papers and unique co-authors for each author.

The main query then uses these aggregated values to calculate a new, derived metric, the `collaborativeImpactScore`, using the `BIND` clause. This ability to define and calculate network-based metrics on the fly is a core strength of the graph model. The query essentially performs a small-scale data science task directly within the database, providing insights into the structure and dynamics of the academic social network. This kind of structural analysis is significantly more complex and less intuitive to implement in SQL.

```
PREFIX ex: <http://example.org/ontology/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?authorName ?paperCount ?totalUniqueCoauthors ?
   collaborativeImpactScore
WHERE {
  # Subquery to calculate core statistics for each author
  {
    SELECT ?author
           (COUNT(DISTINCT ?paper) AS ?paperCount)
           (COUNT(DISTINCT ?coauthor) AS ?totalUniqueCoauthors)
    WHERE {
      ?author a ex:Person .
      ?author ex:isAuthor ?paper .
      ?coauthor ex:isAuthor ?paper .
      FILTER(?author != ?coauthor)
    }
    GROUP BY ?author
  }
  # Filter for authors with more than 2 papers to avoid skewed averages
  FILTER(?paperCount > 2)

  ?author ex:hasName ?authorName .

  # Calculate the collaborative impact score
  BIND((xsd:float(?totalUniqueCoauthors) / xsd:float(?paperCount)) AS ?
   collaborativeImpactScore)
}
ORDER BY DESC(?collaborativeImpactScore)
LIMIT 10
```

Listing 2: Query to find the top 10 authors by collaborative impact.

3 Knowledge Graph Embeddings

3.1 Importing the data (C.1)

Regarding the data selection for KGE, from the full *GraphDB* export, we retained only those relations that carry semantic weight for link prediction in our domain:

- **isAuthor**: authorship edges
- **cites**: citation links
- **publishedIn**: publication-paper links

Other edges (e.g. isAbout, publishedInYear, etc.) were omitted to reduce noise and focus the embedding model on strictly inter-node connectivity. Triples are exported and saved into a `.tsv` file.

3.2 Getting familiar with KGEs (C.2)

3.2.1 Most basic model (C.2_1)

In this section, the first task was to find the most likely embedding vector, according to the TransE model, that would correspond to a paper cited by the chosen paper.

In the TransE model, the score of a triple (h, r, t) is

$$f(h, r, t) = -\|\mathbf{e}_h + \mathbf{e}_r - \mathbf{e}_t\| \ .$$

(inverse of the loss). To predict the tail entity t for a fixed $(h = P, r = \text{cites}, ?)$, we looked for the embedding vector

$$\mathbf{v}_{\text{pred}} = \mathbf{e}_P + \mathbf{e}_{\text{cites}}$$

and then chose the existing paper whose embedding $\mathbf{e}_t$ is closest to $\mathbf{v}_{\text{pred}}$ in Euclidean distance.

Regarding the task of finding the most likely embedding vector for its author, we used the same procedure, but using the relation "isAuthor" instead of "cites." and computed

$$\mathbf{v}_{\text{auth\_pred}} = \mathbf{e}_P + \mathbf{e}_{\text{isAuthor}}$$

With this, we found, among all author-nodes, the one whose embedding is closest to $\mathbf{v}_{\text{auth\_pred}}$.

For the chosen paper with URI:
`<http://example.org/ontology/edition/1124d164-490d-4da9-b0c2-c8ca0dc98202-2018>`,
paper included in citation with minimum Euclidean distance is:
`<http://example.org/ontology/edition/f6b09e7e-ad3d-46cb-857d-23e49394343c-1986>`
with the score of -6.14.

