PROJECT REPORT :

Web scraping, knowledge base construction

Course : *Web Data mining & semantics*
Group members : *Ilan ZINI, Wendy DUONG*

Table of contents :

I. Introduction

II. Web scraping and knowledge base construction (Part 1)

A. Task 1 : Model for NER
1. Text Cleaning & Preprocessing
2. Named Entity Recognition (NER) with CRF model
3. Relation Extraction (RE)
4. Knowledge Graph Building
5. Export

B. Task 2 : Pipeline for Knowledge Graph Construction
1. Fetch News Articles
2. Use Methods from Task 1
3. Pipeline creation

III. Knowledge Graph Embedding (Part 2)

A. PART 1 – WITHOUT Data Augmentation
1. Data Preparation
2. Model: TransE
3. Model: DistMult
4. Performance Comparison

B. PART 2 – WITH Data Augmentation
1. DBpedia Enrichment
2. Retraining Pipeline

C. PART 3 – Comparison WITH vs WITHOUT DA

D. PART 4 – Conclusion

IV. Conclusion

A. Appendices

I. Introduction

This project focuses on building a knowledge graph (KG) from web articles and applying graph embeddings for link prediction. By automating web scraping, named entity recognition (NER), and relation extraction, we aim to create a structured graph that represents entity relationships.
We will construct the KG in RDF format, test two different embedding models, and evaluate their effectiveness in predicting missing links. Additionally, we will analyze how data augmentation impacts model performance using standard metrics like MRR and Hits@k. This will help optimize KG-based tasks such as recommendation systems and data analysis.

II. Web scraping and knowledge base construction (Part 1)

A. Task 1 : Model for NER

1. Text Cleaning & Preprocessing

Firstly, we install the necessary libraries for web scraping, natural language processing, and machine learning tasks. Then, we load the CoNLL-2003 dataset on which web scraping and data preprocessing will be applied.

We load the SpaCy language model en_core_web_sm and define a function to clean text : by removing stop words, punctuation (except for '-'), and applying lemmatization to convert words to their base form in lowercase.

The function is then applied to the first sentence of the dataset to demonstrate the cleaning process :

```
Original tokens: ['EU', 'rejects', 'German', 'call', 'to', 'boycott', 'British', 'lamb', '.']
Cleaned tokens: ['eu', 'reject', 'german', 'boycott', 'british', 'lamb']
```

→ The cleaning process removed stop words ("to") and punctuation (".") and applied lemmatization, converting "rejects" to "reject" and "EU" to "eu". The remaining tokens are key terms for further analysis.

2. Named Entity Recognition (NER) with CRF model

2.1. Feature extraction for CRF

Firstly, we create mappings for the Part-of-Speech (POS) tags and Named Entity Recognition (NER) tags. Then, we define a function to convert each example from the dataset into a CRF-compatible format. Next, we define a function to extract features for each word in a sentence, including word case, word length, POS tag, and neighboring words' features. We then convert the entire sentence into a list of features for each word using the *sent2features(sent)* function and extract the NER tags with *sent2labels(sent)*. Finally, we test the entire process on the first sentence of the dataset and display the extracted features and NER tags:

```
Features for word 0 (EU):
{'bias': 1.0, 'word.lower()': 'eu', 'word[-3:]': 'EU', 'word[-2:]': 'EU', 'word.isupper()': True, 'word.istitle()': False, 'word.isdigit()': False, 'postag': 'NNP', 'postag[:2]': 'NN', 'BO
S': True, '+1:word.lower()': 'rejects', '+1:postag': 'VBZ'}

NER Tags:
['B-ORG', 'O', 'B-MISC', 'O', 'O', 'O', 'B-MISC', 'O', 'O']
```

→ The feature extraction process generates detailed features for each word, considering its context (previous and next words). NER tags are assigned, such as labeling "EU" as an organization (B-ORG), preparing the data for tasks like CRF-based training. These features and tags are used to train a model that can recognize and classify entities in future text.

2.2. CRF model training for NER

We prepare the data by extracting features and labels from the training (2000 sentences) and test (500 sentences) datasets.Then, we train a CRF model using the *sklearn_crfsuite* library with specific hyperparameters (such as c1, c2, and max_iterations). After training, the model is evaluated on the test set, and the classification report is generated to assess the performance of the model in recognizing named entities.

	precision	recall	f1-score	support
B-LOC	0.845	0.816	0.830	288
B-MISC	0.651	0.831	0.730	65
B-ORG	0.601	0.441	0.509	188
B-PER	0.850	0.821	0.835	442
I-LOC	0.750	0.396	0.519	53
I-MISC	0.722	0.830	0.772	47
I-ORG	0.500	0.288	0.365	66
I-PER	0.861	0.956	0.906	343
O	0.972	0.986	0.979	4453
accuracy			0.931	5945
macro avg	0.750	0.707	0.716	5945
weighted avg	0.926	0.931	0.927	5945

→ The CRF model achieved an F1 score of 0.93, with strong performance across entity types like I-PER, B-LOC, and B-PER. While detection of multi-word entities like I-ORG and I-LOC was more challenging, the overall accuracy was 93.1%, and the weighted F1 score was 0.927, indicating solid classification performance.

2.3. Comparison with the spaCy model en_ner_conll03

We load the pre-trained spaCy model **en_ner_conll03** from the specified path. Then, we test the model by applying it to a sample `text = "Apple was founded by Steve Jobs. Elon Musk founded SpaceX."` and

extract the named entities detected by the model. The detected entities and their labels (e.g., organization or person) are then printed.
It then returns :

```
Entities detected by spaCy: [('Apple', 'ORG'), ('Steve Jobs', 'PER'), ('SpaceX.', 'MISC')]
```

→ spaCy correctly identified **Apple** as an organization (ORG) and **Steve Jobs** as a person (PER). However, it incorrectly classified **SpaceX** as a miscellaneous entity (MISC) instead of an organization, which could be due to limitations or errors in the model's handling of certain entities.

3. Relation Extraction

In this code, we load the spaCy model en_core_web_sm and define a function to extract relationships from a given text using three simple rules. The function analyzes the syntactic dependencies in the text to identify relationships between entities. **Rule 1**: Identifies **subject-verb-object** relationships, where the verb is the root of the sentence, **Rule 2**: Detects **subject-verb-preposition-object** relationships, where a preposition is linked to the verb, **Rule 3**: Extracts relationships like "**X is the CEO of Y**", identifying the CEO and their company.

The function then returns these relationships as triples (subject, relation, object).

Here are some examples :

- `text = """Steve Jobs founded Apple. Elon Musk works at SpaceX.
Sundar Pichai is the CEO of Google. Jeff Bezos owns Blue Origin."""` returns

```
Extracted relations:
('Jobs', 'found', 'Apple')
('Musk', 'work_at', 'SpaceX.')
('Bezos', 'own', 'Origin')
```

→ The model correctly extracts key subject-verb-object triples but misclassifies "work at" as "work_at" and splits "Blue Origin," capturing only "Origin.", indicating a limitation in handling multi-word entities.

