Extracting Keyphrases and Relations from Scientific Publications

Team No - 44
Aman Khandelwal (2022201010)
Divyansh Negi (2022201014)
Sumonto Chatterjee (2022201028)

Project Description:

The project focuses on the extraction of keyphrases and relations from scientific publications. This task is essential for understanding the content of research articles, specifically identifying processes, tasks, and materials described in the publications. The challenge lies in the complexity of the scientific text, where keyphrases can vary significantly between domains, lack clear indicators, and consist of multiple tokens. Automatically identifying crucial words or phrases and their interconnections within scientific publications is essential for summarizing the primary topics or concepts discussed. This helps researchers and readers in quickly grasp the core content and contributions of a publication, facilitating more efficient information retrieval and literature exploration.

Datasets:

There are three benchmark datasets commonly used for automatic keyphrase extraction from scientific articles.

● **Semval 2010-SemEval-2010**: Automatic Keyphrase Extraction from Scientific Articles: This dataset was created for the SemEval-2010 Task 5 challenge and contains annotated keyphrases for 244 scientific articles from various domains.

- Source: Created for a specific challenge (SemEval-2010 Task 5).
- Size: 244 scientific articles.
- Content: Articles from various scientific domains with annotations for keyphrases.
- Use Case: Ideal for getting started or exploring keyphrase extraction in a variety of scientific fields due to its domain diversity.

● **Semval 2017-ScienceIE 2017 dataset**: It consists of 500 journal articles evenly distributed among the domains of Computer Science, Material Sciences, and Physics. Three types of documents are provided: plain text documents with sampled paragraphs, brat .ann standoff documents with annotations for those paragraphs, and .xml documents with the original full

article text. The training data part of the corpus consists of 350 documents, 50 are kept for development and 100 for testing.

- **Source**: Created for the ScienceIE 2017 workshop.
- **Size**: 500 journal articles (350 training, 50 development, 100 testing).
- **Content**: Articles are balanced across Computer Science, Material Sciences, and Physics. Three document formats are provided: plain text, annotated text, and original full text.
- **Use Case**: Well-suited for training and testing systems on specific scientific domains (Computer Science, Material Sciences, Physics) due to its focused nature and availability of different text formats.

● **Inspec**: Inspec consists of 2,000 abstracts of scientific journal papers from Computer Science collected between the years 1998 and 2002. Each document has two sets of keywords assigned: the controlled keywords, which are manually controlled assigned keywords that appear in the Inspec thesaurus but may not appear in the document, and the uncontrolled keywords which are freely assigned by the editors, i.e., are not restricted to the thesaurus or to the document. In our repository, we consider a union of both sets as the ground-truth

- **Source**: Collection of abstracts from scientific journals in Computer Science (1998-2002).
- **Size**: 2,000 abstracts.
- **Content**: Each abstract has two sets of keywords:
    1. Controlled keywords: Manually assigned from a thesaurus, might not appear in the text.
    2. Uncontrolled keywords: Freely assigned by editors, may not be in the thesaurus or the document itself.
    3. The dataset considers a combination of both sets as ground truth (correct keyphrases).
- **Use Case**: Useful for training systems on identifying keyphrases from scientific abstracts and evaluating their ability to handle both pre-defined terms and terms not explicitly mentioned in the text.

Implementations:

1. Unsupervised Models

Objective: The primary objective of this project is to implement a keyword extraction technique using BERT embeddings, enabling us to identify and extract the most salient keywords and keyphrases from textual documents. We seek to create a robust and efficient method that

surpasses traditional statistical models by leveraging BERT's capability to capture semantic context.