3.2.2 Improving TransE

1. **Given a KG with the triples (Author1, writes, Paper1), (Author2, writes, Paper1) and (Author1, writes, Paper2), describe the optimal solutions that TransE would give to the embeddings of the entities.**

For those three triples TransE can drive the loss to zero by collapsing everything onto a single point. Concretely:

1. $\mathbf{e}_{\text{Author1}} + \mathbf{r}_{\text{writes}} \approx \mathbf{e}_{\text{Paper1}}$
2. $\mathbf{e}_{\text{Author2}} + \mathbf{r}_{\text{writes}} \approx \mathbf{e}_{\text{Paper1}}$
3. $\mathbf{e}_{\text{Author1}} + \mathbf{r}_{\text{writes}} \approx \mathbf{e}_{\text{Paper2}}$

From (1) and (2) we get

$$\mathbf{e}_{\text{Author1}} + \mathbf{r} = \mathbf{e}_{\text{Paper1}} \quad \text{and} \quad \mathbf{e}_{\text{Author2}} + \mathbf{r} = \mathbf{e}_{\text{Paper1}} \implies \mathbf{e}_{\text{Author1}} = \mathbf{e}_{\text{Author2}}.$$

and plugged into (3):

$$\mathbf{e}_{\text{Author1}} + \mathbf{r} = \mathbf{e}_{\text{Paper2}} \quad \implies \quad \mathbf{e}_{\text{Paper2}} = \mathbf{e}_{\text{Paper1}}.$$

This means all four entity embeddings coincide, and you can choose

$$\mathbf{r}_{\text{writes}} = \mathbf{0}, \quad \mathbf{e}_{\text{Author1}} = \mathbf{e}_{\text{Author2}} = \mathbf{e}_{\text{Paper1}} = \mathbf{e}_{\text{Paper2}} = \mathbf{0}$$

(or any common vector, with $\mathbf{r}$ set to the zero vector). That assignment makes each $\|\mathbf{e}_h + \mathbf{r} - \mathbf{e}_t\|$ exactly zero, which TransE regards as the global optimum.

TransE cannot distinguish a one-to-many or many-to-one relationships – all head nodes collapse and all tail nodes collapse.

2. **From the models considered in this section, find one that has been created to (at least partially) solve the previous issue and briefly explain how it achieves so.**

Next step beyond TransE is **TransH**. Instead of forcing every triple

$$\|\mathbf{e}_h + \mathbf{r} - \mathbf{e}_t\| \approx 0$$

in the same vector space, TransH gives each relation r its *own* hyperplane (defined by a normal vector $\mathbf{w}_r$). We then project entities onto that plane. For a given triple (h, r, t), we first project $\mathbf{e}_h$ and $\mathbf{e}_t$ onto the hyperplane of r:

$$\tilde{\mathbf{e}}_h = \mathbf{e}_h - \mathbf{w}_r^\top \mathbf{e}_h \mathbf{w}_r, \quad \tilde{\mathbf{e}}_t = \mathbf{e}_t - \mathbf{w}_r^\top \mathbf{e}_t \mathbf{w}_r.$$

Then we enforce

$$\|\tilde{\mathbf{e}}_h + \mathbf{r} - \tilde{\mathbf{e}}_t\| \approx 0.$$

Because each relation has its *own* projection, TransH can (for example) map “isAuthor” into a subspace where Author1 and Author2 can remain distinct while still reaching the same Paper1 embedding—and likewise Paper2 can be embedded differently. TransH reduces the “collapse” of TransE on one-to-many or many-to-one patterns.

3. **A relation stating that two authors collaborate with each other (e.g., collaboratesWith) is symmetric. With TransE, will the model give the same score to the triples (Author1, collaboratesWith, Author2) and (Author2, collaboratesWith, Author1)? Should they be the same?**

TransE's score for a triple (h, r, t) is

$$\text{score}(h, r, t) = -\|\mathbf{e}_h + \mathbf{r} - \mathbf{e}_t\|,$$

so if we swapped head and tail we get

$$\text{score}(t, r, h) = -\|\mathbf{e}_t + \mathbf{r} - \mathbf{e}_h\| = -\| - (\mathbf{e}_h + \mathbf{r} - \mathbf{e}_t) + 2\mathbf{r}\|.$$

Unless $\mathbf{r} = \mathbf{0}$ (or the embeddings happen to satisfy a very special symmetry) $\mathbf{e}_h + \mathbf{r} - \mathbf{e}_t \neq \mathbf{e}_t + \mathbf{r} - \mathbf{e}_h$, so TransE will generally assign different scores to (Author$_1$, collaboratesWith, Author$_2$) and (Author$_2$, collaboratesWith, Author$_1$).

