Universitatea Babeș-Bolyai
Facultatea de Matematică și Informatică
Data Mining

Team project report

IBM’s Watson Question Answering System

Sisteme distribuite în internet

Grupa 244

Diaconu Horia
Filipaș Răzvan
Moldovan Bogdana

Table of contents

1 Cerinta problemei

IBM's Watson is a **Question Answering (QA) system** that "can compete at the human champion level in real time on the TV quiz show, **Jeopardy**." This, as we will see in class, is a complex undertaking. However, the answers to many of the Jeopardy questions are actually titles of Wikipedia pages. For example, the answer to the clue "This woman who won consecutive heptathlons at the Olympics went to UCLA on a basketball scholarship" is "Jackie Joyner-Kersee", who has a Wikipedia page with the same title: http://en. wikipedia.org/wiki/Jackie_Joyner-Kersee. In these situations, the task reduces to the classification of Wikipedia pages, that is, finding which page is the most likely answer to the given clue. This is the focus of this project.

In this project you will use the following data (see D2L project folder):

• 100 questions from previous Jeopardy games, whose answers appear as Wikipedia pages. The questions are listed in a single file, with 4 lines per question, in the following format: **CATEGORY CLUE ANSWER NEWLINE**. For example:

NEWSPAPERS

The dominant paper in our nation's capital, it's among the top 10 U.S. papers in circulation

The Washington Post

• A collection of approximately 280,000 Wikipedia pages, which include the correct answers for the above 100 questions. The pages are stored in 80 files (thus each file contains several thousand pages). Each page starts with its title, encased in double square brackets. For example, BBC's page starts with "[[BBC]]".

Your project should address the following points:

1) **(25% of project grade) Indexing and retrieval**: **Index** the Wikipedia collection with a state of the art Information Retrieval (IR) system such as **Lucene** (http:// lucene.apache.org/) or **Whoosh** (https://whoosh.readthedocs.io/en/latest/ intro.html). Make sure that each Wikipedia page appears as a separate document in the index (rather than creating a document from each of the 80 files). Describe how you prepared the terms for indexing (stemming, lemmatization, stop words, etc.). What issues specific to Wikipedia content did you discover, and how did you address them? Implement the retrieval component, which takes as query the Jeopardy clue (and, optionally, the question category) and returns the title of the Wikipedia page that is most similar. Describe how you built the query from the clue. For example, are you using all the words in the clue or a subset? If the latter, what is the best algorithm for selecting the subset of words from the clue? Are you using the category of the question?
2) **(25% of project grade) Measuring performance**: Measure the performance of your Jeopardy system, using at least one of the metrics discussed in class, e.g., precision at

1 (P@1), normalized discounted cumulative gain (NDCG), or mean reciprocal rank (MRR). Note: not all the above metrics are relevant here! Justify your choice, 2 and then report performance using the metric(s) of your choice.

3) **(25% of project grade) Error analysis**: Perform an error analysis of your best system. How many questions were answered correctly/incorrectly? Why do you think the correct questions can be answered by such a simple system? What problems do you observe for the questions answered incorrectly? Aim to group the errors into a few classes and discuss them.
4) **(25% of project grade) Improving retrieval** (GRAD STUDENTS ONLY): Improve the above standard IR system using natural language processing and/or machine learning. For this task you have more freedom in choosing a solution and I encourage you to use your imagination. For example, you could implement a positional index instead of a bag of words. Or, you could use ChatGPT to rerank the top K pages produced by your information retrieval system. What is the performance of your system after this improvement?

2 Importanta proiectului

Proiectul Watson reprezintă o oportunitate valoroasă pentru aplicarea practică a conceptelor de prelucrare a limbajului natural (NLP). Acesta implică utilizarea instrumentelor și tehnicilor NLP într-un context real, aplicând cunoștințele teoretice acumulate.

Pentru studenții de master, proiectul deschide posibilitatea explorării soluțiilor avansate, cum ar fi utilizarea prelucrării limbajului natural (NLP) sau a învățării automate pentru a aduce îmbunătățiri semnificative sistemului. Abilitățile dobândite prin acest proiect sunt relevante în domenii precum căutare și recuperare de informații, asistență virtuală și dezvoltare de sisteme de tip chatbot, contribuind astfel la dobândirea unei competențe valoroase în societatea modernă.

