Лабораторна робота 4

Реранкінг та Cross-Encoder у RAG

Мета роботи: дослідити вплив двоетапного пошуку (bi-encoder + cross-encoder reranking) на якість релевантності відповіді.

Науково-теоретичне обґрунтування

У класичному RAG використовується bi-encoder (окреме кодування запиту і документу).

Cross-encoder оцінює пару (query, document) спільно, що дозволяє значно підвищити точність ранжування.

Стек технологій

Sentence-Transformers (bi-encoder)

CrossEncoder (наприклад ms-marco-MiniLM)

ChromaDB

LangChain

NumPy / Pandas

Корисні посилання

Sentence-Transformers документація: https://www.sbert.net/

ChromaDB: https://docs.trychroma.com/

MS MARCO датасет: https://microsoft.github.io/msmarco/

BEIR benchmark: https://github.com/beir-cellar/beir

LangChain retrieval docs: https://python.langchain.com/docs/modules/data_connection/

Зміст роботи

Завдання 1. *Реалізувати стандартний Top-10 retrieval через bi-encoder.*

Кроки виконання

- Завантажити датасет (використати один з наборів Natural Questions, BEIR, HotpotQA, MS MARCO).
- Створити embeddings для поля description.
- Ініціалізувати ChromaDB.
- Реалізувати пошук Top-10 для 10 тестових запитів.
- Зберегти результати у таблицю:

Query	Rank	Document ID	Cosine Score	Relevant (0/1)
What is RAG?	1	doc_042	0.912	1
What is RAG?	2	doc_017	0.887	1

What is RAG?	3	doc_103	0.831	0
...	...	...	...	...

– Обчислити середню косинусну схожість релевантних результатів.

– Проаналізувати випадки хибного ранжування.

Методичні рекомендації

Встановлюємо потрібні бібліотеки: `sentence-transformers chromadb pandas numpy, time.`

Проведемо імітацію документів MS MARCO, підготовимо умовно 20 документів.

```
# У реальній роботі тут буде завантаження через datasets
data = {
    'id': ['doc1', 'doc2', 'doc3', 'doc4', 'doc5'],
    'text': [
        "Python - високорівнева мова програмування для загального вжитку.",
        "Машинне навчання - це підмножина штучного інтелекту.",
        "Трансформери змінили підхід до обробки природної мови.",
        "ChromaDB - це векторна база даних для зберігання ембедінгів.",
        "Метод k-найближчих сусідів використовується для класифікації."
    ]
}
df = pd.DataFrame(data)
```

Використати модель типу all-MiniLM-L6-v2.

```
model_name = 'all-MiniLM-L6-v2'
bi_encoder = SentenceTransformer(model_name)
```

Налаштування ChromaDB

```
client = chromadb.Client()
# Створюємо колекцію.
# Вказуємо функцію ембедінгу, щоб база сама кодувала запити
collection = client.create_collection(name="ms_marco_mini")
```

Індексація: перетворення текстів у вектори та збереження

```
print("Індексація документів...")
collection.add(
    documents=df['text'].tolist(),
    ids=df['id'].tolist(),
    embeddings=bi_encoder.encode(df['text'].tolist()).tolist()
)
```

Тестові запити для оцінки

```
test_queries = [
```

```
    "Що таке штучний інтелект?",
    "Яка мова програмування популярна для ML?",
    "Як зберігати вектори?"
]
```

Виконання пошуку Top-10 (у прикладі Top-3 через малу базу)

```
results_table = []

for query in test_queries:
    query_embedding = bi_encoder.encode(query).tolist()

    # Пошук у ChromaDB
    results = collection.query(
        query_embeddings=[query_embedding],
        n_results=3
    )
```

Формування звіту

```
    for i in range(len(results['ids'][0])):
        results_table.append({
            'Query': query,
            'Rank': i + 1,
            'Document ID': results['ids'][0][i],
            'Cosine Score': round(1 - results['distances'][0][i], 4),

            'Text': results['documents'][0][i]
        })
```

Cosine Score: Це міра релевантності.