- `text = "Sundar Pichai is the CEO of Google."` returns `[('Pichai', 'ceo_of', 'Google')]`

→ The model correctly identifies the "CEO of" relationship, linking "Pichai" to "Google."

4. Knowledge Graph Building

We install the rdflib library for working with RDF data and knowledge graphs. Then, we create an empty RDF graph to store extracted relations as triples. Entities and relations are converted into RDF format using a custom namespace `EX = Namespace("http://example.org/")` , and the graph is serialized in Turtle format for visualization :

Example :
```
relations = [
    ("Jobs", "found", "Apple"),
    ("Musk", "work_at", "SpaceX"),
    ("Pichai", "ceo_of", "Google")
]
```
returns
```
@prefix ex: <http://example.org/> .

ex:Jobs ex:found ex:Apple .

ex:Musk ex:work_at ex:SpaceX .

ex:Pichai ex:ceo_of ex:Google .
```
.

→ The extracted relations are converted into RDF triples using the predefined namespace. Each subject (ex: "Jobs") is linked to an object (ex: "Apple") through a predicate (ex: "found"), forming structured knowledge in Turtle format.

5. Export

We export the RDF graph into two different formats:

1. RDF/XML format:
   `g.serialize(destination="output/part1_task1_knowledge_graph.rdf", format="xml")`, which is a widely used XML-based format for representing RDF data.
2. Turtle format :
   `g.serialize(destination="output/part1_task1_knowledge_graph.ttl", format="turtle")`, a more readable format for RDF data representation.

This allows for easier sharing, storage, and further processing of the structured knowledge.

B. *Task 2* : Pipeline for Knowledge Graph Construction

1. Fetch News Articles

In this part of the project, we first attempted to scrape articles from Reuters using **selenium** and **BeautifulSoup** to retrieve the HTML content of the page. However, Reuters blocked the scraping by showing a CAPTCHA. As a result, we decided to switch our data source to **BBC News**, specifically targeting the **business** section.

To achieve this, we used **requests** to retrieve the HTML content from the BBC Business page, and then we used **BeautifulSoup** to parse and extract 10 articles, including their titles, summaries, publication dates, authors, and full text when available. The full articles were scraped by following the links to each article.

Here are some articles that have been retrieved by our function :

```
Article 1:
Title: Investors facing tariff turmoil: 'It's fastest finger first'
URL: https://www.bbc.com/news/articles/c9djj7pgz57o
Summary: Traders are trying to reckon with impact of Trump's shifting tariff policy.
Publication Date: 5 hrs ago
Author: Business
Full Text: As a former champion runner, Richard McDonald can move quickly. But the speed of the market falls, triggered by the sweeping global tariffs Donald Trump announced last week, still kept him on his toes. Previously a trader for Credit Suisse, he now buys and sells stocks privately. At his laptop in London last week, he watched as the president unveiled a poster board outlining tariff rates, some as high as 50%, for imports from countries around the world. He raced to understand which companies mig...

================================================================================================

Article 2:
Title: Will trade-shy India gain edge in tariff-driven slowdown?
URL: https://www.bbc.com/news/articles/cgrg2lq0gweo
Summary: India's trade detachment may cushion it as others scramble to adjust to Trump’s 90-day tariff pause.
Publication Date: 3 hrs ago
Author: Asia
Full Text: India is the world's fifth-largest and fastest-growing major economy. Yet, a recent legacy of protectionism and inward-focused trade policies have held back its global competitiveness. Itstariffs are highand the share of global exports remains under 2%. India's vast domestic market has fuelled its growth - outpacing many others, economists argue, largely because the rest of the world is slowing. But in a turbulent, increasingly protectionist era, India's instinct for self-reliance may oddly serv...

================================================================================================

Article 3:
Title: World of Business
URL: https://cloud.email.bbc.com/WorldofBusiness_Newsletter_Signup?&at_bbc_team=studios&at_medium=emails&at_objective=acquisition&at_ptr_type=&at_ptr_name=bbc.com&at_format=Module&at_link_origin=homepage&at_campaign=wob&at_campaign_type=owned
Summary: Gain the leading edge with global insights for the boardroom and beyond, in your inbox every Wednesday.
Publication Date: Date unknown
Author: Author unknown
Full Text: Gain the leading edge on business. Global insights and expert analysis for the boardroom and beyond, every Wednesday to your inbox from New York....
```

2. Use Methods from *Task 1*

Full-text cleanup (full_text):

We then **clean the text using the *clean_text* function**, which follows the same principles as the previous one but is adapted for independent reuse in this context.

The key difference with the earlier function is that this one is written to be easily applied in a more general context, making it more modular and reusable across different parts of the project without relying on the previous dataset-specific structure.

Here is how it applies to the first fetched article :

```
Texte original :
As a former champion runner, Richard McDonald can move quickly. But the speed of the market falls, triggered by the sweeping global tariffs Donald Trump announced last week, still kept him o
n his toes. Previously a trader for Credit Suisse, he now buys and sells stocks privately. At his laptop in Lo ...

Tokens nettoyés :
['champion', 'runner', 'richard', 'mcdonald', 'quickly', 'speed', 'market', 'fall', 'trigger', 'sweeping', 'global', 'tariff', 'donald', 'trump', 'announce', 'week', 'keep', 'toe', 'previous
ly', 'trader', 'credit', 'suisse', 'buy', 'sell', 'stock', 'privately', 'laptop', 'london', 'week', 'watch', 'president', 'unveil', 'poster', 'board', 'outline', 'tariff', 'rate', 'high', '5
0', 'import', 'country', 'world', 'race', 'understand', 'company', 'worst', 'hit', 'sell', 'billion', 'wipe']
```

Named Entity Recognition (NER) :

The code reuses the same ***apply_ner_to_articles*** function as before, but with a key difference: instead of applying it to a predefined text, it now processes the *full_text* field of each article in the articles list. The NER model is loaded and applied to each article's content, extracting named entities (ex: persons, organizations). These entities are then added to the article's dictionary under the entities key. The rest of the logic remains unchanged from the previous implementation.

```
Entities detected in article 1 :
('Richard McDonald', 'PER')
('Donald Trump', 'PER')
('Credit Suisse', 'PER')
('London', 'LOC')
('Trump', 'PER')
('Liberation Day', 'MISC')
('US', 'LOC')
('UK', 'LOC')
('Covid-19', 'LOC')
('US', 'LOC')
('Trump', 'PER')
('Trump', 'PER')
('China', 'LOC')
('America', 'LOC')
('European Union', 'ORG')
('Mexico', 'LOC')
```

Here is how it applies to the first fetched article :

→ The NER model successfully identified key entities in the article, including locations, persons, organizations , and miscellaneous terms. While there were some errors (ex: "Jaguar Land Rover" labeled as a person), but overall the model identified the key entities accurately.

