Part 2: Knowledge Graph Embedding

Duration: 3 sessions(4.5H) / 6 sessions (9h)

Objective:
In this part, you will practise how to create and evaluate knowledge graph embeddings from your constructed knowledge graph. You will address the challenge of limited entity count through data augmentation techniques.

As usual, example codes are provided below. These codes are functional but may not be pertinent. You need to adapt them to your own case.

Knowledge Graph Embedding

1. **Setup PyKEEN:**

```
pip install pykeen torch
```

2. **Basic Embedding Pipeline:**

```
from pykeen.pipeline import pipeline
from pykeen.triples import TriplesFactory

# Convert your RDF triples to PyKEEN format
triples = [(s, p, o) for s, p, o in g]
tf = TriplesFactory.from_labeled_triples(triples)

# Train a TransE model
results = pipeline(
    training=tf,
    model='TransE',
    epochs=100,
    learning_rate=0.01,
    training_batch_size=128,
    random_seed=42,
)

# Access embeddings
entity_embeddings =
results.model.entity_embeddings.weight.detach().numpy()
relation_embeddings =
results.model.relation_embeddings.weight.detach().numpy()
```

3. **Evaluation:** A more detailed introduction is given below in Detailed Embedding Evaluation Guide

```
# Evaluate entity similarities
from sklearn.metrics.pairwise import cosine_similarity

def find_similar_entities(entity_id, entity_embeddings, top_k=5):
    entity_vector = entity_embeddings[entity_id].reshape(1, -1)
    similarities = cosine_similarity(entity_vector, entity_embeddings)
    most_similar = np.argsort(similarities[0])[-top_k-1:-1][::-1]
    return most_similar

# Link prediction
results.evaluate()
```

Tips for Small Knowledge Graphs

1. **Model Selection:**
    - Use simpler models like TransE or DistMult for small datasets
    - Avoid complex models that require large amounts of training data
2. **Training Configuration:**
    - Increase the number of negative samples
    - Use smaller embedding dimensions (e.g., 50-100 instead of 200+)
    - Implement early stopping to prevent overfitting
3. **Example Configuration:**

```
from pykeen.pipeline import pipeline

results = pipeline(
    training=tf,
    model='TransE',
    epochs=100,
    embedding_dim=50,
    learning_rate=0.01,
    training_batch_size=32,
    num_negs_per_pos=10,
    early_stopping=True,
    early_stopping_patience=5,
    random_seed=42,
)
```

4. **Quality Assessment:**

- Compare embeddings before and after data augmentation
- Use visualization tools to inspect entity clusters
- Validate predictions against domain knowledge

Challenge: Limited Entity Count

The knowledge graph constructed from news articles might have a limited number of entities, which can affect the quality of embeddings. To address this:

1. **Data Augmentation Strategies:**
    - Use external knowledge bases like DBpedia to add related entities and relations
    - Implement entity linking to map your entities to Wikidata/DBpedia entities Here is an example code. You may need to improve it with entitiy linking to avoid ambiguity brought by the entity names. (see slide 82 in Lecture 5: Web mining and Information retrieval for knowledge graph building)
2. **Integration with External Knowledge:** Here is an example of how to extract facts from DBpedia knowledge base, by SPARQL query.

You should develop your knowledge graph integration solution adapting to your own data. You can also integrate more entities from external sources, such as 2 or 3 degrees of connections.

```
from SPARQLWrapper import SPARQLWrapper, JSON

def enrich_with_dbpedia(entity_name):
    sparql = SPARQLWrapper("http://dbpedia.org/sparql")
    query = """
    SELECT DISTINCT ?related ?relation WHERE {
        ?s rdfs:label "%s"@en .
        ?s ?relation ?related .
        ?related rdfs:label ?label .
        FILTER(LANG(?label) = 'en')
    } LIMIT 10
    """ % entity_name

    sparql.setQuery(query)
    sparql.setReturnFormat(JSON)
    results = sparql.query().convert()
    return results["results"]["bindings"]
```

Detailed Embedding Evaluation Guide

Please Refer to this tutorial for detailed information on knowledge graph embedding evaluation with framework `pykeen`. The instructions below is a general pipeline that you can follow.

1. **Dataset Splitting**

```
from pykeen.triples import TriplesFactory

# Split your triples into training (80%), validation (10%), and test (10%)
sets
training, validation, testing =
TriplesFactory.from_labeled_triples(triples).split([0.8, 0.1, 0.1])
```

```
print(f"Training triples: {training.num_triples}")
print(f"Validation triples: {validation.num_triples}")
print(f"Testing triples: {testing.num_triples}")
```

2. **Model Training with Multiple Architectures**

```
from pykeen.pipeline import pipeline

# Dictionary to store results
model_results = {}

# Test different models
models = ['TransE', 'DistMult', 'ComplEx']
for model_name in models:
    results = pipeline(
        training=training,
        validation=validation,
        testing=testing,
        model=model_name,
        epochs=100,
        embedding_dim=50,
        training_kwargs=dict(batch_size=32),
        random_seed=42,
    )
    model_results[model_name] = results
```