Intuitively, because TransE treats each relation as a directional "translation" vector, it has no built-in way to force $\mathbf{e}_h + \mathbf{r} \approx \mathbf{e}_t$ and $\mathbf{e}_t + \mathbf{r} \approx \mathbf{e}_h$ simultaneously (unless $\mathbf{r} = \mathbf{0}$). But for a symmetric relation like "collaboratesWith", the model should score those two triples identically.

4. **Describe and sketch (in a simple 2D drawing), why with TransE symmetry is not generally achieved (i.e., unless a trivial and/or degenerate solution is obtained).**

$$\begin{tikzcd} \underset{\text{"head"}}{\mathbf{e}_h} \arrow[r, "\mathbf{r}"] & \mathbf{e}_t \end{tikzcd}$$

- TransE enforces

$$\mathbf{e}_h + \mathbf{r} \approx \mathbf{e}_t,$$

so that adding the "translation" vector $\mathbf{r}$ to the head embedding lands on the tail.

If we now swap head and tail we also need

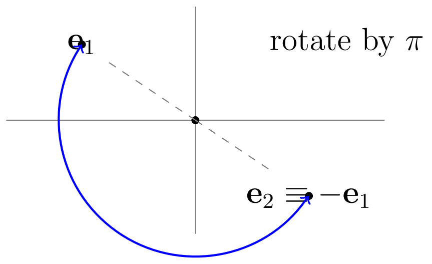
$$\mathbf{e}_t + \mathbf{r} \approx \mathbf{e}_h,$$

i.e.

$$\mathbf{e}_h \approx \mathbf{e}_t + \mathbf{r} \implies \mathbf{e}_h + \mathbf{r} \approx \mathbf{e}_t + 2\mathbf{r}.$$

Graphically:

$$\begin{tikzcd} \underset{\text{"head"}}{\mathbf{e}_h} \arrow[r, "\mathbf{r}"] \arrow[d, dash, dashed] & \underset{\text{"tail"}}{\mathbf{e}_t} \arrow[d, dash, dashed] & \\ \mathbf{e}_h \arrow[r, phantom, "\neq"] & \mathbf{e}_t \arrow[r, dashed, "\mathbf{r}"] & \mathbf{e}_t + \mathbf{r} \end{tikzcd}$$



5. **RotatE is a model that has been created to deal, among other things, with symmetry. Again, in a 2D figure, sketch which are the circumstances under which the symmetry (Author1, relation, Author2) and (Author2, relation, Author1) is achieved.**

- Points $\mathbf{e}$ and $-\mathbf{e}$ are diametrically opposite on the circle.
- The arrow "rotate by π" carries $\mathbf{e} \to -\mathbf{e}$ and also $-\mathbf{e} \to \mathbf{e}$.
- Thus RotatE achieves symmetry exactly by learning a 180° rotation for that relation.

6. **Name (and explain) one of the considered models that allows for symmetry without any consideration.**

KGE model that allows for symmetry without any consideration is **DistMult**. Its scoring function for a triple (h, r, t) is

$$f(h, r, t) = \langle \mathbf{e}_h, \mathbf{w}_r, \mathbf{e}_t \rangle = \sum_{i=1}^{d} (\mathbf{e}_h)_i (\mathbf{w}_r)_i (\mathbf{e}_t)_i$$

where $\mathbf{e}_h, \mathbf{e}_t \in \mathbf{R}^d$ are the head/tail embeddings and $\mathbf{w}_r \in \mathbf{R}^d$ is a diagonal relation embedding.