Relation Extraction (RE) :

We reuse the previous function structure but now focus on extracting relationships (RE) using spaCy's en_core_web_sm model. The approach is similar, processing each article's full text, but instead of named entities, we identify relationships using syntactic rules:

- Subject-Verb-Object (ex: "Tesla announced a new model")
- Verb-Preposition (ex: "Tesla invested in AI")
- CEO relationships (ex: "Elon Musk is the CEO of Tesla")

These extracted relations are then added to the articles.

```
Relationships extracted for article 1:
('McDonald', 'move_As', 'runner')
('he', 'watch_At', 'laptop')
('he', 'say_In', 'years')
('Trillions', 'wipe', 'value')
('Trillions', 'wipe_in', 'aftermath')
('indexes', 'see', 'some')
('indexes', 'see_since', 'onset')
('worries', 'spread_By', 'Wednesday')
('worries', 'spread_to', 'market')
('he', 'put_on', 'pause')
('Some', 'do_over', 'week')
('investors', 'look_for', 'companies')
('Trump', 'induce_into', 'economy')
('we', 'do_with', 'Apple')
('you', 'look_at', 'What')
('you', 'have_In', 'environment')
('you', 'have', 'tendency')
('We', 'trade_on', 'outlook')
('iPhones', 'cost', 'more')
('iPhones', 'cost_because', 'tariffs')
('Garcia', 'deport_to', 'Salvador')
```

Here is how it applies to the first fetched article :

→ The model successfully extracted multiple **subject-verb-object** relationships from the article. Examples include **"Investors sell shares," "Ferrari announce hike,"** and **"drivers react to tariff."** While some relations are slightly imprecise, the overall structure is well captured. This demonstrates that the RE rules are effectively identifying key relationships in a real BBC article.

Converting to RDF:

The code extends our previous function by structuring the extracted relations into an RDF graph using RDFLib. It follows the same logic but instead of storing relations in a list, it adds them as triples to the graph. To ensure valid URIs, entity names are cleaned by replacing spaces with underscores. Finally, the graph is serialized in Turtle format, making the extracted information machine-readable and structured for further analysis.

Here is how it applies to the first fetched article :

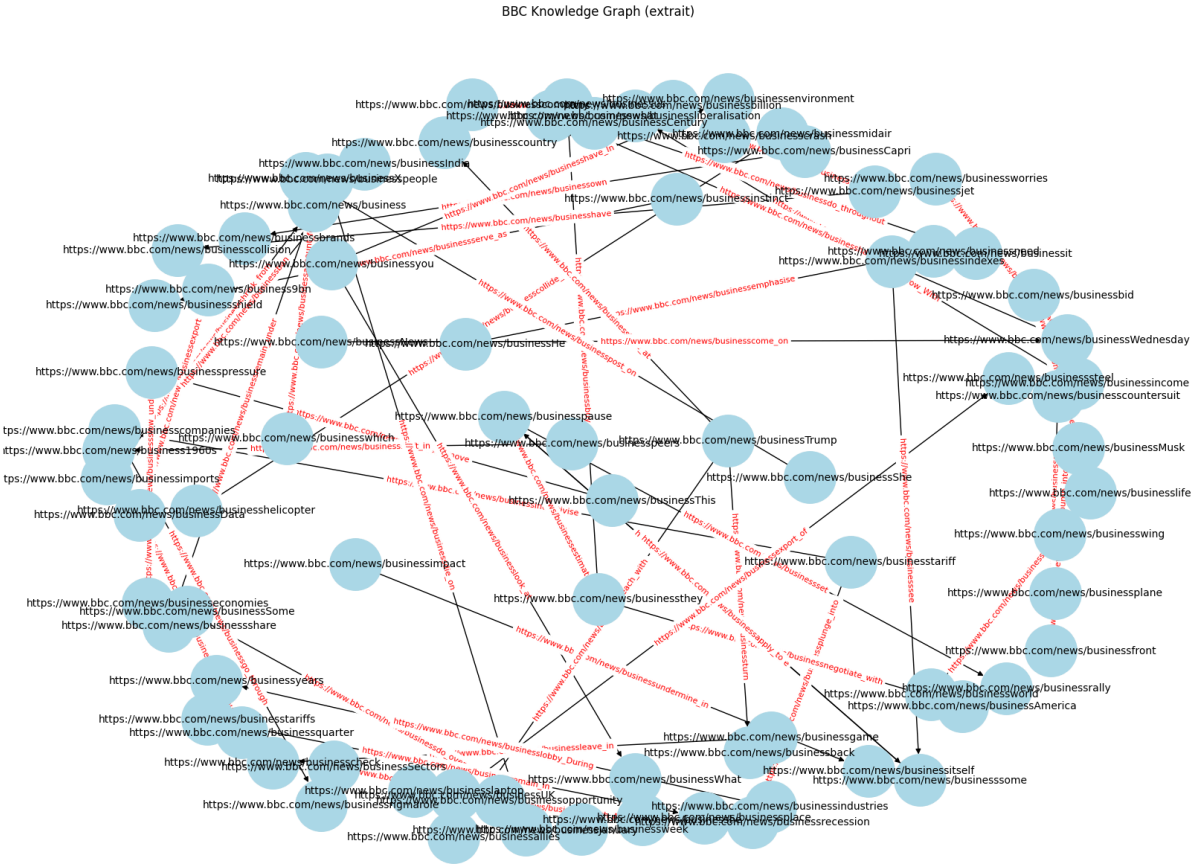
```
@prefix ex: <https://www.bbc.com/news/business> .

ex:1990s ex:note ex:expert ;
    ex:note_In ex:book .

ex:Andcalculations ex:amount_in ex:2024 ;
    ex:amount_to ex:29bn .

ex:Andcountries ex:see ex:break .

ex:BBC ex:approach ex:lawyer ;
    ex:approach_for ex:response .

ex:Capri ex:own ex:brands .

ex:Colombia ex:emerge_as ex:exporter .

ex:Credit ex:see ex:episodes .

ex:Data ex:export ex:9bn ;
    ex:export_around ex:quarter ;
    ex:export_in ex:2024 ;
    ex:export_to ex:US .

ex:Economicshas ex:calculate ex:tariff .

ex:Flight ex:head_to ex:Airport .

ex:Garcia ex:deport_to ex:Salvador .
```

→ The RDF graph has been successfully generated with numerous triplets. It correctly represents the extracted relations from the articles, such as "Investors sell shares," "Ferrari announce hike," and "drivers react to tariff." Each triplet has been properly converted into URIs, structuring the extracted knowledge for further analysis. While most connections are clear, some, like "France vow with minister," may need refinement.

Export to RDF/XML :

Into RDF/XML : `rdf_graph.serialize(destination="output/part1_task2_bbc_graph.rdf", format="xml")`
Into Turtle : `rdf_graph.serialize(destination="output/part1_task2_bbc_graph.ttl", format="turtle")`

RDF graph visualization :



`visualize_rdf_graph_interactive` : This function generates an interactive visualization of the RDF graph using pyvis. It initializes a directed graph, applies the Force Atlas 2 layout algorithm, and then adds nodes and edges based on the extracted RDF triples.

Each subject and object is represented as a node, while each predicate (relation) is added as an edge with a label. To prevent visual overload, the number of edges is limited to 100 by default. Finally, the graph is saved as an HTML file and displayed in the browser.

