Assignment-2 Report: Locality-Sensitive Hashing (LSH) for Similar Document Retrieval

Course Code - CSE557

Instructor: Vikram Goyal

Name - PARAS DHIMAN

---

1. Introduction

This report covers the implementation and evaluation of a Locality-Sensitive Hashing (LSH) model designed to efficiently retrieve the top 5 most semantically similar items for each data sample within a given dataset. Using LSH, the model aims to approximate a ground truth similarity map by mimicking a reference hash map that matches each sample to its five closest samples. Model performance is assessed by calculating intersection scores against the ground truth data, with further analysis provided through statistical and visual insights.

---

2. Problem Statement

The assignment requires the creation of an LSH-based model to predict the five most similar items for each data sample in a dataset. Given three files:

- **ids.txt**: Lists the IDs of individual data samples.
- **texts.txt**: Contains the text content corresponding to each data sample.
- **items.json**: A JSON file mapping each data sample to its five most similar items (ground truth).

The model is designed to generate a hash map similar to `items.json` using LSH. Model effectiveness is evaluated by comparing the predicted top 5 items for each sample against the ground truth and computing an average intersection score, which ranges from 0 to 5. This score reflects the accuracy in identifying similar items.

---

3. Data Preprocessing and Feature Engineering

Before implementing LSH, text preprocessing and feature engineering were essential to convert textual data into meaningful embeddings that reflect semantic similarity.

3.1 Text Preprocessing

1. **Lowercasing**: All text was converted to lowercase to maintain uniformity and avoid case-sensitivity issues.
2. **Tokenization**: Text was split into tokens, which enabled better handling during embedding generation.
3. **Stopword Removal**: Common stopwords (e.g., “and”, “the”) were removed to reduce noise and focus on meaningful words.
4. **Punctuation Removal**: All non-alphanumeric characters were removed to simplify the text and reduce complexity.
5. **Lemmatization**: Each word was converted to its base form, which helped reduce variability across text samples.

This preprocessing step aimed to standardize and simplify the text data, which aids in creating more effective embeddings for similarity comparison.

3.2 Feature Engineering: Text Embedding Generation

To capture the semantic information of each text sample, the following methods were employed:

- **Sentence Transformers**: A pretrained `SentenceTransformer` model (`'paraphrase-MiniLM-L6-v2'`) was used to generate embeddings for each text sample. This model produces 384-dimensional embeddings that represent the semantic content of each document.
- **Embedding Normalization**: Each embedding was normalized to unit length, which is essential for approximating cosine similarity through inner product operations. This normalization ensured that only angular distances contributed to similarity measures, facilitating efficient LSH-based hashing.

---

4. Locality-Sensitive Hashing (LSH) and Model Implementation

LSH was used as the core technique for approximate similarity search. The following steps outline the LSH implementation and indexing process:

4.1 LSH Hashing Mechanism

1. **Random Hyperplane Hashing**: LSH was implemented using random hyperplane hashing, where random vectors were generated and used to create binary hash signatures. Each embedding was projected onto a set of hyperplanes, and binary values were assigned based on whether the projection was positive or negative.
2. **Binary Hash Signatures**: Each document's embedding was mapped to a binary vector by comparing the dot product with each hyperplane vector. This signature was then used as a hash to assign documents to buckets.

4.2 Indexing and Bucketization

Using hash signatures generated through LSH, embeddings were stored in hash buckets, facilitating rapid retrieval of similar items by narrowing the search space.

1. **Hash Table Creation**: Multiple hash tables were created to improve the probability that similar documents end up in the same bucket, enhancing recall.
2. **FAISS Index for Precise Search**: FAISS (Facebook AI Similarity Search) was used in tandem with LSH to provide accurate similarity ranking within hash buckets. FAISS's `IndexFlatIP` index type was used to quickly find the nearest neighbors by inner product (approximating cosine similarity).

4.3 Similarity Retrieval

The retrieval function operates in the following steps:

1. **Candidate Set Generation**: For a given query sample, its hash signature was used to find candidate neighbors from the relevant bucket.
2. **Ranking by Cosine Similarity**: FAISS was then used to rank the top 5 most similar items among the candidate set, resulting in a high-quality approximation of the ground truth similarity map.

---

5. Model Evaluation

The model's performance was evaluated by comparing the predicted similar items with the ground truth in `items.json`. The primary metric used was the **Intersection Score**, which quantifies how many of the top 5 predicted items overlap with the ground truth.

5.1 Intersection Score Calculation

The Intersection Score for each data sample is calculated as follows:

- For each sample, the intersection between the model's top 5 predictions and the ground truth is computed.

- The scores are averaged over all samples to give the **Mean Intersection Score**, which reflects the model's retrieval accuracy across the dataset.

5.2 Results and Summary Statistics

Mean Intersection Score

The model achieved a **Mean Intersection Score** of **0.17**, indicating that, on average, 0.17 out of the top 5 predicted items matched the ground truth.