Because the score is a dot-product of three vectors (head, relation, tail) with the relation vector in the middle, no matter how it is trained, $f(h, r, t) = f(t, r, h)$; every relation is modeled as symmetric.

3.3 Training KGEs (C.3)

For training KGEs and experimenting with evaluation results of various models, we chose *TransE*, *DistMult* and *ComplEx*. Regarding the hyperparameters, we experimented with 64 and 128 embedding dimension, 1 and 5 number of negative samples per positive samples. Number of epochs was set to 20, and the learning rate to 0.05.

In order to obtain insightful comparison of KGE training runs, we focused on the following evaluation metrics:

- **Mean Reciprocal Rank (MRR)**: gives a single real-valued summary of how high up, on average, the "correct" tail (or head) appears in the model's full ranking
- **Hits at 1 and 10 (H@1 & H@10)**: tells the fraction of times the true entity is the #1 guess (very stringent), while Hits@10 tells the fraction it's anywhere in the top ten (more forgiving)

- **Median Rank (MDR)**: gives a robust sense of the "typical" position of the correct answer

From Table 4, we can see that DistMult emerges clearly on top for this domain of "authorship" and "citation" relations. Its best setting (128-d embeddings, 5 negatives per positive) achieves an MRR of 0.967 and Hits@1 of 0.956, versus TransE's best MRR of only ∼0.343 (Hits@1 ≈ 0.012) and ComplEx's peak MRR ≈ 0.105.

DistMult's ability to model one-to-many and many-to-one patterns gives it a huge edge over TransE's simple translation mechanism, which struggles with our "cites" relation. ComplEx, while being able to handle asymmetry, underperforms here—likely because our KG is dominated by one-to-many patterns that DistMult already captures well.

Table 4: Comparison of KGE models on evaluation metrics

Model	**Dim**	**Neg/Pos**	**MRR**	**MDR**	**H@1**	**H@10**
TransE	64	1	0.2181	7	0.0030	0.5511
TransE	64	5	0.3428	2	0.0118	0.7819
TransE	128	1	0.1525	30	0.0008	0.3896
TransE	128	5	0.2171	8	0.0015	0.5277
DistMult	64	1	0.8939	1	0.8638	0.9472
DistMult	64	5	0.9659	1	0.9552	0.9841
DistMult	128	1	0.8856	1	0.8555	0.9365
DistMult	128	5	0.9673	1	0.9559	0.9855
ComplEx	64	1	0.0789	125	0.0396	0.1485
ComplEx	64	5	0.0800	56	0.0285	0.1719
ComplEx	128	1	0.1051	109	0.0622	0.1842
ComplEx	128	5	0.0982	44	0.0391	0.2065

Based on these results, we will use **DistMult** with embedding dimension 128 and 5 negative samples per positive. It delivers both the highest overall ranking quality and best precision, making it the best foundation for further ML tasks.

3.4 Exploiting KGEs (C.4)

For the task of Exploiting KGEs, we grouped authors into clusters based on the similarity of their knowledge-graph embeddings, to see whether authors who play similar roles or work in similar venues naturally "sit together" in the vector space. We chose this task since it is an unsupervised downstream application, using only the learned embeddings.

The triples were formed the same as before (3.1), and they are split 80/20 to train and test data. Parameters used for training include the model *DistMult*, embedding dimension of 128, and learning rate of 0.05, while negative samples per positives was set to 5.

Then, the author embeddings were extracted. By using *KMeans* algorithm, 3 clusters were created, with sizes of `C0:2044, C1:1140, C2:63`. Cluster `C2` is quite small and well isolated—those authors form a tightly-connected subcommunity. The other clusters blend more, reflecting overlapping co-publication patterns in broader venues.