Pipeline creation :

The pipeline automates and accelerates **knowledge extraction** from **BBC News articles**, converting unstructured text into a structured **RDF knowledge graph**. It scrapes article details, applies **NER** to detect key entities, and extracts **subject-verb-object**

relationships using **spaCy**. The extracted data is structured into an **RDF graph**, exported in Turtle and RDF/XML formats, and visualized both **statically (NetworkX)** and **interactively (PyVis)**. By automating these steps, the process becomes **faster, scalable, and more efficient**, reducing manual effort while ensuring structured knowledge extraction.

We can choose to apply the pipeline to any URL :

```
#url = "https://www.bbc.com/business"
#url = "https://www.bbc.com/culture"
url = "https://www.bbc.com/news/business"
```

(We also tested it on cnn.com, and it works well.)

II. Knowledge Graph Embedding (Part 2)

A. PART 1 – WITHOUT Data Augmentation

1. Data Preparation

Convert RDF graph to PyKEEN format :

We begin by loading the RDF knowledge graph and extracting all its triples. These are then converted into a NumPy array and used to create a TriplesFactory, a format required by PyKEEN for training embedding models. This step ensures the knowledge graph is correctly parsed and ready for link prediction tasks.

Train/Validation/Test Split :

We split the triples into 80% for training, and 10% each for validation and testing. This ensures balanced data for model training and evaluation. Each subset is converted into a TriplesFactory, making them compatible with PyKEEN for the next embedding steps.

2. Model: TransE

Step 1 - Training of TransE with PyKEEN :

We loaded the RDF graph and extracted all subject-predicate-object triples to create a TriplesFactory. Using PyKEEN, we trained a TransE model with an embedding dimension of 50 over 100 epochs. Due to limited hardware, training and evaluation were performed on CPU. The model was trained on 269 triples and evaluated on the validation and test sets.

Step 2 – Entity Similarity (Cosine Distance)

After training, we retrieved the entity embeddings and used cosine similarity to find semantically close entities. Given a target entity, the model identifies others with similar vector representations. For example, it grouped business-related pages like businessmarket

and businessshares near the main business page. This suggests the model captured meaningful semantic relationships, even from an automatically generated RDF graph.

Step 3 – Link Prediction

Automatic Evaluation :

To assess the model's ability to predict links, we rely on standard metrics:

- Mean Rank: Average position of the correct entity (lower is better)
- MRR (Mean Reciprocal Rank): Average of inverse ranks (higher is better)
- Hits@k: Percentage of times the correct entity appears in the top-k predictions

Using PyKEEN's built-in evaluation, our TransE model performs poorly:

- Mean Rank: ~160
- MRR: 0.0062
- Hits@10: 4.4%

This indicates that the model rarely ranks the correct entity among top results.
Why the model performs poorly:

- The graph was automatically built from BBC articles, making it noisy and inconsistent.
- It contains only ~300 triples, which is too small to train meaningful embeddings.
- Relations lack structure and are often too rare or irregular (e.g., say_on, place, get_to).
- The TransE model, being simple, struggles to generalize in this setting.

Impact of the train/test split:

- An 80/10/10 split leaves just ~246 triples for training, limiting learning.
- Key entities or relations may appear only in test/validation, making them unlearnable.
- The model can't predict what it hasn't seen—leading to poor generalization.

Conclusion:

This step reveals the limitations of TransE on a small, noisy, and automatically generated knowledge graph. Better results would likely require more data, cleaner relations, or a more advanced model.

Customised Link Prediction :

In this step, we manually explore the model's ability to predict missing links. Given a known **head entity** and **relation**, the model suggests the most likely **tail entities**, using PyKEEN's predict_target() function.

For example, when we query a triple like:
(https://www.bbc.com/news/business, https://www.bbc.com/news/businessabandon, ?)
the model returns the most probable completions, ranking entities such as:

- https://www.bbc.com/news/businessIndia
- https://www.bbc.com/news/businesswhistleblower
- https://www.bbc.com/news/businessboxes

These predictions make contextual sense, as they are thematically related to the original entity. This shows that the model has learned to group semantically similar concepts, even within a small and noisy graph.

Conclusion:

Despite limited data and some inconsistencies in the graph, the model captures certain semantic regularities. The embeddings allow meaningful link suggestions between related entities, highlighting the potential of knowledge graph embeddings for inference and discovery tasks.

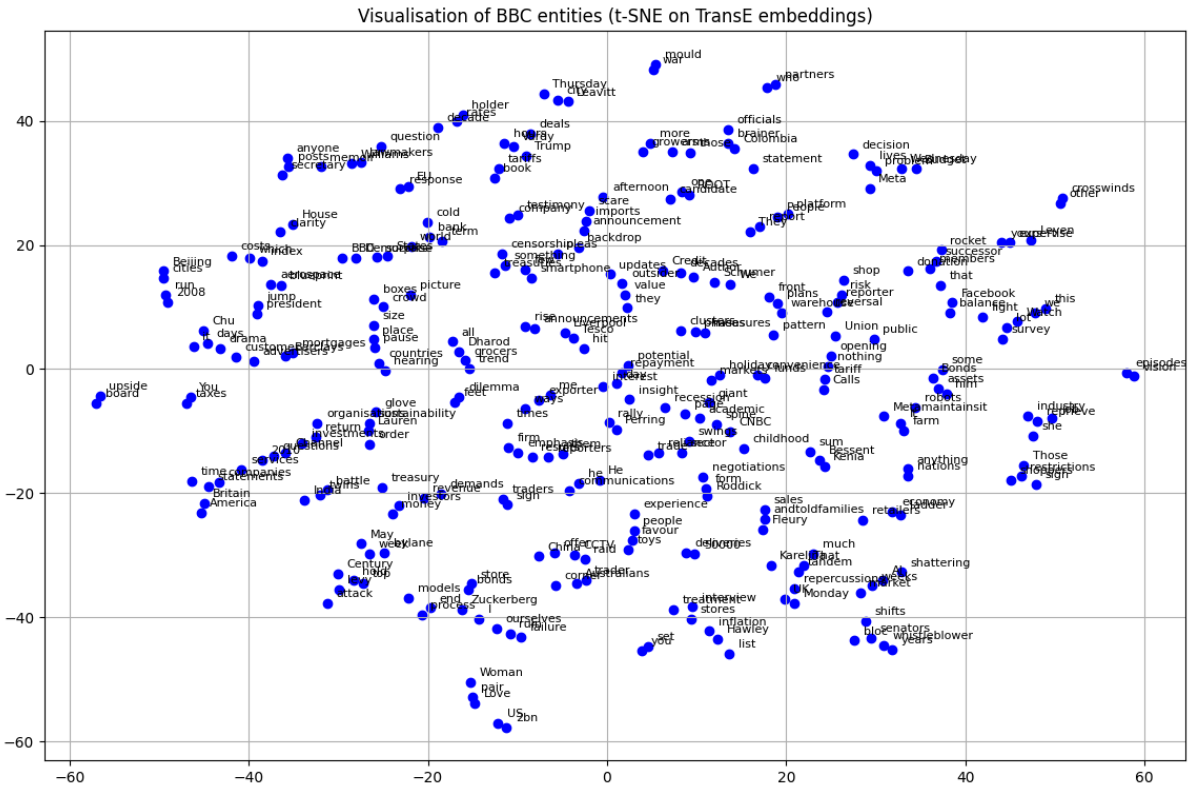
Step 4 - Embedding Visualization (with t-SNE) :

In this step, we visualize the entity embeddings learned by the TransE model:

1. **Embedding Extraction:**
   We first extract the vector representations (embeddings) of all entities from the trained model. These high-dimensional vectors capture semantic information based on the graph structure.

