Apprentissage statistique

Projet – Question Answering

Olivier Schwander <olivier.schwander@sorbonne-universite.fr>

2023-2024

Attention Voir sur la page du cours pour les consignes pour le rendu.

Remarque Ce sujet est accompagné de plusieurs fichiers qui serviront d'aide.

Même si la quasi totalité des systèmes d'intellence artificielle reposent sur des réseaux de neurones, la plupart ne se limitent pas au tryptique: construction d'un dataset; mise au point d'une architecture et apprentissage par descente de gradient. Pour la plupart des tâches, au-delà des modèles deep proprements dits, qui seront souvent pré-entraînés, il s'agit en effet de combiner de façon non triviale ces différents modèles.

On va s'intéresser dans ce projet à la tâche de *question answering* qui combine:

- interprétation d'une question en langue naturelle;
- recherche d'information dans une base de connaissance;
- génération de la réponse.

Dataset

On utilisera le dataset NQ-Open https://huggingface.co/datasets/nq_open qui contient un ensemble de questions et les réponses associées. Par exemple:

```
{
    "question": "names of the metropolitan municipalities in south africa",
    "answer": [
        "Mangaung Metropolitan Municipality",
        "Nelson Mandela Bay Metropolitan Municipality",
        "eThekwini Metropolitan Municipality",
        "City of Tshwane Metropolitan Municipality",
        "City of Johannesburg Metropolitan Municipality",
        "Buffalo City Metropolitan Municipality",
        "City of Ekurhuleni Metropolitan Municipality"
    ]
}
```

Les questions ont la particularité d'avoir toujours leur réponse disponible dans Wikipedia.

Le dataset est décomposé en deux sous-ensemble:

- *train*: 90000 exemples utilisable par exemple pour construire ou fine-tuner des embeddings (a priori il n'y en a pas besoin dans ce projet),
- *validation*: 3600 exemples pour calculer les métriques.

L'évaluation se fait avec la méthode *exact match*: une réponse est considérée comme exacte si et seulement si elle est exactement identique (à la casse près) à la réponse attendue.

Par exemple, pour la question "Où se situe la tour Eiffel ?":

- réponse attendue "Paris, France",
- mauvaise réponse "France, Paris".

Outils

Modèles de langue

On propose d'utiliser plusieurs modèles de langue: GPT2, ChatGPT et Llama2.

ChatGPT C'est le modèle le plus simple à utiliser au travers de l'API et avec la clé fournie. Il tourne sur les serveurs d'OpenAI donc indépendamment de la machine qu'on utilise.

Llama2 Ce modèle a l'intérêt d'être disponible au téléchargement et il existe des versions quantifiées utilisables sur une machine personnelle. Ça reste plus compliqué à mettre en œuvre pour ce projet, et probablement plus lent à l'exécution. Si c'est trop lent, on peut limiter l'évaluation à quelques exemples au lieu de parcourir tout le dataset.

GPT2 Contrairement aux précédents modèles, il s'agit d'un *petit* modèle de langue. Il a l'intérêt d'être petit et rapide, donc utilisable largement sur une machine personnelle. Attention, il ne s'agit pas d'un modèle de dialogue: on ne peut pas discuter et lui poser des questions, mais seulement compléter l'entrée qu'on lui envoie.

Bibliothèques

Huggingface

Pyserini

Langchain

Différentes approches

Closed-book

Les gros modèles ont une grande capacité à stocker des informations. On pose ici directement la question à un LLM et on regarde la réponse.

Il faut travailler le prompt pour obtenir une réponse courte au bon format.

Pure information-retrieval

On pose la question directement à un moteur de recherche (Google ou autre) et on construit la réponse à partir des résultats.

On utilisera Lanchain pour effecturer la requête au moteur de recherche. On utilisera Duckduckgo qui a l'avantage de ne pas nécessiter de clé d'API ou d'authentification.

Comment construire la réponse ?

- utiliser un modèle Extractive QA https://huggingface.co/timpal0l/mdeberta-v3-base-squad2, capable de générer directement des réponses courtes à partir d'un contexte;
- utiliser un gros modèle avec le bon prompt.

Comme dans toutes les approches de *Retrieval Augmented Generation*, le prompt prendra en général une forme de ce type:

```
Answer only with the following context:
- [First document]  \
- [Second document] |
...                 | L'étape de retrieval a pour but de remplir ces cases
...                 |
- [Last document]   /

[Question]
```

Par exemple:

```
Answer only with the following context:
- Harrisson Ford is an American actor
- Harrisson Ford was born in 1942
- Harrisson Ford portrayed Indiana Jones

How old is Harrisson Ford ?
```

En changeant la date de naisssance, on peut s'assurer qu'avec cet exemple le modèle n'utilise vraiment que le contexte fourni.

Modèle probabiliste: BM25

BM25 est un modèle à base de sac de mots qui est longtemps resté l'état de l'art en matière de recherche d'information. On utilisera un index précalculé utilisable grâce à Pyserini.

Chaque document présent dans l'index est un fragment d'article d'une longueur de 100 mots. On génère la réponse à partir des documents récupérés.

Modèles neuronaux

L'index est cette fois-ci construit par apprentissage de représentation. Il s'agit d'un index qualifié de dense (par opposition au sac de mot qui est un vecteur sparse), dans un espace vectoriel.

