NLP project 1 report

TripAdvisor Recommendation Challenge
Beating BM25

Stella HU - *Individual project*

DIA2 ESILV A5

Google colab link :
https://colab.research.google.com/drive/1PtlFyO7yfCl4Lp6umvlc11PbowP5ytAN?usp=sharing

Goal of the project

The main goal of the project is making a hotel recommendation system while beating BM25. The evaluation metrics is the MSE (mean squared error) between the ratings of query hotel and the preticted hotels' ones. The hotel recommendation needs to be solely based on the text reviews : no rating should be involved in the training of the model.

1st step : Learn about the dataset and preprocessing

Understanding the dataset

First of all, the dataset is composed of two csv files : `ratings.csv` and `offerings.csv` .

The first one contains all of the ratings in our sample and the other one contains information about each hotel. To make sure to compare places accurately, it is needed to select the ratings that are only rated on these aspects : 'service', 'cleanliness', 'overall', 'value', 'location', 'sleep_quality', 'rooms'. Therefore, a code was done to make this first filtration .

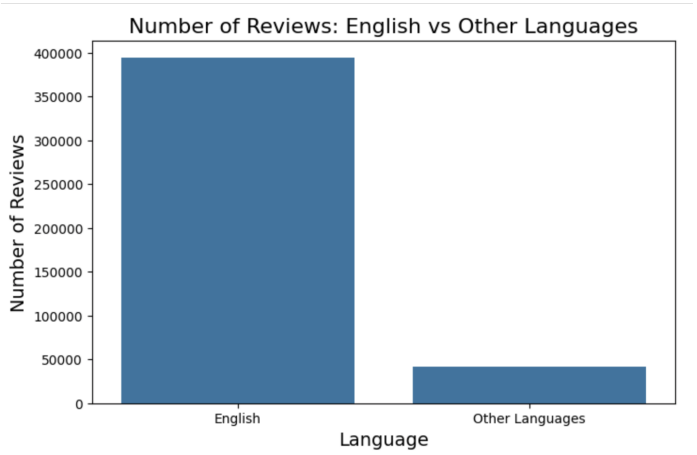
To have a global vision of the dataset, the first important step was to join the two datasets to make an unique one, especially by merging the hotel ID columns (**id and offering_id**). The columns of this unique dataset are now details about the reviews and hotels.

Selection of the reviews

Before processing the textual reviews, it is important to make sure that all the reviews will be processed in the same languages. I therefore used the `fasttext` module with its `'lid.176.bin'` model to detect all the languages of the reviews. A new column 'language' was created to indicate in which language a review was written. With this done, I used `print(data['language'].unique())` to have a view on all the different languages of the dataset:

```
['__label__en' '__label__es' '__label__it' '__label__de' '__label__fr'
 '__label__pt' '__label__ja' '__label__no' '__label__nl' '__label__sv'
 '__label__ru' '__label__zh' '__label__da' '__label__tr' '__label__el'
 '__label__pl' '__label__ko' '__label__sh' '__label__ar' '__label__ca'
 '__label__id' '__label__th' '__label__hu' '__label__fa' '__label__nn']
```

I presupposed that the dominant language was English, so I plotted the number of reviews in English VS other languages :



It is observable that the vast majority (roughly 400000) of the reviews were in English, so I had the idea of translating the remaining reviews in English with the `'Helsinki-NLP/opus-mt-mul-en'` model from MarianMTModel (transformers). *[This part is not in the final notebook]*