2. **Dimensionality Reduction (t-SNE):**
   Since embeddings are in a high-dimensional space (e.g., 50 dimensions), we reduce them to 2D using **t-SNE**, a technique well-suited for visualizing clusters and local patterns. We set perplexity=5 to reflect the small dataset size.

3. **Visualization:**
   Each entity is plotted as a point in 2D space. For clarity, we clean the labels by removing the repetitive URL prefix. This makes it easier to read and identify clusters or related concepts in the business news domain.



We're now going to do exactly the same visualisation, but in an interactive version using Plotly, so that we can zoom in, hover over and move the points around.

In this step, we create an **interactive 2D visualization** of the entities learned by the TransE model:

1. **Embeddings & Labels:**

   We extract the entity embeddings and their corresponding labels from the trained model. These vectors represent each entity in the knowledge graph based on its structural and semantic role.

2. **Label Cleaning:**

   For readability, we strip the repeated part of the URL (https://www.bbc.com/news/business) from each entity label. If nothing remains (i.e., the root node), we label it as "ROOT".

3. **Dimensionality Reduction with t-SNE:**

   We apply t-SNE to reduce the high-dimensional embeddings (e.g., 50D) to 2D for

visualization. We use a low perplexity value (5), appropriate for small datasets, to better reveal local structure.

4. **DataFrame for Plotting:**

   We organize the 2D coordinates, clean labels, and full URLs into a DataFrame to be used by Plotly.

5. **Interactive Scatter Plot (Plotly):**

   Using Plotly Express, we create an interactive scatter plot:

   - Each point represents an entity.
   - The short label is shown directly on the plot.
   - The full URL appears on hover for detailed inspection.
   - The layout is customized for better readability and exploration.

The t-SNE visualization shows that the entities extracted from the BBC website are grouped by theme. For example, terms related to the economy such as *market*, *investors*, *falls*, or *shares* appear close to one another. Other areas bring together geopolitical entities or organizations like *EU*, *China*, *administration*, *government*, or *IMF*. This suggests that the TransE model has learned to represent entities based on their semantic context within the RDF graph, even though the graph remains relatively noisy and automatically generated. This type of visualization is useful for understanding how entities are structured and perceived by the model.

3. Model: DistMult

Step 1 – Training DistMult with PyKEEN

We train the **DistMult** model using the same train/validation/test splits as for TransE. It uses a bilinear scoring function, which is better suited for symmetric relations.

- **Settings:** embedding size = 50, CPU training, fixed seed.
- **Training:** Fast due to the small graph; only 5 epochs were run.
- **Evaluation:** Performed automatically after training using standard link prediction metrics (MRR, Hits@k, etc.).

This sets up a fair comparison with TransE to evaluate how well DistMult handles our noisy, article-based RDF graph

Step 2 – Entity Similarity (DistMult)

After training DistMult, we extract the entity embeddings and compute cosine similarity between them to identify semantically close entities.

- **How:** We select a known entity (e.g., the BBC Business homepage) and retrieve its nearest neighbors in the embedding space using cosine similarity.
- **Result:** Entities like *businesssurprise*, *businesslawmakers*, or *businesssuccessor* are identified as close.
- **Insight:** This shows that DistMult also captures some semantic regularities between related economic topics, though the proximity may vary slightly compared to TransE due to its symmetric nature.

This step confirms that embeddings trained with DistMult can reflect contextual closeness between business-related entities.

Step 3 – Link Prediction with DistMult :

3.1 Automatic Evaluation

The performance of the DistMult model is evaluated with the following metrics:

- **Mean Rank**: 139.09
- **MRR (Mean Reciprocal Rank)**: 0.0072
- **Hits@1 / @3**: 0.0000 (the model never ranks the correct entity in the top results)
- **Hits@10**: 0.0588 (only around 6% of correct entities appear in the top 10)

DistMult performs slightly better than TransE, but its predictions are still weak. It struggles to accurately predict links due to the noisy and unstructured graph.

3.2 Custom Link Prediction

The model is asked to predict possible target entities given a specific head entity and relation.

Example:

For the entity https://www.bbc.com/news/business and relation businessabandon, DistMult predicts entities like businessraid, businessshe, businesspublic, etc.

Observations:

- Predictions are close to each other, indicating uncertainty in the model's choices.
- The suggested entities are somewhat thematically related, but the model fails to capture strong semantic links between the head entity and the predicted ones.
- DistMult shows some understanding of the graph structure but lacks precision in predicting meaningful relationships.

DistMult shows a better grasp of graph structures than TransE, but its predictions remain vague and imprecise, particularly in noisy, unstructured graphs.

Step 4 - Embedding Visualization (with t-SNE):

In this step, we visualize the entity embeddings from the DistMult model using t-SNE to reduce the high-dimensional embeddings to 2D for easier interpretation. After extracting the embeddings and labels, we use Plotly to create an interactive scatter plot, displaying how entities are grouped based on their learned representations.

The t-SNE plot shows some thematic clustering, such as groups related to economics, countries, and business entities. This suggests that the model has learned some structure in the relationships between entities. However, the clusters are not well-defined, indicating that the DistMult model still struggles with fine-grained organization of the entities. The overall representation remains partial, showing that while the model captures some patterns, its understanding of the graph is still limited.

