Task 2 – Semantic Search on NiniSite

This project involves working with the PerCQA dataset, which contains around 1,000 Persian-language questions and over 21,000 answers from the NiniSite Q&A forum. The goal is to first perform exploratory data analysis and apply basic NLP techniques such as text cleaning, normalization, and tokenization on the informal Persian text. Then, a semantic retrieval system will be developed to retrieve and rank relevant answers based on a user's query using advanced semantic similarity methods.

1. Preprocessing (20 pts + 5 pts bonus)

Normalize Persian and Arabic characters (4 pts)

- Identify and fix Arabic characters (e.g., "ك" ,"ي") used in place of Persian ones ("ی","ک""). Use tools like hazm or parsivar after exploring the data for such cases.
- Look for inconsistent punctuation, extra spaces, or unusual symbols. Clean and standardize them using available text preprocessing tools.

Remove diacritics and unwanted characters (3 pts)

- Persian and Arabic texts may contain diacritics (e.g., "ِ ", " ّ ", "َ ", " ُ ") that aren't useful for most NLP tasks. Remove these symbols to simplify the text. Start by cleaning common ones like the examples above, then explore your dataset to find and handle others as needed.

Tokenization (3 pts)

- Tokenization means breaking text into smaller units like words or sentences, which is a key step in most NLP tasks. For example, turning a sentence into a list of words helps with tasks like text classification or sentiment analysis.You can perform tokenization using libraries such as hazm or parsivar, or any other method you prefer.

Remove stopwords (3 pts)

- Stopwords are very common words (like "برای" ,"که" ,"به" ,"از") that usually don't add meaningful information to text analysis. Removing them helps models focus on more important words.You can use the built-in stopword list in the hazm library or define your own list based on your task.

Stemming and Lemmatization (4 pts)

- Words in Persian often appear in different forms (e.g., "گفتند" ,"می‌گوید" ,"گفت"). Reducing them to a base form helps group similar words and improves model

performance. Use hazm.Stemmer() for root extraction or hazm.Lemmatizer() for dictionary forms.

> **Stemming** cuts off word endings to get the root (e.g., "رفت" → "رفتیم").
> **Lemmatization** finds the base dictionary form (e.g., "رفتن" → "رفته‌ام").
> This helps reduce vocabulary size and improve model performance.

Normalize informal stretching and repetition (3 pts)

- In casual writing, users often repeat letters for emphasis—like "عاااالییییی" instead of "عالی". These exaggerated forms can confuse NLP models and increase vocabulary size unnecessarily.

  Ask yourself: *Should "عاااالی" and "عالیییییی" really be treated as different words?*

- To handle this, use text normalization tools such as hazm or apply regular expressions to reduce repeated characters.

Replace informal or slang expressions(optional - 2 pts bonus)

- Informal words like "خخخ" or "عههههه" are common in social media texts but can reduce model accuracy if not handled properly.

  Replace slang terms with their standard equivalents using a custom dictionary or tools like hazm or manual mapping.

[Optional – 3 pts bonus] Displaying Persian Text Correctly

Persian text may appear with broken or disconnected letters in Jupyter Notebook due to Unicode limitations and lack of right-to-left rendering. To fix this, use arabic_reshaper for shaping and python-bidi for proper RTL display. This ensures that Persian words appear correctly in visualizations and outputs.

2. EDA (20 pts)

Understand the structure of questions and answers: (4 pts)

- First display 2 sample questions along with their corresponding answers.
- Then, compute the average and median lengths across the entire dataset—both by word count and by character count.
- Visualize the distribution of lengths using:
  - Histograms to observe shape.
  - Boxplots to detect outliers and spread.

Identify the most engaging questions: (3 pts)

- Find which questions (`QID`) have the highest number of answers. Analyze overall response rates to understand which topics get more attention.

Analyze user activity patterns: (4 pts)

- Use the `CDate` field to:
  - Determine peak hours and days when users are most active.
  - Identify when the platform is most responsive.Visualize this using Bar plots or line charts for hourly/daily activity.Heatmaps to show intensity of activity across time.