First, I tested if this model could translate all the languages of the dataset by testing with a simple sentence in each language :

```
from transformers import MarianMTModel, MarianTokenizer

model_name = 'Helsinki-NLP/opus-mt-mul-en'
model = MarianMTModel.from_pretrained(model_name)
tokenizer = MarianTokenizer.from_pretrained(model_name)

test_sentences = {
    "Spanish": "Hola, ¿cómo estás?",
    "Italian": "Ciao, come stai?",
    "German": "Hallo, wie geht es dir?",
    "French": "Bonjour, comment ça va?",
    "Portuguese": "Olá, como vai?",
    "Japanese": "こんにちは、お元気ですか？",
    "Norwegian": "Hei, hvordan går det?",
    "Dutch": "Hallo, hoe gaat het?",
    "Swedish": "Hej, hur mår du?",
```

```
    "Russian": "Привет, как дела?",
    "Chinese": "你好，你好吗？",
    "Danish": "Hej, hvordan går det?",
    "Turkish": "Merhaba, nasılsınız?",
    "Greek": "Γεια, τι κάνεις;",
    "Polish": "Cześć, jak się masz?",
    "Korean": "안녕하세요, 어떻게 지내세요?",
    "Serbo-Croatian": "Zdravo, kako si?",
    "Arabic": "مرحباً، كيف حالك؟",
    "Catalan": "Hola, com estàs?",
    "Indonesian": "Halo, apa kabar?",
    "Thai": "สวัสดีครับ คุณสบายดีไหม?",
    "Hungarian": "Szia, hogy vagy?",
    "Persian": "سلام، چطوری؟",
    "Norwegian Nynorsk": "Hei, korleis går det?"
}

for lang, sentence in test_sentences.items():
    tokens = tokenizer(sentence, return_tensors='pt', padding=True)
    translated_tokens = model.generate(**tokens)
    translated_text = tokenizer.decode(translated_tokens[0], skip_special_tok
    print(f"{lang}: {translated_text}")
```

I finally got the right output for each language's sentence in English so it was confirmed that the model could translate all the reviews in English. To translate all the non-English reviews, this code was used :

```
model_name = 'Helsinki-NLP/opus-mt-mul-en'
model = MarianMTModel.from_pretrained(model_name)
tokenizer = MarianTokenizer.from_pretrained(model_name)

def translate_text(text):
    try:
        tokens = tokenizer(text, return_tensors="pt", padding=True, truncatio
        translated_tokens = model.generate(**tokens)
        return tokenizer.decode(translated_tokens[0], skip_special_tokens=Tru
    except Exception as e:
        return text

def translate_reviews(df):
    updated_texts = []
    updated_titles = []
    non_english_mask = df['language'] != '__label__en'

    for index, row in tqdm(df[non_english_mask].iterrows(), total=non_english
     desc="Traduction des avis"):
        translated_text = translate_text(row['text'])
```

```
            translated_title = translate_text(row['title'])
            updated_texts.append((index, translated_text))
            updated_titles.append((index, translated_title))

        for idx, text in updated_texts:
            df.at[idx, 'text'] = text
        for idx, title in updated_titles:
            df.at[idx, 'title'] = title
        return df

data_translated = translate_reviews(data)
display(data_translated)
```

The main problem was the time I would take to translate all of the non-English reviews, because this processus was very costly :

```
/opt/anaconda3/lib/python3.12/site-packages/transformers/models/marian/tokenizati
emoses.
  warnings.warn("Recommended: pip install sacremoses.")
Traduction des avis:   0%|                                | 0/41984 [00:00<?, ?it/s]
```

Error displaying widget

```
Traduction des avis:   0%|                | 13/41984 [00:49<36:00:41,  3.09s/it]
```

I realized that it was almost impossible to make this on time, therefore I abandoned the idea of translating the reviews. I finally decided to only keep the English reviews, as the other languages reviews only represent nearly 10% of the dataset.

```
Total number of reviews : 436356
Remaining english reviews : 394372
```

Finally, I created a new column called `'combined_text'` to concatenate the title of the review and the review itself in one text, to keep all informations.

2nd part : Making our recommendation system with BM25

Pre-processing

The first step in pre-processing was to clean the combined text into a new '`cleaned_text'` column by applying :

- HTML removal : `.apply(lambda x: re.sub(r'<.*?>', '', x) if isinstance(x, str) else '')`
- URL removal : `.apply(lambda x: re.sub(r'http\S+|www\S+', '', x))`
- Special characters removal : `.apply(lambda x: re.sub(r'[^a-zA-Z\s]', '', x))`
- Lowering : `.str.lower()`
- Stop words removal : `set(stopwords.words('english'))`

The second step was tokenization, as the result of the previous preprocessing is a list of non tokenized cleaned text.

The third step was creating a new dataframe by grouping all our tokenized reviews per `offering_id` (hotel ID). It is also important to group the tokens per hotel and create a dictionary to make it easier and faster for our model to retrieve each hotel's combined tokens.