4. Performance Comparison

Performance Comparison

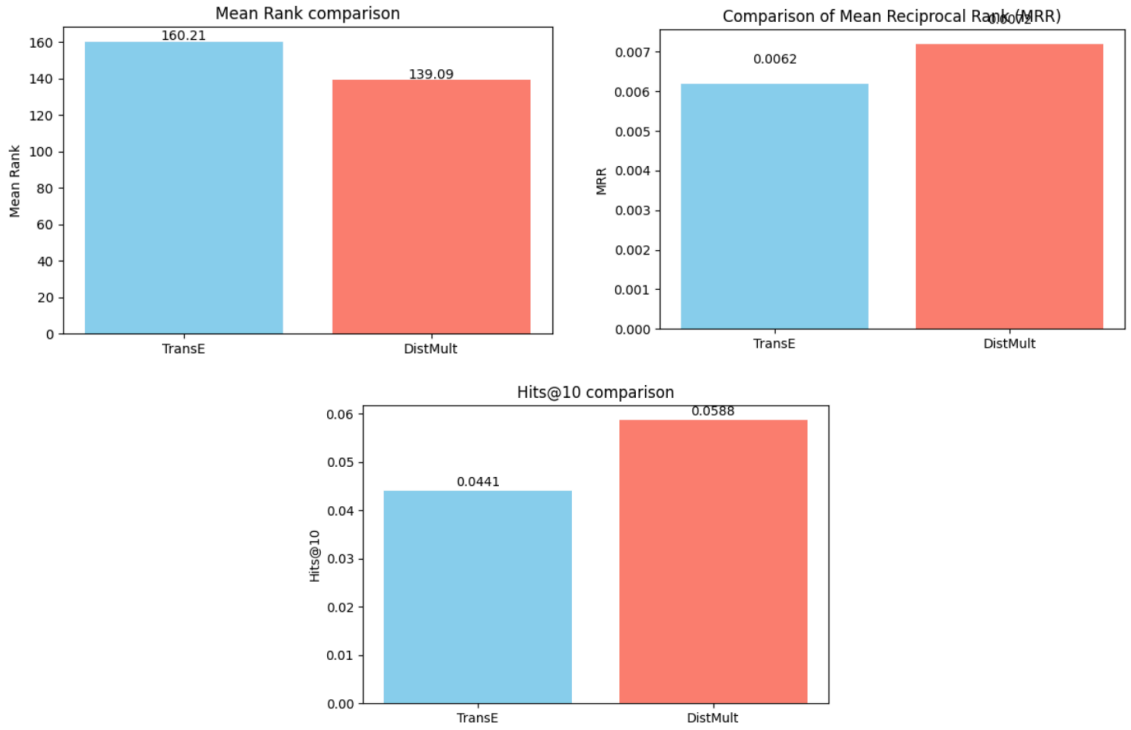
In this step, we compare the performance of two models, TransE and DistMult, using key evaluation metrics: *Mean Rank*, *Mean Reciprocal Rank (MRR)*, and *Hits@10*. The comparison is performed by extracting these metrics from the evaluation results of each model and presenting them in a table format.

The create_comparison_table function generates a table displaying these metrics for each model. After running the function, we get a side-by-side comparison of how each model performs across the three chosen metrics. This allows us to visually assess the strengths and weaknesses of both models.

	arithmetic_mean_rank	inverse_arithmetic_mean_rank	hits_at_10
TransE	160.2059	0.0062	0.0441
DistMult	139.0882	0.0072	0.0588

The results show that DistMult outperforms TransE, as it has a lower *Mean Rank* (139.09 vs. 160.21), indicating better overall ranking of correct entities. Additionally, DistMult achieves a slightly higher *Mean Reciprocal Rank* (0.0072 vs. 0.0062) and a higher *Hits@10* (0.0588 vs. 0.0441), suggesting that DistMult is better at predicting correct entities within the top 10 rankings.

Let's do a more telling visual analysis:



The DistMult model outperforms TransE across all metrics. It achieves a lower Mean Rank (129.02 vs. 150.08), meaning that, on average, it ranks the correct answers higher. Its MRR (0.0078) is also slightly better than that of TransE (0.0067), indicating it is a bit more accurate in its rankings. Finally, it successfully places the correct answer in the top 10 in 4.84% of cases, compared to just 1.61% for TransE.

Although the scores remain low overall, DistMult demonstrates a better ability to capture relationships within the graph, particularly due to its bilinear structure, which is better suited for noisy contexts like the BBC articles

Qualitative analysis: good and bad predictions :

In this qualitative analysis, we compare predictions made by the **TransE** and **DistMult** models, focusing on both good and bad predictions.

For **TransE**, a good prediction involves the entity https://www.bbc.com/news/business being linked to itself via the relation add_to, which, while odd, could reflect a logical loop within the graph structure. Its bad prediction (businesscomparison) is still related to the "business" domain but lacks precision in distinguishing between concepts. This shows that TransE is simple and conservative in its predictions, often sticking to generic entities.

For **DistMult**, the model's good prediction links the "business" entity to businessterritory, which is more thematically relevant, suggesting a geographical extension of the business domain. The bad prediction (businesschildren), however, is a mismatch, as "children" has

little association with "business," which shows that while DistMult makes more conceptually varied predictions, some predictions can still be nonsensical.

Overall, DistMult demonstrates a higher degree of conceptual differentiation in its predictions compared to TransE, which remains more basic, reflecting the structural differences between the models: TransE is based on vector translations, while DistMult captures symmetries and co-occurrences more effectively.

B. PART 2 – WITH Data Augmentation

1. DBpedia Enrichment

In this section, we focus on enriching the RDF graph we obtained from the BBC articles using external data from DBpedia. The goal is to improve the quality of the data, enhancing the graph's semantic richness by adding additional facts about the entities present in the graph.

1. Enriching Entities with DBpedia

We start by retrieving related facts about entities from DBpedia. The DBpedia SPARQL endpoint allows us to query DBpedia's structured data to retrieve facts such as entity types, relationships, and related entities. For instance, if the entity in our graph is "Barack Obama," we can use DBpedia to gather additional information about him, such as his profession, related events, and other entities linked to him (e.g., "President of the United States").

2. Extracting Entities from the Graph

Once the RDF graph is loaded, we extract the entities that are present as either subjects or objects in the graph. These entities can be anything from companies like "Apple" or "Microsoft" to concepts like "business trends" or "economic measures." However, since the entities extracted from the BBC graph tend to have a prefix like "business" (e.g., "businessmeasures," "businesstrend"), they may not match the corresponding entities in DBpedia directly.

3. Querying DBpedia with Cleaned Entity Labels

The next challenge is to clean and map the extracted entities to known entities in DBpedia. To achieve this, we clean the entity names by removing irrelevant prefixes like "business," which are specific to the BBC articles. After cleaning the entity names (e.g., "businessmeasures" becomes "measures"), we perform SPARQL queries to retrieve relevant data from DBpedia. If a match is found in DBpedia, we fetch the related facts.

4. Handling Entities that Don’t Match DBpedia

Not all entities in our graph will have a direct counterpart in DBpedia. For instance, terms like "businessfailure" or "businessthey" may not exist as formal entities in DBpedia. In such cases, we may choose to ignore these entities or handle them using more advanced techniques such as entity linking (mapping to known entities using NLP tools like spaCy). For now, we start by simply removing the "business" prefix to see if we can match the resulting terms to known entities in DBpedia.

5. Enriching the Graph with DBpedia Triples

Once we identify matching entities, we add new RDF triples to the graph. For each matching entity, we query DBpedia to retrieve related facts such as its type, links to other entities, and related concepts. These triples are added to the RDF graph, expanding the graph's semantic depth.

6. Integrating and Exporting the Enriched Graph

After adding new triples from DBpedia, we export the enriched RDF graph. This enriched graph now contains additional knowledge, which improves its potential for tasks such as embedding generation and semantic analysis. The addition of new triples increases the graph's connectivity, providing more context and relationships between entities.

Results of Data Augmentation

After cleaning the entities and querying DBpedia, we successfully added 649 new triples to the RDF graph. This significantly increased the density and richness of the graph. The final enriched graph is saved as an RDF file, ready for further analysis or use in machine learning tasks.

By augmenting the RDF graph with data from DBpedia, we've made the graph more comprehensive, which helps in improving the performance of downstream tasks, such as entity recognition and relationship extraction, through better semantic knowledge representation.