Methodology:

1. Data Selection: We begin by selecting a suitable dataset for experimentation. For demonstration purposes, we chose a document related to supervised machine learning.
2. Candidate Keywords/Keyphrases Generation: Using Scikit-Learn's CountVectorizer, we generate candidate keywords and keyphrases from the chosen document. This step allows for customization of the length and removal of stop words to enhance the quality of candidate phrases.
3. Embeddings: The document and candidate keywords/keyphrases are transformed into numerical vectors using BERT embeddings. Specifically, we utilize DistilBERT due to its proven performance in semantic similarity tasks.
4. Cosine Similarity: To identify the most relevant keywords/keyphrases, we calculate cosine similarity between the document vector and candidate vectors. The candidates with the highest similarity scores are considered potential keywords/keyphrases.
5. Diversification Techniques: We employ two diversification algorithms to balance the accuracy and diversity of extracted keywords/keyphrases:

    a. **Max Sum Similarity:** Selects top candidates based on maximizing their similarity to the document while minimizing similarity between candidates.
    b. **Maximal Marginal Relevance (MMR):** Iteratively selects new candidates that are both similar to the document and dissimilar to already selected keywords/keyphrases, minimizing redundancy.

To Run: when in the `project/unsupervised` dir

Sample input doc

“

Keyphrase extraction is the task of automatically selecting a small set of phrases that
best describe a given free text document. Supervised keyphrase extraction requires large
amounts of labeled training data and generalizes very poorly outside the domain of the
training data. At the same time, unsupervised systems have poor accuracy, and often
do not generalize well, as they require the input document to belong to a larger corpus also
given as input. Addressing these drawbacks,
in this paper, we tackle keyphrase extraction from single documents with EmbedRank:
a novel unsupervised method, that leverages
sentence embeddings. EmbedRank achieves
higher F-scores than graph-based state of the
art systems on standard datasets and is suitable for real-time processing of large amounts
of Web data. With EmbedRank, we also explicitly increase coverage and diversity among
the selected keyphrases by introducing an
embedding-based maximal marginal relevance
(MMR) for new phrases. A user study including over 200 votes showed that, although reducing the phrases’ semantic overlap leads
to
no gains in F-score, our high diversity selection is preferred by humans.

”

Output:


```
{
    "topK": [
        "keyphrase extraction requires",
        "paper tackle keyphrase",
        "documents embedrank novel",
        "supervised keyphrase extraction",
        "embedrank novel unsupervised"
    ],
    "MSS": [
        "extraction requires large",
        "novel unsupervised method",
        "200 votes showed",
        "diversity selection preferred",
        "web data embedrank"
    ],
    "MMR": [
        "embedrank novel unsupervised",
        "supervised keyphrase extraction",
        "200 votes showed",
        "paper tackle keyphrase",
        "extraction task automatically"
    ]
}
```


2. Supervised Models:

a. Neural Methods

BiLSTM with CRF:

- Utilizing deep contextualized embeddings, the method represents input text words, employing a **Bidirectional LSTM-CRF** for sequence labeling across three standard datasets: Inspec, SemEval 2010, and SemEval 2017, employing both standard and pretrained GloVe word embeddings.

- The **Bidirectional LSTM-CRF** architecture, widely favored for sequence labeling, adeptly captures bidirectional dependencies within the text, while the CRF layer imposes global constraints on output sequences, ensuring coherence and consistency in predicted keyphrases.
- Through assessment across the three benchmark datasets, the proposed architecture allows for an impartial **comparison** with **existing methods**. Incorporating both pretrained **GloVe** embeddings and learned embeddings offers insights into the contextual impact on keyphrase extraction.

1. **Bilstm** on **Inspec** dataset using **Naive(Without Glove)** Embeddings

Model Parameters:

```
BiLSTMCRF(
  (sent_vocab): Vocab()
  (tag_vocab): Vocab()
  (embedding): Embedding(3937, 300)
  (dropout): Dropout(p=0.5, inplace=False)
  (encoder): LSTM(300, 300, bidirectional=True)
  (hidden2emit_score): Linear(in_features=600, out_features=5, bias=True)
)
```

Training Report (Taken from .ipynb file):

```
{'I-KEY': 4, 'B-KEY': 3, '<PAD>': 1, 'O': 2, '<UNK>': 0}
              precision    recall  f1-score   support

           2       0.93      0.91      0.92     58168
           3       0.38      0.54      0.44      3418
           4       0.56      0.55      0.55      5706

    accuracy                           0.86     67292
   macro avg       0.62      0.67      0.64     67292
weighted avg       0.87      0.86      0.87     67292
```