BM25 model for our recommendation system

Before creating the function that will recommend a hotel with BM25, the creation of the corpus is made and the application of the model as well, to avoid restarting these steps each time the function is called. When these steps are done, the function `recommend_hotel_bm25(hotel_id_query)` can finally be created :

```
def recommend_hotel_bm25(hotel_id_query):

    tokens_query = hotel_tokens_dict.get(hotel_id_query, [])
    scores = bm25.get_scores(tokens_query)

    for i, score in enumerate(scores):
        if grouped.iloc[i]['offering_id'] == hotel_id_query:
            scores[i] = -1

    best_index = np.argmax(scores)
    recommended_hotel_id = grouped.iloc[best_index]['offering_id']

    return recommended_hotel_id
```

Given a query hotel ID, it retrieves the associated tokens and computes BM25 relevance scores for all hotels. To avoid self-recommendation, the score for the query hotel is set to -1. The function then identifies the hotel with the highest BM25 score and returns its ID as the recommended hotel.

```
#test
import time

start_time = time.time()
recommended_hotel = recommend_hotel_bm25(87617)
end_time = time.time()
execution_time = end_time - start_time

print(f"Execution time : {execution_time:.4f} secondes")
print(f"Recommanded hotel : {recommended_hotel}")
```

```
Execution time : 168.0513 secondes
Recommanded hotel : 114581
```

It is noticeable that the model seems to work

3rd part : Making our recommendation system that beats BM25 using SentenceTransformers (SBERT)

When testing the system based on BM25, I noticed that the model was very strong. Starting from that, I decided to keep BM25 as a base and improve it with another model. After researches, Sentence Transformers (SBERT) seemed to be the best to complete the BM25 model by providing a semantic analysis dimension. Having this combination gives the best of both models:

- **BM25**: Captures keyword-based relevance effectively, emphasizing exact term matches and term frequency, making it suitable for precise lexical retrieval
- **SBERT**: Encodes semantic meaning, allowing for nuanced understanding of context, synonyms, and paraphrases, enabling recommendations based on deeper textual relationships.

Pre-processing

The preprocessing part for SBERT is different than the one for BM25. Indeed, while BM25's corpus needs tokenized words, SBERT needs sentences to make semantic analysis. To make it possible, here are the steps done to the 'combined_text' column :

- HTLM removal : `.apply(lambda x: re.sub(r'<.*?>', '', x) if isinstance(x, str) else '')`
- URL removal : `.apply(lambda x: re.sub(r'http\S+|www\S+', '', x))`
- Special characters **except punctuation characters** removal : `.apply(lambda x: re.sub(r'[^\w\s.,!?;:]', '', x))`
- Lowering : `.str.lower()`

After all of that, the cleaned text is grouped by 'offering_id', such as the tokens for BM25 (what we have previously done).

Recommendation system by hybriding Sentence Transformers with BM25

As the BM25 model and its corpus have already previously been created, the remaining step is to make the SBERT model : I chose `'all-MiniLM-L6-v2'` from Sentence transformers, which is a lightweight model optimized for sentence embeddings and semantic search developed by Hugging Face.

First, the sentences for each hotel needs to be embedded (represented in a vectorial space). These steps are done out of the function to avoid restarting these steps each time the function is called.

When the embeddings were created, the function `recommend_hotel_hybrid(hotel_id_query, alpha)` is made :

```
def recommend_hotel_hybrid(hotel_id_query, alpha):

    tokens_query = hotel_tokens_dict.get(hotel_id_query, [])
    bm25_scores = bm25.get_scores(tokens_query)
    for i, score in enumerate(bm25_scores):
        if grouped.iloc[i]['offering_id'] == hotel_id_query:
            bm25_scores[i] = -1
```

```
    hotel_query = grouped[grouped['offering_id'] == hotel_id_query].iloc[0]
    query_embedding = model_st.encode(hotel_query['cleaned_text'], convert_to
    cos_similarities = util.pytorch_cos_sim(query_embedding, embeddings)[0].c
    cos_similarities[grouped['offering_id'] == hotel_id_query] = -1

    bm25_scores_normalized = (bm25_scores - np.min(bm25_scores)) /
                             (np.max(bm25_scores) - np.min(bm25_scores))
    cos_similarities_normalized = (cos_similarities - np.min(cos_similarities
                                  (np.max(cos_similarities) - np.min(cos_simi

    bm25_weight = alpha
    sbert_weight = 1 - alpha

    combined_scores = bm25_weight * bm25_scores_normalized + sbert_weight * c
    best_index = np.argmax(combined_scores)
    recommended_hotel_id = grouped.iloc[best_index]['offering_id']

    return recommended_hotel_id
```