2. Retraining Pipeline

This pipeline enriches an RDF graph by integrating external data from DBpedia.

Step 1: Clean Entities

We load the original RDF graph and extract all entities. We clean the entity labels by removing prefixes like "business" and punctuation to create a list of 372 cleaned entities.

Step 2: Query DBpedia

For each cleaned entity, we query DBpedia using SPARQL to find related facts (e.g., types or relationships) and collect the results.

Step 3: Integrate into RDF Graph

We add the retrieved triples (subject, predicate, object) to the original RDF graph, enriching it with 649 new triples.

Step 4: Export Enriched RDF Graph

The enriched RDF graph, now containing 665 triples, is saved for further use. This process enhances the graph's density and semantic value through data augmentation.

C. PART 3 – Comparison WITH vs WITHOUT DA

Metric Table :

We compare the results of four models: TransE (original), DistMult (original), TransE (with data augmentation), and DistMult (with data augmentation).

The table extracts the following metrics for comparison:

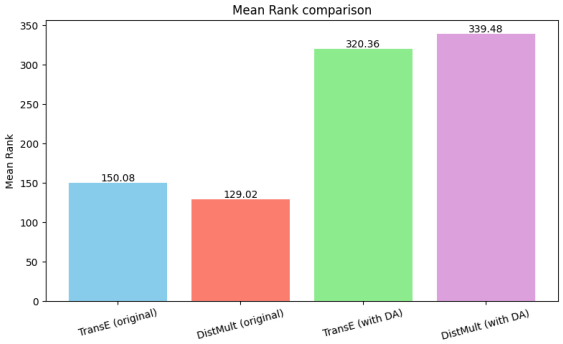
- Arithmetic Mean Rank
- Inverse Arithmetic Mean Rank
- Hits at 10

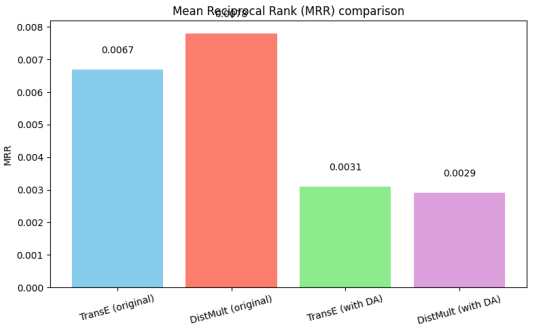
We create a comparison table by extracting these metrics for each model and rounding the results to four decimal places. The final table presents the model performance across the specified metrics.

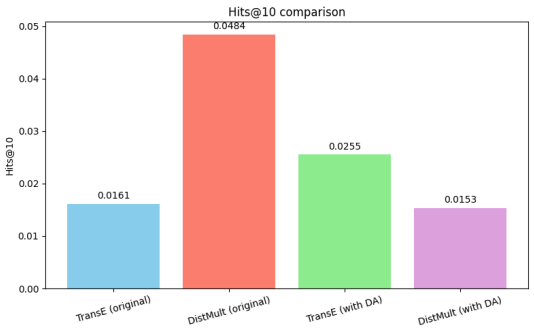
	arithmetic_mean_rank	inverse_arithmetic_mean_rank	hits_at_10
TransE (original)	160.2059	0.0062	0.0441
DistMult (original)	139.0882	0.0072	0.0588
TransE (with DA)	389.7879	0.0026	0.0051
DistMult (with DA)	408.8333	0.0024	0.0101

The results show that the original models (TransE and DistMult) outperform the data-augmented versions in all metrics. **DistMult (original)** achieves the best performance with a mean rank of 139.0882 and hits at 10 of 0.0588. After applying data augmentation, both **TransE (with DA)** and **DistMult (with DA)** show a significant decline in performance, with higher mean ranks and lower hits at 10. This suggests that the data augmentation strategy did not improve model performance.

Bar charts :







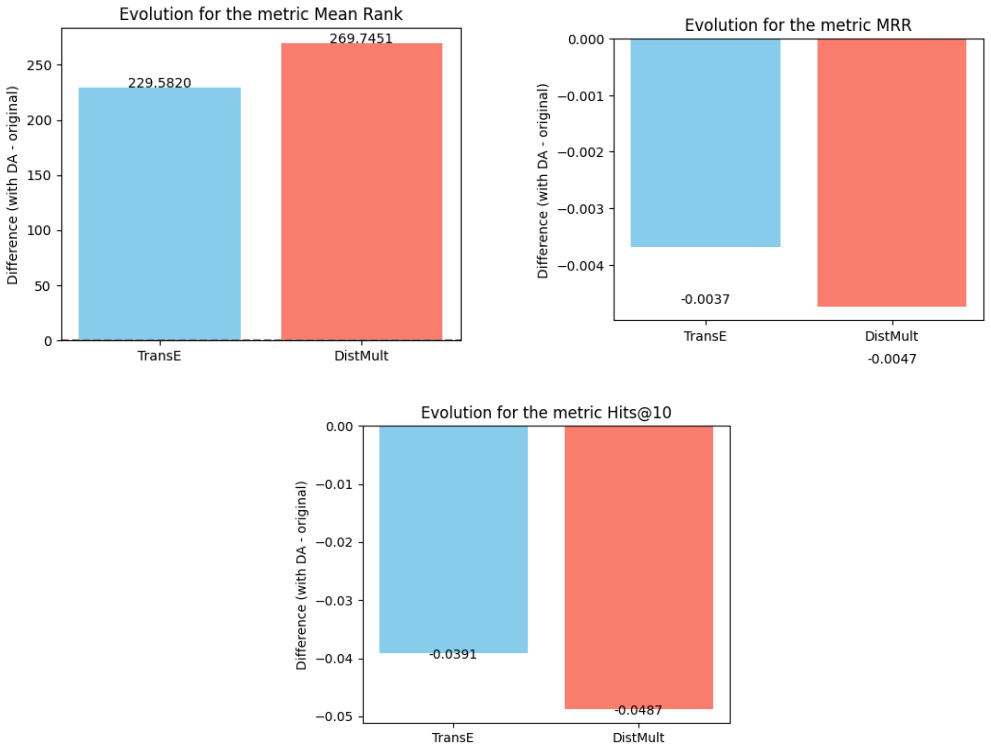
The results show quite an unexpected behavior regarding the impact of data augmentation on model quality.

Without data augmentation, the original DistMult achieves the best overall performance: its arithmetic mean rank is the lowest (129.01), meaning that the correct answers are, on average, better ranked. It is also the only model to reach a relatively high hits@10 (4.84%), indicating that it manages to place correct triplets in its top 10 predictions more often than the others.

On the other hand, adding external data through DBpedia does not seem to have improved the results. Quite the opposite, the performance has deteriorated for both models. The mean rank increases significantly for both TransE (from 150.08 to 320.36) and DistMult (from 129.01 to 339.48), indicating a decrease in precision. MRR and hits@10 also follow this trend, both dropping after the integration of DBpedia triples.

This suggests that the added entities or relations have likely introduced noise or semantic ambiguities, making the learning of representations more difficult. Naive enrichment, without proper disambiguation, can thus harm the quality of the embeddings. A more selective filtering strategy or more robust linking would be necessary to leverage data augmentation effectively.