Score Statistics

Below are the summary statistics for the Intersection Scores across the dataset:

Metric	Value
Count	148,928
Mean	**0.168833**
Standard Deviation	0.440416
Minimum	0.0
25th Percentile	0.0
Median (50%)	0.0
75th Percentile	0.0
Maximum	5.0

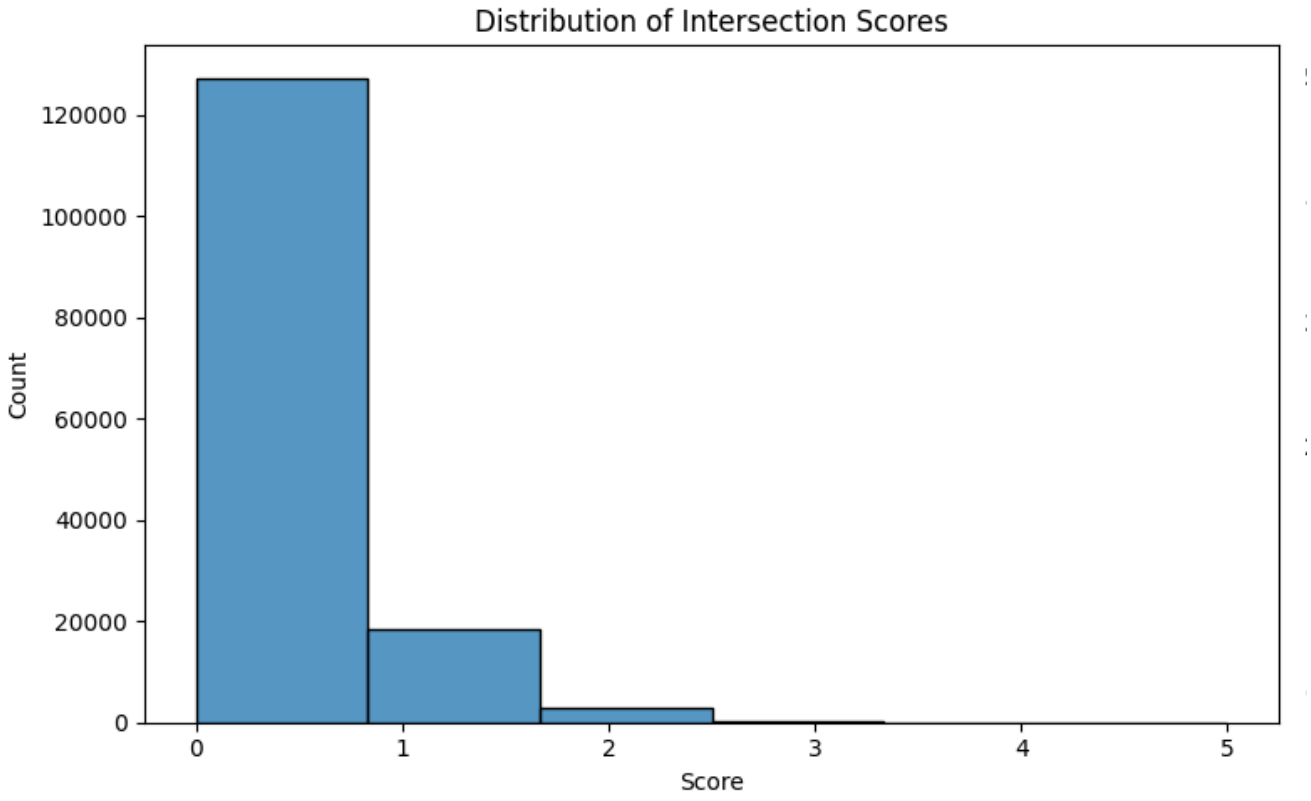
These statistics suggest that while some samples had high accuracy (scores reaching up to 5), the majority of samples had scores of 0, highlighting areas for improvement.