This hybrid recommendation system combines BM25 and SBERT to recommend the most relevant hotel based on a query hotel's description. The process integrates the strengths of lexical and semantic matching to improve the quality of recommendations:

- **BM25 Scoring**: Computes relevance scores based on token overlap between the query and other hotels, emphasizing exact term matching.
- **SBERT Embeddings**: Uses Sentence Transformers to calculate semantic similarity between the query hotel's text and others, capturing deeper contextual meaning.
- **Score Normalization**: Normalizes BM25 and SBERT scores to a common scale using min-max normalization for fair combination.
- **Weighted Combination**: Merges the normalized scores using a tunable parameter `alpha` to balance lexical and semantic importance, making the alpha parameter tunable is a effective way to decide which weight to give to BM25 and SBERT according to the lowest MSE. In the function, alpha refers to BM25 weight.
- **Final Recommendation**: Selects the hotel with the highest combined score as the recommendation.

Testing the model

MSE computing

To evaluate the model, it is needed to compute the MSE between the query hotel and the corresponding recommended hotel. To do so, a dataframe name `'ratings_df'` was created to group the ratings means on each aspect by 'offering_id'. However, `'the ratings'` column in the initial dataset contains json documents. The extraction of json data for ratings in each aspect was

mandatory to make a proper dataset. The aspects werre then grouped in a column, and using the `.pivot()` function, I was able to group every rating means for each aspects (each aspect is a column).

I decided to compute each aspect's rating means's MSE instead of computing the global mean's MSE to not lose any information :

- By computing the MSE of the global mean of a hotel, we can lose precious information when it comes to the nuance of ratings about different aspects
- By computing a MSE for each aspect, then making the average with all the aspects to get a global mean MSE, we can keep the subtilities in the difference of ratings by aspect.

To compute the MSE, a function was created `calculate_aspect_mse(hotel_id_query, recommended_hotel_id)`. For a pair of hotel IDs, the individual aspect MSE values are stored in a dictionary, and their average is calculated. The function returns both the per-aspect MSE and the overall average MSE.

Evaluation

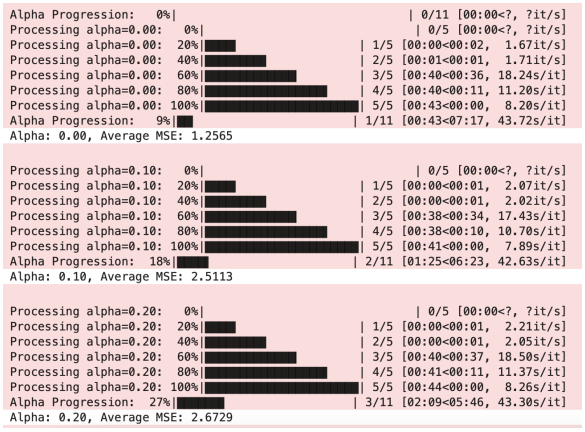
While testing the model, I noticed that the execution time was very long.

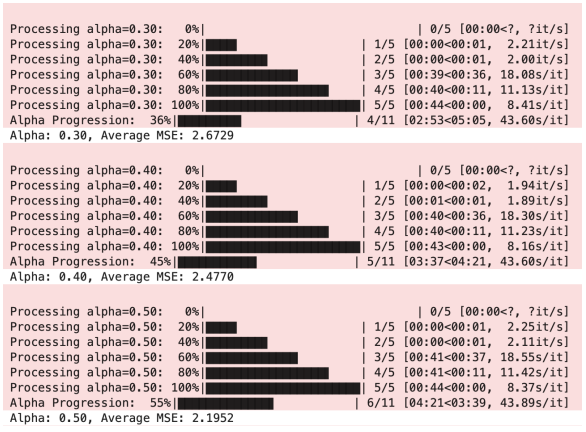
```
#test
import time

start_time = time.time()
recommended_hotel = recommend_hotel_hybrid(87617, 0.5)
end_time = time.time()
execution_time = end_time - start_time

print(f"Execution time : {execution_time:.4f} secondes")
print(f"Recommanded hotel : {recommended_hotel}")
```