3. **Evaluation Metrics**

```
def evaluate_model(results, model_name):
    metrics = results.metric_results.to_dict()
    print(f"\nResults for {model_name}:")
    print(f"Mean Rank: {metrics['both']['mean_rank']:.2f}")
    print(f"Mean Reciprocal Rank: {metrics['both']
['mean_reciprocal_rank']:.4f}")
    print(f"Hits@1: {metrics['both']['hits_at_1']:.4f}")
    print(f"Hits@3: {metrics['both']['hits_at_3']:.4f}")
    print(f"Hits@10: {metrics['both']['hits_at_10']:.4f}")

for model_name, results in model_results.items():
    evaluate_model(results, model_name)
```

4. **Entity Similarity Analysis**

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

def analyze_entity_neighborhood(model, entity_id, k=5):
```

```
    # Get entity embeddings
    entity_embeddings = model.entity_embeddings.weight.detach().numpy()
    entity_labels = model.triples_factory.entity_labels

    # Calculate similarities
    similarities = cosine_similarity([entity_embeddings[entity_id]],
entity_embeddings)[0]
    most_similar = np.argsort(similarities)[-k-1:-1][::-1]

    print(f"\nMost similar entities to {entity_labels[entity_id]}:")
    for idx in most_similar:
        print(f"{entity_labels[idx]}: {similarities[idx]:.4f}")
```

5. **Link Prediction Examples**

```
def predict_tail_entities(model, head, relation, k=5):
    # Get predictions
    predictions = model.predict_scores(
        heads=torch.tensor([head]),
        relations=torch.tensor([relation]),
    )

    # Get top k predictions
    top_tails = torch.topk(predictions, k=k, dim=1)

    return [(model.triples_factory.entity_labels[idx.item()],
             score.item())
            for idx, score in zip(top_tails.indices[0],
top_tails.values[0])]
```

6. **Visualization of Embeddings**

```
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

def visualize_embeddings(model, entity_types=None):
    # Get embeddings and reduce dimensionality
    embeddings = model.entity_embeddings.weight.detach().numpy()
    tsne = TSNE(n_components=2, random_state=42)
    reduced_embeddings = tsne.fit_transform(embeddings)

    # Plot
    plt.figure(figsize=(10, 10))
    if entity_types is None:
        plt.scatter(reduced_embeddings[:, 0], reduced_embeddings[:, 1],
alpha=0.5)
    else:
        # Color by entity type
        for entity_type, indices in entity_types.items():
```

```
            plt.scatter(reduced_embeddings[indices, 0],
                        reduced_embeddings[indices, 1],
                        label=entity_type, alpha=0.5)
        plt.legend()

    plt.title("Entity Embeddings Visualization")
    plt.show()
```

7. **Performance Comparison Template**

```
def create_comparison_table(model_results):
    metrics = ['mean_rank', 'mean_reciprocal_rank', 'hits_at_10']
    comparison = {model: {} for model in model_results.keys()}

    for model_name, results in model_results.items():
        for metric in metrics:
            comparison[model_name][metric] =
results.metric_results.to_dict()['both'][metric]

    return pd.DataFrame(comparison).round(4)
```

For your final report, include:

- Performance metrics for at least 2 different embedding models
- Examples of successful and failed link predictions
- Impact of data augmentation on embedding quality

Evaluation Criteria

Criterion	Notes	Points
Basic Implementation	- Functional web scraping script with at least 10 articles - Clean text preprocessing (no HTML tags, proper sentence structure) - Complete pipeline from scraping to graph construction	2
Named Entity Recognition	- Number of unique entities identified (>50) - Comparison between model CRF and Spacy `conll23`	4
Relation Extraction	- Basic relation extraction implementation - At least 3 custom rules (with examples extracted from the text)	3
Knowledge Graph Quality	- All triples in valid RDF format using proper namespaces - At least 50 meaningful triples in the graph	2

Criterion	Notes	Points
Entity Linking	- Working DBpedia/Wikidata linking implementation - >50% entities linked to external knowledge bases - Documentation of disambiguation strategy	2
Knowledge Graph Embedding	- Model training with at least 2 different architectures - How the performance is improved by data enhencement	3
Link Prediction	- Evaluation using standard metrics (MRR, Hits@k)	2
Documentation & Analysis	- Detailed methodology in report - Quantitative performance analysis	2

Total points: 20

Note:

- The points distribution emphasizes both implementation quality and results
- Partial points can be awarded based on completion level

Submission Guidelines

1. **Code Repository**
    - Jupyter notebooks must be clean and well-commented
    - The criteria above should be highlighted in the results
2. **Technical Report**
    - Project overview and methodology
    - Quantitative results for each criterion (such as reconized entities, extracted relations, evaluation of KG embedding.)
    - Analysis of challenges and solutions
    - Examples of extracted knowledge
    - Visualizations of the knowledge graph
    - Maximum 10 pages, PDF format

Bonus Points Opportunities

- Novel relation extraction rules with >80% precision (+0.5)
- Integration with multiple external knowledge bases (+0.5)