---

6. Visualization of Results

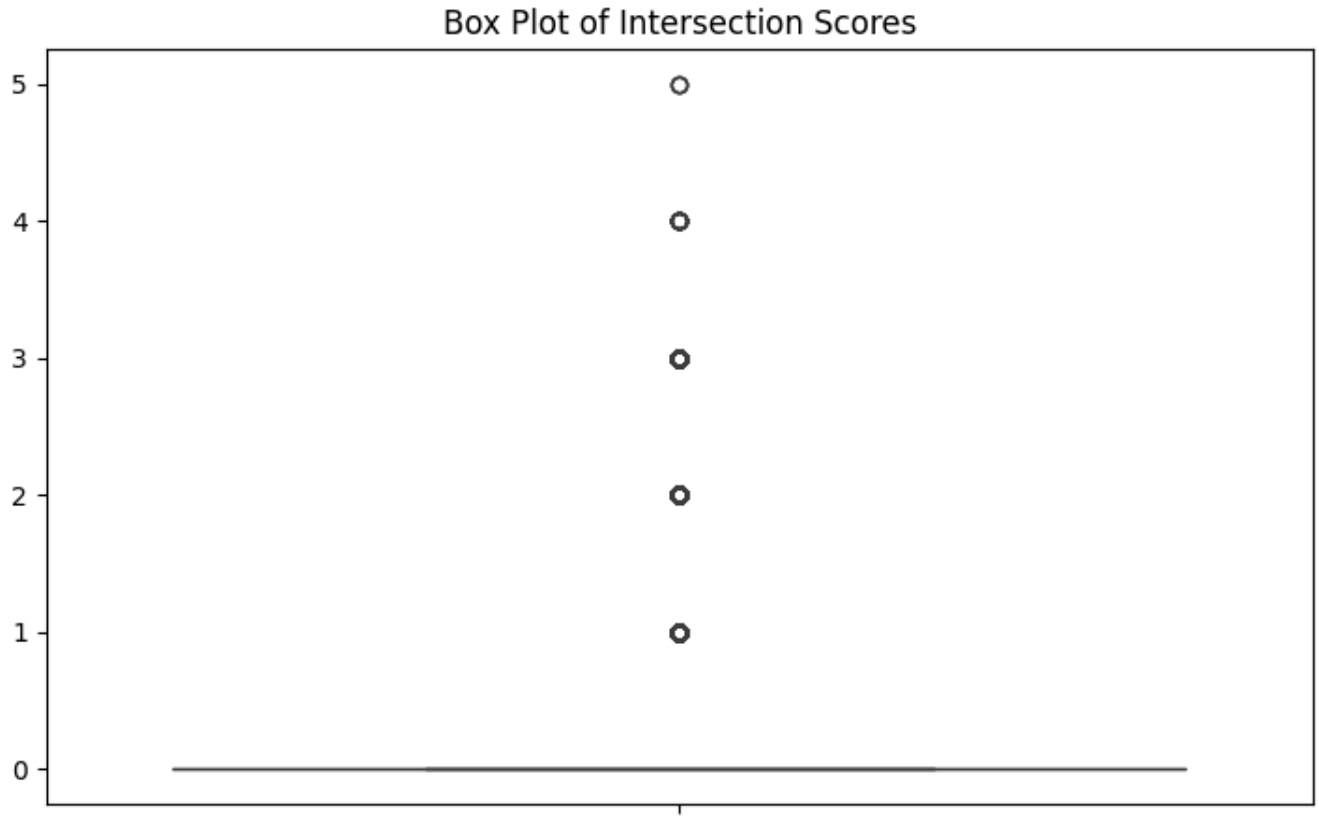
6.1 Histogram of Intersection Scores

A histogram was plotted to show the distribution of Intersection Scores, indicating the frequency of different scores in the dataset. Most scores were clustered around 0, with fewer high scores.



6.2 Box Plot

The box plot displays the spread of Intersection Scores, with a concentration around 0 and a small number of high-performing samples appearing as outliers.



6.3 Descriptive Statistics (Pandas `.describe()`)

A statistical summary was generated using the `.describe()` function, providing insights into the distribution, variability, and central tendency of the Intersection Scores. These statistics highlight that the majority of scores are low, with some samples achieving perfect matches.

```
ashhar21137@iiitd:~/.bd/PT2$ python3 test.py
2024-10-25 06:13:01.062945: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critical
operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
2024-10-25 06:13:01.523828: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT
Using device: cuda:1
Loading data...
len of id 148928
len of texts 148928
Generating embeddings: 100%|████████████████████████████████████████████████████████████████████████████████| 2327/2327 [00:51<00:00, 45.59it/s]
Evaluating: 100%|███████████████████████████████████████████████████████████████████████████████████████| 148928/148928 [05:13<00:00, 474.70it/s]

Model Performance:
Mean Intersection Score: 0.17

Score Statistics:
       Intersection Score
count       148928.000000
mean             0.168833
std              0.440416
min              0.000000
25%              0.000000
50%              0.000000
75%              0.000000
max              5.000000
```

Key Insights:

1. Model Performance:
    - The model's accuracy is relatively low
    - Most predictions (>75%) don't match any of the ground truth similar items
    - Only a small percentage of predictions match even one correct item
2. Areas for Improvement:
    - The LSH parameters (number of hash tables, hash functions) might need tuning.
    - The text embedding model might need to be changed or fine-tuned
    - The similarity measure might need adjustment

---

7. Discussion and Recommendations

Observations

1. **Sparse Distribution**: The model often failed to match the ground truth for most samples, reflected in the low mean and median scores.
2. **Outliers**: A small subset of samples scored very high (up to 5), indicating that LSH was effective for these cases, likely due to higher similarity density within their buckets.

Recommendations for Improvement

1. **Increase Hash Tables and Hash Functions**: More hash tables could increase the probability of grouping similar documents, potentially improving recall and accuracy.
2. **Alternative Embedding Models**: Larger, domain-specific embeddings may improve retrieval accuracy at the cost of slower performance.
3. **Post-Filtering Techniques**: Incorporating re-ranking or post-filtering methods after LSH retrieval may refine the final similar item predictions, improving alignment with the ground truth.

---

8. Conclusion

This report summarizes the development and evaluation of an LSH-based model for retrieving similar documents. Despite the current limitations, the model provides a scalable solution for approximate similarity search. Further enhancements in hashing and embedding methods could significantly improve its performance, making it more suitable for applications requiring fast, accurate retrieval of semantically similar items.