Before/After differences :



The visualizations clearly show that adding data through DBpedia (data augmentation) has had a negative impact on the performance of both models, TransE and DistMult.

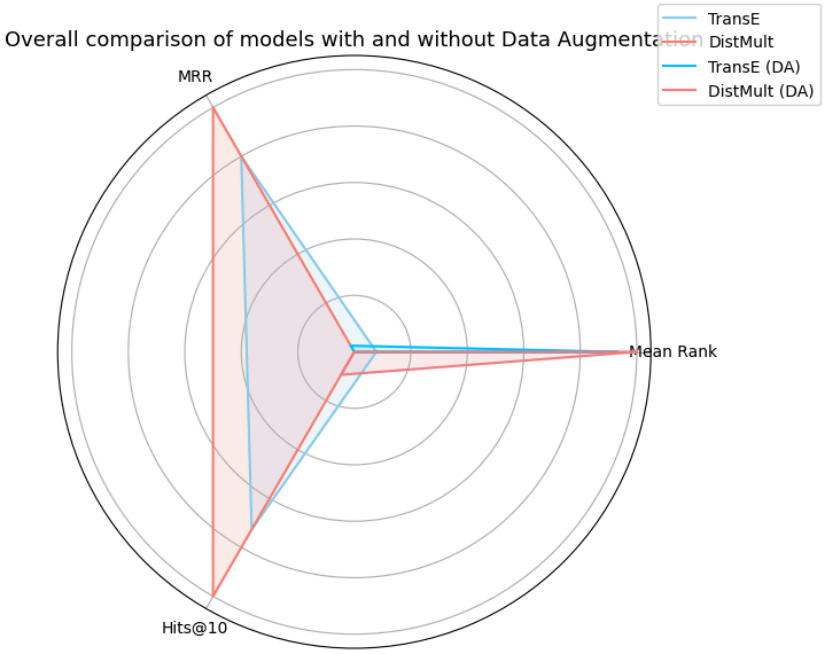
For the **Mean Rank** metric, there is a noticeable degradation after enrichment, with an increase of 170 points for TransE and over 210 points for DistMult. This indicates that the correctly predicted entities are now further down the rankings, reflecting a decrease in precision.

Regarding the **MRR** (Mean Reciprocal Rank), both models also experience a slight decline. DistMult is more affected, losing 0.0048, which could indicate that it had a better ability to capture the original structure of the graph, but the addition of external information introduced noise.

Finally, for the **Hits@10** metric, the effects are contrasting. TransE's score improves slightly, suggesting that it has learned new interesting associations through the enrichment. On the other hand, DistMult's score drops significantly, losing over 3 percentage points, which suggests that the model was disrupted by the new entities and relations added.

Overall, these results suggest that data augmentation does not always have a beneficial effect. While it can enrich the graph, it can also introduce ambiguity or noise, especially if the integration is not accompanied by thorough semantic cleaning or filtering.

Spider chart :



The spider chart provides a **holistic comparison** of the performance across all four model configurations—**TransE**, **DistMult**, and their augmented counterparts—based on three key metrics: **Mean Rank**, **MRR**, and **Hits@10**.

It's evident that the **original TransE** (light blue) demonstrates a balanced performance, forming a decently shaped polygon, particularly along the **MRR** and **Hits@10** axes. However, when we look at **TransE (DA)** (dark blue), the shape collapses inward, particularly on the **MRR** and **Hits@10** dimensions, indicating a **notable drop in precision** due to the data augmentation.

On the other hand, **original DistMult** (light red) stands out with the most expansive coverage in the chart. It performs **exceptionally well** on **MRR** and **Hits@10**, giving it a strong, outward-stretched shape. But this advantage diminishes significantly for **DistMult (DA)** (dark red), which shows a **major contraction**, particularly on **Hits@10**, where the drop is sharp.

In summary, this radar plot visually confirms that **data augmentation negatively impacted both models' performance**, especially **DistMult**, which originally performed the best. The **shrinking** of the polygons after augmentation serves as a **clear visual indicator** of degraded results.

D. PART 4 – Conclusion

1- Reflections

When comparing the models with and without data augmentation using DBpedia, one might expect that adding external knowledge would naturally improve performance across all metrics. However, our experiments revealed the opposite. In both TransE and DistMult models, the introduction of data augmentation led to a deterioration in key performance

metrics such as Mean Reciprocal Rank (MRR) and Hits@10, and a noticeable increase in Mean Rank. This was surprising, as the enriched graph theoretically contains more semantic context and a broader set of relationships that could benefit link prediction and entity similarity tasks.

One possible explanation for this drop in performance is the nature of the added data. The external triplets retrieved from DBpedia may not have been tightly aligned with the core structure or semantics of the BBC knowledge graph we originally built. Without precise entity linking or semantic filtering, the integration of loosely related or noisy entities could have introduced ambiguity, making it harder for the embedding models to capture consistent patterns. Additionally, the new entities introduced by DBpedia might have disrupted the local neighborhoods of important nodes, resulting in embeddings that are more dispersed and less informative for specific tasks. This shows that data augmentation must be handled carefully, and that quantity alone does not guarantee quality.

2- Final Conclusion

In this second part of the project, focused on Knowledge Graph Embedding, we explored how to generate vector representations of entities from our RDF graph using two popular embedding models: TransE and DistMult. After converting our RDF graph into a format compatible with PyKEEN, we split the data into training, validation, and testing sets. We then trained both models, evaluated their performance using standard metrics (Mean Rank, MRR, Hits@10), and visualized the embeddings using t-SNE. We also performed qualitative analyses to interpret both correct and incorrect predictions made by each model, providing deeper insight into their behavior.

In the second half of the pipeline, we introduced data augmentation by enriching our graph with external knowledge extracted from DBpedia via SPARQL queries. We selected candidate entities, cleaned them, integrated new triplets into the original RDF graph, and repeated the entire embedding pipeline on the extended dataset. This allowed us to rigorously compare the impact of external knowledge on embedding quality, both quantitatively (through metric comparison and visualizations) and qualitatively (via prediction examples and embedding maps).

Overall, the process allowed us to understand not only how different models behave, but also how knowledge integration from heterogeneous sources can affect downstream embedding performance.

3- Possible Improvements

There are several ways this project could be further improved. First and foremost, the data augmentation strategy could benefit from more precise entity linking. Instead of using simple string cleaning and name matching, we could incorporate Named Entity Disambiguation tools to map local entities to DBpedia or Wikidata entities more accurately. This would help avoid adding irrelevant or weakly related facts to the graph.

Additionally, we could apply more advanced filtering mechanisms to keep only semantically strong and useful relationships during the enrichment phase. On the modeling side, experimenting with more recent or complex embedding algorithms (such as RotatE, ConvE,

or ComplEx) might yield better performance, especially on enriched graphs. We could also introduce hyperparameter optimization to fine-tune the model settings for each dataset.

Finally, moving beyond static embeddings, incorporating temporal or contextual dimensions—especially for news-related graphs—could open new avenues for capturing evolving relationships and trends over time.