2. **Bilstm** on **Inspec** dataset using **Glove** Embeddings

Model Parameters:

```
BiLSTMCRF(
  (sent_vocab): Vocab()
  (tag_vocab): Vocab()
  (embedding): Embedding(3937, 300)
  (dropout): Dropout(p=0.5, inplace=False)
  (encoder): LSTM(300, 300, bidirectional=True)
  (hidden2emit_score): Linear(in_features=600, out_features=5, bias=True)
)
```

Training Report :

```
 {'I-KEY': 4, 'B-KEY': 2, '<PAD>': 1, 'O': 3, '<UNK>': 0}
              precision    recall  f1-score   support

           2       0.29      0.56      0.38      2517
           3       0.95      0.90      0.92     60526
           4       0.43      0.57      0.49      4249

    accuracy                           0.86     67292
   macro avg       0.56      0.67      0.60     67292
weighted avg       0.90      0.86      0.88     67292
```

3. **Bilstm** on **Semeval_2010** dataset using **Naive(Without Glove)** Embeddings .

Model Parameters:

```
BiLSTMCRF(
  (sent_vocab): Vocab()
  (tag_vocab): Vocab()
  (embedding): Embedding(3937, 300)
  (dropout): Dropout(p=0.5, inplace=False)
  (encoder): LSTM(300, 300, bidirectional=True)
  (hidden2emit_score): Linear(in_features=600, out_features=5, bias=True)
)
```

Training Report (Taken from .ipynb file):

	precision	recall	f1-score	support
2	0.16	0.34	0.22	476
3	0.16	0.38	0.22	301
4	0.98	0.93	0.95	19892
accuracy			0.91	20669
macro avg	0.43	0.55	0.47	20669
weighted avg	0.95	0.91	0.93	20669

4. **Bilstm** on **Semeval_2010** dataset using **Glove** Embeddings.

Model Parameters:

```
BiLSTMCRF(
  (sent_vocab): Vocab()
  (tag_vocab): Vocab()
  (embedding): Embedding(3937, 300)
  (dropout): Dropout(p=0.5, inplace=False)
  (encoder): LSTM(300, 300, bidirectional=True)
  (hidden2emit_score): Linear(in_features=600, out_features=5, bias=True)
)
```

Training Report (Taken from .ipynb file):

	precision	recall	f1-score	support
2	0.99	0.92	0.96	20420
3	0.06	0.39	0.10	151
4	0.07	0.54	0.13	98
accuracy			0.92	20669
macro avg	0.38	0.62	0.40	20669
weighted avg	0.98	0.92	0.95	20669

5. **Bilstm** on **Semeval_2017** dataset using **Naive(Without Glove)** Embeddings.

Model Parameters:

```
BiLSTMCRF(
  (sent_vocab): Vocab()
  (tag_vocab): Vocab()
  (embedding): Embedding(3937, 300)
  (dropout): Dropout(p=0.5, inplace=False)
  (encoder): LSTM(300, 300, bidirectional=True)
  (hidden2emit_score): Linear(in_features=600, out_features=5, bias=True)
)
```

Training Report (Taken from .ipynb file):

	precision	recall	f1-score	support
2	0.31	0.23	0.26	1670
3	0.14	0.22	0.18	532
4	0.01	0.62	0.01	8
5	0.95	0.84	0.89	19790
6	0.00	0.00	0.00	0
7	0.00	0.00	0.00	0
8	0.01	0.26	0.02	31
accuracy			0.78	22031
macro avg	0.20	0.31	0.19	22031
weighted avg	0.88	0.78	0.82	22031

6. Bilstm on **Semeval_2017** dataset using **Glove** Embeddings.

Model Parameters.

```
BiLSTMCRF(
  (sent_vocab): Vocab()
  (tag_vocab): Vocab()
  (embedding): Embedding(3937, 300)
  (dropout): Dropout(p=0.5, inplace=False)
  (encoder): LSTM(300, 300, bidirectional=True)
  (hidden2emit_score): Linear(in_features=600, out_features=5, bias=True)
)
```