3 Tehnologii folosite

3.1 Python

Python este un limbaj de programare de nivel înalt, interpretat și generalist, cunoscut pentru sintaxa sa clară și concisă. A fost creat în 1991 de către Guido van Rossum și este un limbaj versatil, utilizat într-o varietate de domenii, inclusiv dezvoltarea web, analiza datelor, inteligența artificială și prelucrarea limbajului natural (NLP). Python oferă o gamă largă de biblioteci și framework-uri, făcându-l potrivit pentru proiecte de diverse dimensiuni și complexități.

În contextul proiectului, Python a fost folosit pentru implementarea funcționalităților legate de procesarea limbajului natural (NLP), manipularea datelor și gestionarea interacțiunilor cu sistemul Whoosh pentru indexarea și recuperarea informațiilor din colecția de pagini Wikipedia.

3.2 Whoosh

Whoosh este o bibliotecă Python pentru indexarea și căutarea de text. Este o soluție simplă și eficientă pentru implementarea unui sistem de căutare într-un set mare de documente textuale. Whoosh suportă funcționalități precum indexarea inversă, stemming, și căutare avansată. Este dezvoltată pentru a fi ușor de folosit și extinsă, făcând-o potrivită pentru proiecte în care manipularea și recuperarea informațiilor din texte sunt esențiale.

În proiectul menționat, Whoosh a fost utilizat pentru a realiza indexarea și recuperarea eficientă a informațiilor din cele 80 de fișiere Wikipedia. Acest lucru a inclus implementarea unui sistem de căutare care să gestioneze eficient interogările bazate pe întrebările Jeopardy și să returneze paginile Wikipedia cele mai relevante. Funcționalitățile oferite de Whoosh, împreună cu capacitățile de prelucrare a limbajului natural din Python, au contribuit la implementarea cu succes a componentei de indexare și recuperare a informațiilor în cadrul proiectului.

4 Descrierea codului

4.1 Functia de indexare a fisierelor

Funcția ***index_wikipedia_data*** este responsabilă pentru indexarea datelor din colecția de pagini Wikipedia. Această funcție primește doi parametri:

- **index** - reprezintă obiectul indexului creat cu ajutorul funcției `**create_index**`
- **data_folder** - indică către directorul care conține fișierele Wikipedia ce urmează a fi procesate și indexate.

Procesul de indexare se desfășoară astfel:

- un obiect `writer` este creat pentru a permite adăugarea de documente în index.
- se iterează prin fișierele din directorul specificat (`data_folder`).
- fiecare fișier este deschis și conținutul său este citit. Acest conținut este ulterior divizat în **articole**, folosind markerul `[[`.
- pentru fiecare articol, se identifică **titlul** și **conținutul**. Titlul este obținut până la prima apariție a markerului `]]`, iar conținutul este tot ceea ce urmează după titlu.

- se adaugă documentul în index, unde titlul devine un câmp numit "title", iar conținutul devine un câmp numit "content".
- procesul se repetă pentru toate fișierele din directorul specificat.

La finalul procesului, toate modificările adăugate în index sunt confirmate prin apelul metodei `writer.commit()`.

```
def index_wikipedia_data(index, data_folder):
    writer = index.writer()
    k = 0
    for file_name in os.listdir(data_folder):
        k = k + 1
        print(f"Processing file: {file_name}")
        print(k)
        with open(os.path.join(data_folder, file_name), 'r', encoding='latin-1') as file:
            content = file.read()
            articles = content.split('[[')[1:]
            for article in articles:
                title_end = article.find(']]')
                title = article[:title_end].strip()
                text = article[title_end + 2:].strip()
                writer.add_document(title=title, content=text)

    writer.commit()
```

Funcția ***create_index*** este apelată înainte ca indexul sa fie utilizat in "index_wikipedia_data" și este responsabilă pentru crearea unui nou index cu o anumita schema și un director specific. Dacă directorul indicat nu există, acesta este creat. Apoi, un index nou este creat și returnat pentru a fi utilizat în funcția "index_wikipedia_data".

```
def create_index(schema, index_dir):
    if not os.path.exists(index_dir):
        os.mkdir(index_dir)
    index = create_in(index_dir, schema)
    return index

```

4.2 Functia de cautare in functie de cuvinte cheie si/sau categorie

Funcția ***retrieve_best_page*** este concepută pentru a interoga un index și a recupera cea mai relevantă pagină Wikipedia pentru o anumită întrebare (query). Funcția primește trei parametri:

- index - indexul creat anterior

- query - întrebarea pentru care se caută răspunsul
- category - categoria întrebării, optional.