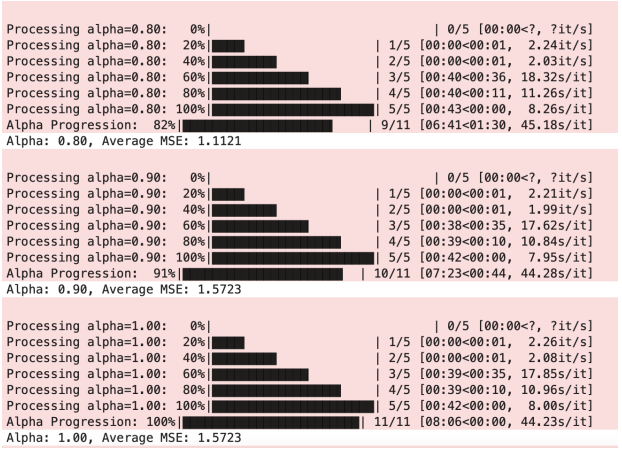
```
Execution time : 171.0286 secondes
Recommanded hotel : 609738
```

I therefore chose to only use a small random sample (5) to test the model and average the MSE scores. To evaluate the model, a code tests different values of **alpha** to optimize a hybrid hotel recommendation system combining BM25 and SBERT scores. For each alpha (ranging from 0 to 1, 0 meaning 100% SBERT and 1 meaning 100%BM25) , it calculates the MSE between query hotels and recommended hotels, averaged across a sample. The alpha that minimizes the average MSE is identified as the best.

Output :

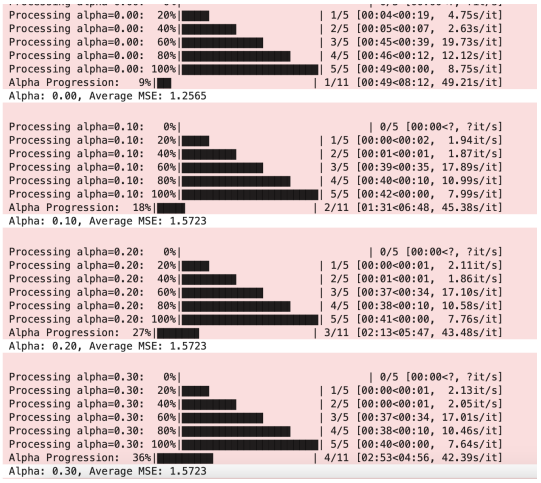


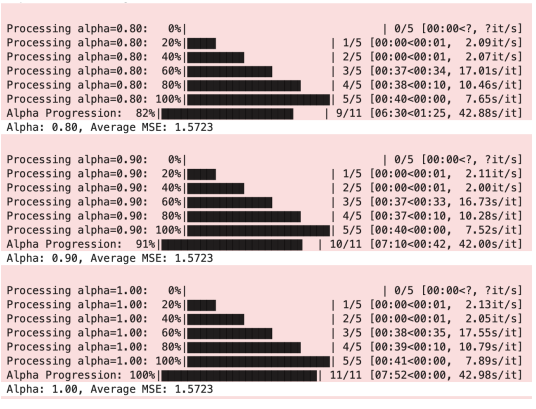




We can see that the best alpha here is 0.80 : it means that the lowest MSE is reached when BM25 has a weight of 0.80 and SBERT has the remaining weight of 0.20

Here is a test when the scores are not normalized :





We can notice that there is a convergence of MSE as soon as the SBERT weight is introduced.

When BM25 and SBERT scores are not normalized, their scales might differ significantly, causing one scoring system to dominate the other as the weight parameter (alpha) shifts. This is why

normalizing the scores in the `recommend_hotel_hybrid(hotel_id_query, alpha)` function is crucial.

Limits of the model : as the model has been tested each time on samples of 5 random hotel IDs, it does not cover the entire dataset. Factors such as luck can influence the results, even though all the tests so far showed that BM25 was beaten by the SBERT / BM25 combination (mostly with alpha=0.8). The execution time is also a big issue in the model, which could be improved ulteriorly.

Application

The systems asks to the user to input a valid hotel ID and returns the most recommended hotel by calling the function `recommend_hotel_hybrid(query_hotel_id, best_alpha)` with :

- **query_hotel_id** being the user's input
- **best_alpha** being the best BM25 weight previously computed with the evaluation code.

	Hotel query	Recommendation
Hotel ID	87617	114581
Name	Hyatt Regency Chicago	Swissotel Chicago
Class	4.0	4.0

Example with hotel 87617 - with computed best alpha = 0.80, meaning that the model was 80% BM25 and 20% SBERT.