Deux fonctions d'embedding sont utilisées:

- un embedding de documents, utilisé lors de la création de l'index Wikipedia;
- un embedding de questions, utilisé lors de la recherche pour projeter les questions en langue naturelle vers le même espace latent que les questions.

On pourrait encore utiliser Pyserini mais l'index devient beaucoup trop gros (60Go de stockage et autant de mémoire vive) et la recherche beaucoup trop lente pour être gérée sur une machine personnelle. Dans le cadre du projet on utilisera des résultats de recherche précalculés (et donc utilisable uniquement avec les questions de la base d'évaluation du dataset *nq_open*): c'est le but du module *fakesearch* fourni avec le projet.

Comme précédemment, on génère la réponse à partir des documents récupérés.

Tool-based answering

Dans cette variante, le LLM est utilisé pour la plupart des questions, mais en cas d'incertitude, on fait une requête vers l'index.

La difficulté est de savoir quand le LLM n'est pas sûr de lui.

Attendus, questions et perspectives

Rapport

Un rapport de 6 à 10 pages est attendu (un peu plus, un peu moins, peu importe, mais sans jouer avec les marges ou les double colonnes pour en faire rentrer plus…). La clarté et la lisibilité sont primordiales.

Le contenu attendu dans le rapport est le suivant:

- description des modèles utilisés;
- évaluation expérimentales;
- réflexion sur les limites des approches utilisées et des ouvertures.

Le code n'est pas demandé.

Remarque Comme il s'agit d'un projet assez technique, il est évident qu'il ne sera pas possible d'essayer toutes les approches proposées, ni même de les évaluer en profondeur. Le minimum attendu est de pouvoir utiliser au moins une méthode de question answering et de pouvoir réaliser au moins une petite analyse.

Évaluation expérimentale

Quelques implémentations des approches proposées sont attendues: pas forcément tout et pas forcément exactement ce qui est décrit ici (mais tout doit être clair dans le rapport).

L'analyse sera constituée de:

- les scores des différents modèles;
- une comparaison entre eux;
- une brève analyse des différences.

Remarque Il n'est pas indispensable d'évaluer sur le dataset complet. En fonction du temps disponible et de la puissance de calcul, on peut se contenter de quelques centaines, voire dizaines d'exemples. Vous ne serez pas jugé sur les scores, ni sur la taille de l'évaluation, mais sur l'analyse et la pertinence de vos remarques.

Ouvertures

Réfléchir aux limites des approches suggérées ici et proposer, sans implémentation, des idées de contributions pour améliorer le système de question-réponse.

Quelques suggestions:

- Aspect temporel avec des réponses qui peuvent évoluer dans le temps ?
- Gestion du nombre de documents dans le contexte et de leur taille ?
- Compromis entre la connaissance contenue dans le LLM et celle contenue dans un index externe ?
- Fine-tuning éventuel ?

Annexes

Installation de Pyserini

L'installation est assez délicate, les instruction suivantes sont inspirées de https://github.com/castorini/pyserini/blob/master/docs/installation.md:

```
conda create -n pyserini python=3.10 -y
conda activate pyserini
conda install -c conda-forge openjdk=11 maven -y
conda install -c conda-forge lightgbm nmslib -y
conda install -c pytorch faiss-cpu mkl=2021 blas=1.0=mkl pytorch -y
pip install pyserini
```

Une version sans conda devrait marcher aussi, à condition d'avoir Python 3.10 et pas une version plus récente:

```
python -m venv serini
source serini/bin/activate
sudo apt update
sudo apt install default-jdk build-essential
pip install faiss-cpu
pip install torch --index-url https://download.pytorch.org/whl/cu118
pip install pyserini
```

Pyserini et index probabiliste

On fait la recherche avec un index pré-calculé sur Wikipedia, disponible dans Pyserini (attention, 11Go d'espace disque nécessaire):

```
import json
from pyserini.search.lucene import LuceneSearcher

searcher = LuceneSearcher.from_prebuilt_index('wikipedia-dpr-100w')
```

```
hits = searcher.search('How old is Harrisson Ford ?')

for i in range(0, 10):
    docid = hits[i].docid
    score = hits[i].score
    content = json.loads(searcher.doc(docid).raw())["contents"]

    print(score, docid, content)
```

Pyserini et index dense

À titre de référence (ne pas faire tourner ce code), voici quand même un fragment de code pour faire la recherche dans les embeddings avec Pyserini:

```
from pyserini.search.faiss import FaissSearcher, DprQueryEncoder

encoder = DprQueryEncoder('facebook/dpr-question_encoder-single-nq-base')
searcher = FaissSearcher.from_prebuilt_index(
    'wikipedia-dpr-100w.dpr-single-nq',
    encoder
)
hits = searcher.search('what is a lobster roll')
```

GPT2

https://huggingface.co/gpt2

ChatGPT avec le module OpenAI

```
from openai import OpenAI

client = OpenAI(api_key=LA_CLÉ_ICI)

chat_completion = client.chat.completions.create(
    messages=[
        {
            "role": "user",
            "content": "Who is Joe Biden ?",
        }
    ],
    model="gpt-3.5-turbo",
)
```

ChatGPT avec Langchain

Llama2

Pour Llama2, on a besoin de:

- Ollama https://ollama.ai/ pour télécharger et faire tourner le modèle
- Langchain pour l'utiliser