Procesul funcției este următorul:

- se creează un obiect **searcher** pentru index, utilizând o schemă a căutării bazată pe modelul TF-IDF (Term Frequency-Inverse Document Frequency).
- un obiect **parser** de interogare este creat pentru a prelucra întrebările, având ca bază câmpul "content" din schema indexului.
- in cazul în care este specificată o categorie (`category`), aceasta este adăugată la începutul interogării.
- se parsează interogarea și se obțin rezultatele căutării, limitate la un singur rezultat (`limit=1`).
- dacă există rezultate, se returnează titlul primei pagini (`results[0]['title']`), altfel se returnează `None`.

Această funcție este apelată într-o secțiune specifică pentru a testa sistemul cu un set de întrebări predefinite. În această secțiune, indexul este deschis, iar setul de întrebări este citit din fișierul "questions.txt". Fiecare întrebare este apoi procesată utilizând “retrieve_best_page”, iar rezultatele sunt afișate pentru a fi evaluate.

De asemenea, funcția calculează și afișează numărul total de răspunsuri corecte, în raport cu răspunsurile așteptate. Această secțiune poate fi considerată un scenariu de testare și evaluare a performanțelor sistemului de întrebări și răspunsuri bazat pe indexul Wikipedia.

```
5   def retrieve_best_page(index, query, category=None):
6       searcher = index.searcher(weighting=scoring.TF_IDF())
7       parser = QueryParser("content", schema=index.schema)
8
9       if category:
10          query = f"{category} {query}"
11
12      q = parser.parse(query)
13      results = searcher.search(q, limit=1)
14
15      if results:
16          return results[0]['title']
17      else:
18          return None
19
```

5 Rezultate. Anliza erorilor

5.1 Evaluarea performantei

În procesul de evaluare a performanțelor sistemului nostru, am ales să ne focalizăm pe metrica **Precision at 1 (P@1)** pentru a obține o evaluare detaliată a eficacității căutării. P@1 este o măsură specifică, concentrată asupra acurateții primului rezultat returnat de către sistem în răspuns la o interogare. Alegerea acestei metrici a fost strategică în contextul proiectului nostru, unde relevanța primului rezultat este deosebit de importantă pentru utilizatori.

Această perspectivă detaliată asupra eficacității căutării ne-a permis să identificăm și să aducem îmbunătățiri, în vederea maximizării acurateței și relevanței primului rezultat furnizat de către sistem. În esență, folosirea metricii P@1 a oferit o transparență asupra performanței noastre într-un mod specific la nevoile proiectului nostru, contribuind astfel la dezvoltarea continuă a sistemului pentru a îndeplini cerințele utilizatorilor.

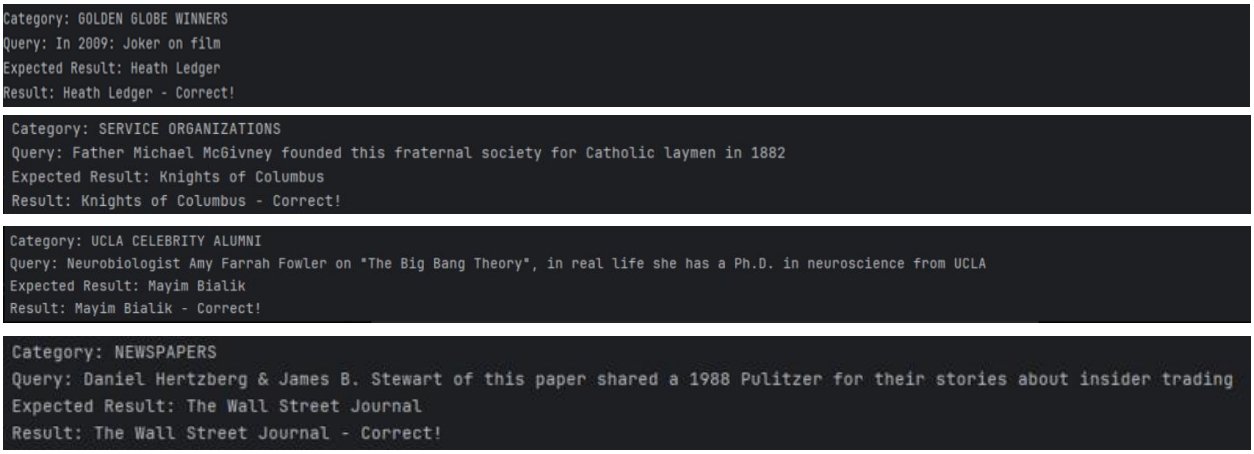
5.2 How many questions were answered correctly/incorrectly?

