Task 3.2 – Natural Language Processing (practical)

In this task, we will utilize NLTK and transformers to perform text analytics. You have the option to create custom Python classes and methods, or just create command blocks in a Jupyter notebook. The course material contains many code snippets that you can use as a starting point:

a) [easy] Use NLTK to determine the language of an input text. Download texts in Italian, German, and English (or any other language) with the same encoding for simplicity. Employ NLTK's stop word lists to identify the text's language based on the stop occurrences (pick the language with most stop words).

b) [intermediate] We once again work with the movie dataset from Kaggle.com. This time, we will exclusively use the movie titles and apply sub-word tokenization, which involves fixed-length sequences of 2, 3, or 4 characters within words. You may use special codes to indicate the start or end of words. To simplify the task, utilize unidecode to convert words into non-accented versions. Create a set-of-word representation (using Python's set) for the sub-tokens extracted from the titles and ignore all other fields. For searching, employ the same tokenization and set-of-word approach as for the titles, and establish an appropriate similarity function to match queries with movie titles. https://www.kaggle.com/datasets/harshitshankhdhar/imdb-dataset-of-top-1000-movies-and-tv-shows.

c) [intermediate] We perform the same task as in b), but this time we will utilize sentence transformers. Select a suitable model for your hardware and explore various sentence transformer models. Encode the movie titles and proceed with a semantic search for your query:

```
from sentence_transformers import SentenceTransformer, util
model = SentenceTransformer('all-MiniLM-L6-v2')
a, b = model.encode(title_a), model.encode(title_b)
float(util.dot_score(a, b))
```

d) [difficult] Let's revisit task a) with language detection. The approach in a) relies on having sufficiently long texts with an adequate number of stop words for accurate language prediction. In the script, we discussed a method based on Naive Bayes, which operates with sub-word sequences. You can reuse the method from b) to generate these sub-sequences, first for learning the Naive Bayes likelihoods, and then for language prediction. To simplify, you can choose an equal prior for all languages and select 3-5 languages that use the same alphabet (eliminating alphabet-related rules). Extract and count all sub-sequences, but only keep the top-n sub-sequences per language (choose n as relatively small, such as 100 or 1000). During prediction, disregard sub-sequences for which we lack likelihoods (to avoid 0-posteriors).