- – - ідеальний збіг.
- – - жодного зв'язку.

У ChromaDB за замовчуванням використовується $L2$ відстань, тому для отримання Cosine Score ми робимо трансформацію (залежно від налаштувань відстані в БД).

```
output_df = pd.DataFrame(results_table)
print("\nРезультати пошуку Bi-Encoder:")
print(output_df.to_string(index=False))
```

!!! Зверніть увагу, чи знаходить Bi-Encoder документи, де немає спільних слів із запитом (наприклад, запит "ML", а документ "Машинне навчання"). Це покаже силу семантичного пошуку.

Заміряйте час виконання collection.query за допомогою time.time().

Завдання 2. *Застосувати cross-encoder для reranking Top-10 → Top-3.*

Застосувати cross-encoder для переоцінки Top-10 результатів bi-encoder та відібрати найрелевантніший Top-3. Порівняти якість ранжування до і після реранкінгу.

Кроки виконання:

– Встановити та ініціалізувати модель cross-encoder (наприклад, ms-marco-MiniLM-L-6-v2)

– Для кожного запиту взяти Top-10 кандидатів від bi-encoder

– Передати пари (query, document) до cross-encoder для оцінки

– Відсортувати за cross-encoder score та відібрати Top-3

– Зберегти нові ранги, скори та мітки релевантності

– Порівняти зміни в ранжуванні: які документи піднялися, які опустилися

Методичні рекомендації

Cross-Encoder отримує на вхід пару (Query, Document) одночасно. Він аналізує взаємозв'язки між кожним словом запиту та кожним словом документа.

Чому Top-10 → Top-3? НЕ можна прогнати через Cross-Encoder тисячі документів (це занадто повільно), тому беремо лише 10 найкращих кандидатів від Bi-Encoder і "переставляємо" їх у правильному порядку.

Використаємо модель *cross-encoder/ms-marco-MiniLM-L-6-v2*, яка спеціально навчена для ранжування в задачах MS MARCO.

1. Передавати пари у форматі списку кортежів: [(query, doc1), (query, doc2), ...].

2. Cross-encoder повертає raw logits, не ймовірності — вищий скор означає вищу релевантність.

3. Зберігати обидва скори (bi-encoder cosine та cross-encoder score) для подальшого порівняння.

4. Виміряти latency реранкінгу окремо від bi-encoder retrieval.

5. Проаналізувати приклади, де cross-encoder змінив порядок: визначити причини (лексичні збіги, семантичні нюанси тощо).

Приклад коду

Завдання 3. Обчислити таки метрики якості, як Precision@k, MRR (Mean Reciprocal Rank) та nDCG для порівняння якості bi-encoder і двоетапного пошуку.

Методичні рекомендації

Оскільки для прикладу було взято скорочений наборі (5 документів), проведемо розрахунок метрик для 3 тестових запитів. Це дозволить наочно побачити різницю в якості між звичайним пошуком та реранкінгом.

Для коректного обчислення спочатку визначимо еталонні відповіді, які документи є справді релевантними для конкретних запитів.

```
from sklearn.metrics import ndcg_score

# Формат: {запит: [список релевантних doc_id]}
gold_labels = {
    "Що таке штучний інтелект?": ["doc2"],
    "Яка мова програмування популярна для ML?": ["doc1"],
    "Як зберігати вектори?": ["doc4"]
}

# результати після Завдання 1 (Bi-Encoder)
# та Завдання 2 (Cross-Encoder) у вигляді списків ID
results_bi = {
    "Що таке штучний інтелект?": ["doc2", "doc5", "doc3"],
    "Яка мова програмування популярна для ML?": ["doc4", "doc1", "doc2"],
    "Як зберігати вектори?": ["doc4", "doc3", "doc1"]
}

results_cross = {
    "Що таке штучний інтелект?": ["doc2", "doc3", "doc5"],
    "Яка мова програмування популярна для ML?": ["doc1", "doc2", "doc4"],
    "Як зберігати вектори?": ["doc4", "doc1", "doc3"]
}
```

Необхідно обчислити метрику Precision@k для кожного запиту окремо, після чого визначити її середнє значення по всій множині запитів.