Training Report (Taken from .ipynb file):

	precision	recall	f1-score	support
2	0.24	0.23	0.24	1285
3	0.00	0.00	0.00	0
4	0.96	0.83	0.89	20324
5	0.00	0.33	0.01	6
6	0.01	0.11	0.01	37
7	0.00	0.00	0.00	0
8	0.09	0.19	0.12	379
accuracy			0.78	22031
macro avg	0.19	0.24	0.18	22031
weighted avg	0.90	0.78	0.84	22031

7. **Bilstm** with **Self Attention** - This model uses a Multi Headed Attention Layer with positional encoding.

Model Parameters:

```
attbilstm(
  (positionalencoding): PositionalEmbedding()
  (embedding): Embedding(13832, 256)
  (attention): MultiHeadAttention(
    (query_matrix): Linear(in_features=64, out_features=64, bias=False)
    (key_matrix): Linear(in_features=64, out_features=64, bias=False)
    (value_matrix): Linear(in_features=64, out_features=64, bias=False)
    (out): Linear(in_features=512, out_features=512, bias=True)
  )
  (encoder): LSTM(256, 256, num_layers=2, dropout=0.5, bidirectional=True)
  (fc): Linear(in_features=512, out_features=3, bias=True)
  (dropout): Dropout(p=0.5, inplace=False)
)
```

Training Classification Report:

```
100%|██████████| 1000/1000 [01:04<00:00, 15.61it/s]
              precision    recall  f1-score   support

           0       0.87      0.98      0.92    123449
           1       0.10      0.03      0.05      9633
           2       0.13      0.00      0.01      8426

    accuracy                           0.86    141508
   macro avg       0.37      0.34      0.33    141508
weighted avg       0.78      0.86      0.81    141508
```

Test Classification Report:

```
              precision    recall  f1-score   support

           0       0.87      1.00      0.93    123449
           1       0.00      0.00      0.00      9633
           2       0.00      0.00      0.00      8426

    accuracy                           0.87    141508
   macro avg       0.29      0.33      0.31    141508
weighted avg       0.76      0.87      0.81    141508
```

b. Pretrained Models:

1. **Keybert with keyphrase-vectorizers**: KeyphraseVectorizers package together with KeyBERT. This will significantly enhance the efficiency of keyword extraction from textual data. The KeyphraseVectorizers package employs sophisticated part-of-speech patterns to meticulously identify keyphrases within a corpus of text documents. These key phrases, representing meaningful syntactic structures such as noun or verb phrases, are then organized into a document-keyphrase matrix. This matrix serves as a comprehensive representation of the frequency of keyphrase occurrences across the entire document collection, providing valuable insights into the significance of each keyphrase within individual documents and the corpus as a whole. Leveraging this matrix, KeyBERT utilizes advanced similarity metrics, likely including cosine similarity, to assess the relevance of each key phrase to the overarching content. By ranking the keyphrases based on their relevance scores, KeyBERT extracts the most salient keywords or keyphrases, offering a succinct summary of the primary topics and themes

encapsulated within the text. This seamless integration streamlines the keyword extraction process, facilitating tasks such as document summarization, information retrieval, and content analysis with heightened precision and efficiency.

Model Class:

```
class KeyphraseGenerationPipeline(Text2TextGenerationPipeline):
    def __init__(self, model, keyphrase_sep_token=";", *args, **kwargs):
        super().__init__(
            model=AutoModelForSeq2SeqLM.from_pretrained(model),
            tokenizer=AutoTokenizer.from_pretrained(model),
            *args,
            **kwargs
        )
        self.keyphrase_sep_token = keyphrase_sep_token

    def postprocess(self, model_outputs):
        results = super().postprocess(
            model_outputs=model_outputs
        )
        return [[keyphrase.strip() for keyphrase in result.get("generated_text").split(self.keyphrase_sep_token) if keyphrase != ""] for result in results]

model_name = "ml6team/keyphrase-generation-keybart-inspec"
generator = KeyphraseGenerationPipeline(model=model_name)
keyphrases = generator(test_document)
```