În procesul nostru de analiză, am constatat că, pe setul de date furnizat de problema inițială, am obținut o precizie concretă de doar **4%**, semnalând astfel o performanță suboptimală în identificarea răspunsurilor corecte. În plus, am observat o tendință de furnizare a unor răspunsuri din categoria corectă, chiar dacă acestea nu corespundeau întotdeauna cu răspunsurile corecte.

Cu toate acestea, prin dezvoltarea și implementarea propriului set de date, am obținut îmbunătățiri semnificative. Astfel, pe noul nostru set de date, am reușit să răspundem corect la 14 din 20 de întrebări, atingând astfel o precizie notabilă de **70%**. Această adaptabilitate și capacitate de îmbunătățire evidențiază importanța dezvoltării și evaluării sistemelor pe seturi de date personalizate pentru a asigura o performanță optimă în contextul specific al problemei abordate.

5.3 Why do you think the correct questions can be answered by such a simple system?

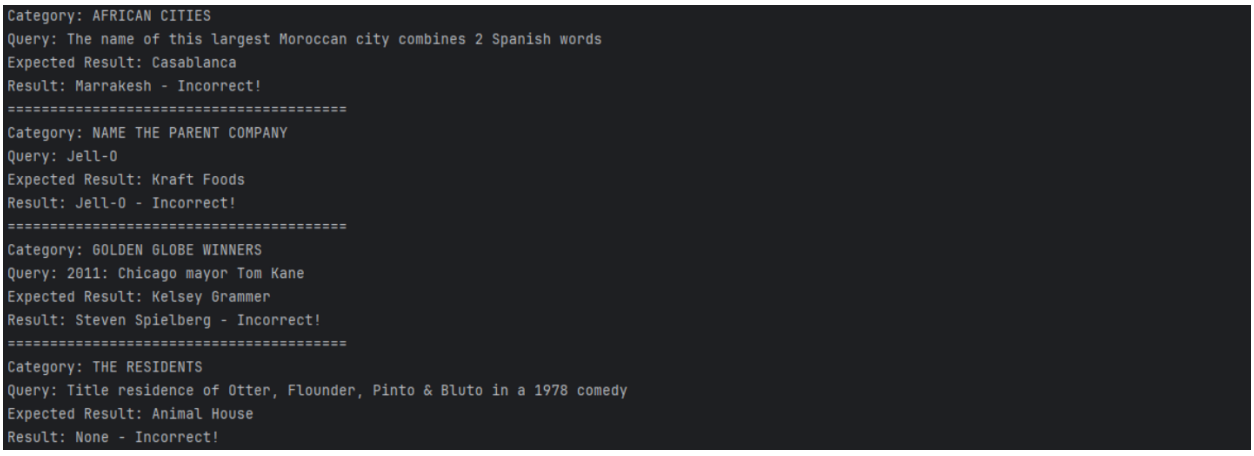
Sistemul simplu poate răspunde la întrebările corecte datorită implementării metodelor eficiente de indexare și căutare. Prin crearea unui index corespunzător sistemul este capabil să identifice rapid și să furnizeze răspunsuri relevante, în special în cazul întrebărilor pentru care există un răspuns clar și bine definit în datele furnizate. De asemenea, simplitatea sistemului permite o execuție eficientă și ușor de înțeles, fără a necesita resurse computaționale complexe.



Intrebari la care s-a raspuns corect

5.4 What problems do you observe for the questions answered incorrectly?

Pe de alta parte, faptul ca implementarea este una simpla nu favorizeaza majoritatea cazurilor. Se observa ca intrebarile primesc un raspuns, dar nu e cel corect. Cu toate acestea, totusi, raspunsurile tind sa fie din categoria ceruta in query, cum ar fi, orase, nume de personalitati etc.



Intrebari la care s-a rapsuns gresit