Detect top answer contributors: (3 pts)

- Count the number of answers posted by each user (`CUsername`).
- Create a bar chart of the top contributors (e.g., top 10 users) to show who is most active in the community.

Linguistic and word-level analysis: (6 pts)

- Extract and visualize most frequently used words in both questions and answers.
- Generate word clouds to capture common language patterns and user concerns.
- Go deeper with n-gram analysis:
  - Plot unigram, bigram, and trigram frequencies. Perform this both before and after stopword removal to observe the effect on meaningful patterns.

3. Analyse the dataset (60 pts)

You've recently been hired as a data scientist at Nini Site, a Q&A-based platform. Your first assignment is to improve the site's internal search functionality.

Currently, the site uses traditional keyword-based search algorithms, which are insufficient for retrieving results that are semantically similar to a user's query. To solve this problem, you propose using embedding-based search that captures the meaning of queries rather than just matching keywords.

After researching available options, you choose bge-m3, a multilingual embedding model capable of capturing semantic similarity across various languages. To store and search embeddings efficiently, you decide to use LanceDB, a vector database that is easy to set up and integrates well with modern embedding workflows.

Your job is divided into the following steps:

1. **Load the `bge-m3` embedding model and test it.**
    - Use the model to encode the `QBody` (question body) of a random row from the dataset.
    - Analyze the output of the model: What does it return? What do the components of the output represent? Explain their meanings and potential uses.
2. **Install LanceDB and set up an embedding function.**
    - Install the LanceDB Python library.
    - Use the `TextEmbeddingFunction` class to define a custom embedding function that uses the `bge-m3` model to encode question texts. Only use the dense embeddings.
3. **Define a schema for your vector database.**
    - Your schema must include at least:
        - `qid`: a unique identifier for each entry
        - `qbody`: the raw question text
        - `embedding`: the vector representation of the `qbody` using your custom embedding function
4. **Create and populate a LanceDB table.**
    - Use your defined schema to create a new table.
    - Load the Nini Site dataset (questions only, excluding comments) into this table.
    - Note that the embeddings must be generated automatically by LanceDB using the custom embedding function. You should not compute the embeddings manually before inserting the data into the database.
5. **Perform semantic search with LanceDB.**
    - Use LanceDB's `search` function to retrieve the top 5 semantically similar results for at least 5 different user queries.
    - Evaluate whether the results are semantically relevant to the query Manually.
6. **Implement classical full-text search using LanceDB.**
    - Read the documentation on how to index fields for full-text search.
    - Create a full-text index on `qbody` and perform the same 5 queries from Task 5.
    - Compare the results with those from semantic search. Which approach gives more relevant results?
7. **Research hybrid search techniques.**
    - Investigate and explain what hybrid search is and explain why hybrid search might be more effective than using one method alone.
8. **Evaluation methods.**
    - In this task you have manually checked whether the results were good or not. Research common evaluation metrics for search systems and only explain them briefly in your report (e.g., precision@k, recall, NDCG).

4. Answer Ranking Enhancement with a Reranker (10 pts Bonus)

After performing semantic search and retrieving relevant answers, apply a reranker model to improve the ranking of the results. A reranker evaluates each (question, answer) pair more precisely to determine better relevance ordering.

Example tools:
Use a ready-made model such as bge-reranker or a cross-encoder from the sentence-transformers library.

Steps:

- Perform semantic search for 5 user queries using the embedding-based method.
- Use the reranker to re-rank the top retrieved answers.
- Compare and report the difference in results before and after reranking.

Notes

- The search engine focuses only on the questions/topics in the dataset, not on the comments or answers.
- LanceDB provides an AskAI feature to help you understand its documentation and codebase—feel free to use it if you get stuck.
- You are required to answer questions related to the concepts used in the project, which are part of the 60-point third section of this task.