Validation Classification Report:

	precision	recall	f1-score	support
B	0.38	0.00	0.00	3950
I	0.00	0.00	0.00	4522
O	0.87	1.00	0.93	57819
accuracy			0.87	66291
macro avg	0.42	0.33	0.31	66291
weighted avg	0.78	0.87	0.81	66291

Test Classification Report:

	precision	recall	f1-score	support
B	0.56	0.38	0.46	4207
I	0.63	0.41	0.50	4875
O	0.91	0.96	0.94	58218
accuracy			0.88	67300
macro avg	0.70	0.59	0.63	67300
weighted avg	0.87	0.88	0.87	67300

2. **Keyphrase-extraction-kbir**: This model uses KBIR as its base model and fine-tunes it on the Inspec dataset. KBIR or Keyphrase Boundary Infilling with Replacement is a pre-trained model that utilizes a multi-task learning setup for optimizing a combined loss of Masked Language Modeling (MLM), Keyphrase Boundary Infilling (KBI) and Keyphrase Replacement Classification (KRC).

Model Class:

```
class KeyphraseExtractionPipeline(TokenClassificationPipeline):
    def __init__(self, model, *args, **kwargs):
        super().__init__(
            model=AutoModelForTokenClassification.from_pretrained(model),
            tokenizer=AutoTokenizer.from_pretrained(model),
            *args,
            **kwargs
        )

    def postprocess(self, model_outputs):
        results = super().postprocess(
            model_outputs,
            aggregation_strategy=AggregationStrategy.SIMPLE,
        )
        return np.unique([result.get("word").strip() for result in results])

model_name = "ml6team/keyphrase-extraction-kbir-inspec"
extractor = KeyphraseExtractionPipeline(model=model_name)

keyphrases = extractor(test_document)
```

Validation Classification Report:

	precision	recall	f1-score	support
B	0.27	0.00	0.00	3950
I	0.00	0.00	0.00	4522
O	0.87	1.00	0.93	57819
accuracy			0.87	66291
macro avg	0.38	0.33	0.31	66291
weighted avg	0.78	0.87	0.81	66291

Test Classification Report:

	precision	recall	f1-score	support
B	0.63	0.65	0.64	4207
I	0.70	0.71	0.70	4875
O	0.95	0.95	0.95	58218
accuracy			0.91	67300
macro avg	0.76	0.77	0.76	67300
weighted avg	0.91	0.91	0.91	67300

3. **Keyphrase-generation-t5-small**: This model uses the T5-small model as its base model and finetunes it on the Inspec dataset. The T5-small model architecture inherits the capabilities of T5 (Text-To-Text Transfer Transformer), a renowned transformer-based model celebrated for its versatility in various text generation tasks. The distinctive feature of this model lies in its fine-tuning process, wherein the T5-small architecture is further trained on the Inspec dataset, a comprehensive collection of research articles predominantly from the fields of computer science and information technology, augmented with meticulously annotated keyphrases. By fine-tuning on this dataset, the model acquires an understanding of the nuanced relationships between text and corresponding key phrases. Keyphrase generation is framed as a text-to-text generation problem, where the model is tasked with generating key phrases that succinctly capture the essence of the input text.

Model Class:

```
class KeyphraseGenerationPipeline(Text2TextGenerationPipeline):
    def __init__(self, model, keyphrase_sep_token=";", *args, **kwargs):
        super().__init__(
            model=AutoModelForSeq2SeqLM.from_pretrained(model),
            tokenizer=AutoTokenizer.from_pretrained(model),
            *args,
            **kwargs
        )
        self.keyphrase_sep_token = keyphrase_sep_token

    def postprocess(self, model_outputs):
        results = super().postprocess(
            model_outputs=model_outputs
        )
        return [[keyphrase.strip() for keyphrase in result.get("generated_text").split(self.keyphrase_sep_token) if keyphrase != ""] for result in results]


# Load pipeline
model_name = "ml6team/keyphrase-generation-t5-small-inspec"
generator = KeyphraseGenerationPipeline(model=model_name)

keyphrases = generator(test_document)

print(keyphrases)
```