Метрика MRR (Mean Reciprocal Rank) враховує лише позицію першого знайденого релевантного документа у списку результатів пошуку, що дозволяє оцінити здатність системи повертати найбільш релевантний результат на верхніх позиціях ранжування.

Метрика nDCG (normalized Discounted Cumulative Gain) дає змогу враховувати різні рівні релевантності документів (наприклад, 0, 1, 2). У межах даної роботи використовується спрощена бінарна шкала релевантності, де значення можуть дорівнювати лише 0 або 1.

Для обчислення IDCG (Ideal Discounted Cumulative Gain) необхідно відсортувати еталонні значення релевантності у порядку ідеального ранжування та на їх основі розрахувати значення DCG.

```
        # --- Precision@k ---
        hits = len(set(retrieved_ids[:k]) & set(relevant_ids))
        precisions.append(hits / k)

        # --- MRR (Mean Reciprocal Rank) ---
        score_mrr = 0
        for i, doc_id in enumerate(retrieved_ids[:k]):
            if doc_id in relevant_ids:
                score_mrr = 1 / (i + 1)
                break
        mrr_scores.append(score_mrr)

        # --- nDCG (Normalized Discounted Cumulative Gain) ---
        # Створюємо вектори релевантності: 1 якщо док релевантний, 0 якщо ні
        true_relevance = [[1 if doc_id in relevant_ids else 0 for doc_id in
retrieved_ids[:k]]]
        # Ідеальний порядок — це коли релевантний док на 1 місці
        scores_for_ndcg = [[len(retrieved_ids) - i for i in
range(len(retrieved_ids[:k]))]]

        ndcg = ndcg_score(true_relevance, scores_for_ndcg)
        ndcg_scores.append(ndcg)
```

Обчислення і формування порівняльної таблиці

```
metrics_bi = get_metrics(results_bi, gold_labels)
metrics_cross = get_metrics(results_cross, gold_labels)

metrics_df = pd.DataFrame([metrics_bi, metrics_cross], index=['Bi-Encoder',
'Bi+Cross-Encoder'])
```

```
print(metrics_df.round(3))
```

Використовуйте scikit-learn або реалізуйте власні функції - обидва підходи прийнятні.

Завдання 4. *Порівняння методів та візуалізація.* Побудувати порівняльний аналіз bi-encoder та двоетапного пошуку (bi + cross-encoder), візуалізувати приріст якості у вигляді графіків.

Методичні рекомендації

1. Побудувати барчарт для порівняння Precision@k при k = 1, 3, 5, 10.
2. Побудувати барчарт для порівняння nDCG@k при k = 1, 3, 5, 10.
3. Відобразити MRR як горизонтальну лінію або окремий графік.
4. Побудувати scatter plot: Bi-encoder score vs Cross-encoder score для кожного документу.
5. Побудувати box plot для розподілу косинусних скорів релевантних і нерелевантних документів.
6. Обчислити та відобразити відсоток покращення: (CE - BI) / BI * 100%.
7. Додати latency comparison: стовпчикова діаграма час відповіді bi-encoder vs bi+cross-encoder.
8. Описати висновки: за яких умов двоетапний пошук дає найбільший приріст?

Контрольні запитання

1. Поясніть архітектурну різницю між bi-encoder та cross-encoder.
2. Чому cross-encoder не використовується для повного індексування?
3. Що таке reciprocal rank?
4. У чому різниця між Precision@k та Recall@k?
5. Чому nDCG вважається більш інформативною метрикою?
6. Які переваги та недоліки двоетапного пошуку?
7. Як latency впливає на UX системи?

8. Чому cross-encoder краще працює для складних запитів?
9. У чому полягає проблема false positive у retrieval?
10. Як статистично довести, що reranking покращив якість?