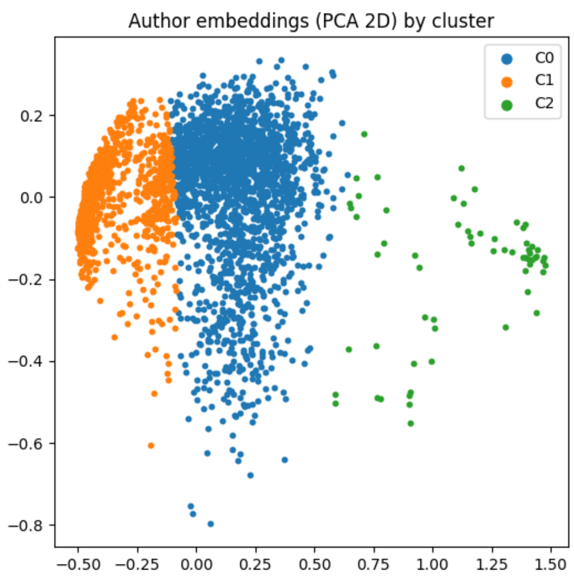


Figure 2: Author embeddings clustered with k-means (projected to 2D via PCA).

Silhouette score we received is 0.093, which is considered to be low. This means most author embeddings sit almost on the boundary between clusters. In this case, DistMult embeddings may not be strongly grouping authors, or authors simply do not form different "groups" in the embedding space.

As shown in Figure 2, clusters 0 and 1 show natural grouping, while cluster 2 captures the small set of "outlier" authors whose embedding coordinates are furthest from the main mass. These may be highly specialized authors who publish in different venues or on very niche topics.

4 Appendix: SPARQL Queries for Statistical Analysis

This section contains the complete set of SPARQL queries used to generate the statistics presented in Table 1.

TBox Statistics

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT (COUNT(DISTINCT ?class) AS ?numberOfClasses)
WHERE {
  ?class a rdfs:Class .
  # We filter to count only the classes from our specific namespace.
  FILTER(STRSTARTS(STR(?class), "http://example.org/ontology/"))
}
```

Listing 3: Query to count the total number of classes in the ontology.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT (COUNT(DISTINCT ?property) AS ?numberOfProperties)
WHERE {
  ?property a rdf:Property .
  # We filter to count only the properties from our specific namespace.
  FILTER(STRSTARTS(STR(?property), "http://example.org/ontology/"))
}
```

Listing 4: Query to count the total number of properties in the ontology.

ABox Instance Statistics

```
PREFIX ex: <http://example.org/ontology/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT (COUNT(DISTINCT ?s) AS ?count)
WHERE {
  ?s a ex:Paper .
}
```

Listing 5: Query to count unique ex:Paper instances.

```
SELECT (COUNT(DISTINCT ?s) AS ?count)
WHERE {
  ?s a ex:Person .
}
```

Listing 6: Query to count unique ex:Person instances.

```
SELECT (COUNT(DISTINCT ?s) AS ?count)
WHERE {
  ?s a ex:Journal .
}
```

Listing 7: Query to count unique ex:Journal instances.

```
SELECT (COUNT(DISTINCT ?s) AS ?count)
WHERE {
  ?s a ex:Volume .
}
```

Listing 8: Query to count unique ex:Volume instances.

```
SELECT (COUNT(DISTINCT ?s) AS ?count)
WHERE {
  ?s a ex:Topic .
}
```

Listing 9: Query to count unique ex:Topic instances.

ABox Property Usage Statistics

```
PREFIX ex: <http://example.org/ontology/>

SELECT (COUNT(*) AS ?count)
WHERE {
  ?subject ex:isAuthor ?object .
}
```

Listing 10: Query to count triples using the ex:isAuthor property.

```
SELECT (COUNT(*) AS ?count)
WHERE {
  ?subject ex:cites ?object .
}
```

Listing 11: Query to count triples using the ex:cites property.

```
SELECT (COUNT(*) AS ?count)
WHERE {
  ?subject ex:isAbout ?object .
}
```

Listing 12: Query to count triples using the ex:isAbout property.

```
SELECT (COUNT(*) AS ?count)
WHERE {
  ?subject ex:publishedIn ?object .
}
```

Listing 13: Query to count triples using the ex:publishedIn property.