Validation Classification Report:

	precision	recall	f1-score	support
B	0.07	0.02	0.04	3950
I	1.00	0.00	0.00	4522
O	0.87	0.98	0.92	57819
accuracy			0.86	66291
macro avg	0.65	0.33	0.32	66291
weighted avg	0.83	0.86	0.81	66291

Test Classification Report:

	precision	recall	f1-score	support
B	0.40	0.34	0.37	4207
I	0.62	0.36	0.46	4875
O	0.91	0.95	0.93	58218
accuracy			0.87	67300
macro avg	0.64	0.55	0.58	67300
weighted avg	0.85	0.87	0.86	67300

4. **Keyphrase-extraction-distilbert**: This model is designed for the specific task of keyphrase extraction, which involves identifying and extracting the most significant phrases or keywords from a given text to encapsulate its essence. At its core, this model leverages DistilBERT, a variant of the powerful BERT (Bidirectional Encoder Representations from Transformers) model, known for its prowess in understanding the nuances of natural language. In the fine-tuning process, we further trained the DistilBERT architecture on the Inspec dataset. The Inspec dataset, a widely recognized benchmark in the domain, comprises research articles primarily from computer science and information technology, each meticulously annotated with keyphrases that encapsulate the main themes and topics of the respective articles. By fine-tuning DistilBERT on this rich and diverse dataset, the "Keyphrase-extraction-distilbert" model becomes adept at comprehending the context of research articles and discerning the key phrases that best summarize their content. This enables researchers and practitioners to efficiently navigate through vast volumes of academic literature, swiftly extracting key insights and pertinent information crucial for their endeavors.

Model Class:

```python
class KeyphraseExtractionPipeline(TokenClassificationPipeline):
    def __init__(self, model, *args, **kwargs):
        super().__init__(
            model=AutoModelForTokenClassification.from_pretrained(model),
            tokenizer=AutoTokenizer.from_pretrained(model),
            *args,
            **kwargs
        )

    def postprocess(self, model_outputs):
        results = super().postprocess(
            model_outputs,
            aggregation_strategy=AggregationStrategy.FIRST,
        )
        return np.unique([result.get("word").strip() for result in results])


# Load pipeline
model_name = "ml6team/keyphrase-extraction-distilbert-inspec"
extractor = KeyphraseExtractionPipeline(model=model_name)

keyphrases = extractor(test_document)
```

Validation Classification Report:

	precision	recall	f1-score	support
B	0.10	0.00	0.00	3950
I	1.00	0.00	0.00	4522
O	0.87	1.00	0.93	57819
accuracy			0.87	66291
macro avg	0.66	0.33	0.31	66291
weighted avg	0.84	0.87	0.81	66291

Test Classification Report:

	precision	recall	f1-score	support
B	0.51	0.68	0.58	4207
I	0.62	0.70	0.66	4875
O	0.96	0.92	0.94	58218
accuracy			0.89	67300
macro avg	0.70	0.77	0.73	67300
weighted avg	0.90	0.89	0.90	67300

Error Analysis:

1. Due to the overabundance of outside the keyphrase words. Words with O label. The LSTM models overfit on it and produce the same output for almost all inputs.

2. Challenges in changing the tokens according to the words when using BERT encodings led to the failure of the BiLSTM CRF with contextualized embeddings.

Materials Referred:

1. Keyphrase Extraction from Scholarly Articles as Sequence Labeling using Contextualized Embeddings (https://arxiv.org/pdf/1910.08840.pdf)
2. Keyphrase Extraction with BERT Transformers and Noun Phrases (https://towardsdatascience.com/enhancing-keybert-keyword-extraction-resultswith-keyphrasevectorizers-3796fa93f4db)
3. Learning Rich Representation of Keyphrases from Text (https://arxiv.org/pdf/2112.08547.pdf)
4. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer (https://jmlr.org/papers/volume21/20-074/20-074.pdf)
5. Automatic Keyphrase Extraction: A Survey of the State of the Art (https://aclanthology.org/P14-1119.pdf)