PROJECT REPORT FOR CS 839

Team ID: Group-10
Member names: Venkata Abhijeeth Balabhadruni, Sujay Chandra Shekara Sharma, Divy Patel

Problem Description

Document matching is a fundamental task in various domains such as information retrieval, natural language processing, plagiarism detection, and recommendation systems. Text tokenization forms the building blocks for implementing and achieving applications in such domains. Understanding how different tokenization methods and string-matching techniques affect the accuracy and efficiency of identifying matching documents is important. This exploration is crucial for optimizing document matching models for resource utilization and selecting the best algorithms to fulfil the user-query SLOs (Service Level Objectives) in the form of user-query accuracy, latency, cost, and throughput requirements.

In this project, we will build a framework that takes input as a list of news headlines, a target headline, and user-query SLOs. The output will be a list of headlines referring to the same news article. First, the framework evaluates different combinations of tokenizers and string-matching techniques (both rule-based and LLM-based) based on a certain set of performance metrics and chooses the combination that best fulfils the user's SLOs. The tokenizer-algorithm combination obtained from the above program can then be considered an optimal combination given the query requirements and can be used to generate the output list of headlines associated with the same news article.

The tokenizers under consideration for this project are:

1) Whitespace tokenizer
2) Word-level N-gram tokenizer
3) OpenAI's TikToken tokenizer [6]

The string-matching techniques utilized for this project include:

1) Jaccard
2) TF-IDF
3) An Open-Source LLM model called Distil-RoBERTa [2][3]

Finally, for each tokenizer and string-matching technique combination, we evaluated the following performance metrics:

1) Time taken to match a pair of headlines using the model
2) Precision
3) Recall
4) F1-Score

Data

We utilized the UCI Machine Learning News Aggregator Dataset [1], which comprises headlines, URLs, and categories for 422,937 news stories gathered by a web aggregator from March 10th to August 10th, 2014. In total, there are roughly 7,000 unique news stories referenced by these headlines.

The news categories in this dataset encompass business, science and technology, entertainment, and health. Articles referring to the same news item, such as multiple pieces on recently released employment statistics, are grouped together based on the alphanumeric story ID of the article, which serves as a label for training and testing purposes.

The dataset includes the following columns:

ID: numeric ID of the article

TITLE: headline of the article

URL: URL of the article

PUBLISHER: publisher of the article

CATEGORY: category of the news item (b: business, t: science and technology, e: entertainment, m: health)

STORY: alphanumeric ID of the news story discussed in the article

HOSTNAME: hostname where the article was posted

TIMESTAMP: approximate publication timestamp of the article in Unix time (seconds since midnight on Jan 1, 1970)

Among the columns provided in the dataset, the essential ones for our purposes are the TITLE and STORY columns. We will focus on these columns as they contain the information necessary for comparing article titles using our models to determine if they correspond to the same story.

On average, there are about 57 titles per story, with a total of approximately 400,000 titles (news article headlines) and around 7,000 stories.

Performance Measures

For each combination of tokenizer and string-matching technique, we measured the following metrics - Time taken to match a pair of headlines using the model, precision, recall, and F1 score.

Model Run time to match a pair of headlines

To assess the time required for matching two headlines using the model (tokenizer+string-matching technique), we employed the Python time module. Initially, we measured the cumulative time taken to match one headline with approximately 2000 headlines. Then, we divided this total time by 2000 to determine the average time for matching a pair of headlines. This method enabled us to accurately evaluate the efficiency of various model combinations.

Precision

Precision is a performance metric used for evaluating the accuracy of a model. It quantifies the reliability of positive predictions by expressing the ratio of true positive predictions to the total number of positive predictions made by the model. In other words, precision quantifies the reliability of positive predictions made by a model by indicating how often those predictions are correct.

The precision formula is:

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

Where:
True Positives (TP) are the number of correctly predicted positive instances.
False Positives (FP) are the number of incorrectly predicted positive instances.

High precision indicates that the model produces few false positive predictions, meaning that when it predicts a positive outcome, it is usually correct.

In the context of our model, the precision is the number of matching headlines successfully identified by our model, divided by the total number of predicted headlines for a particular target headline.

The method we employed to measure precision involved generating a random headline for each story and applying the model combination to obtain predictions. Subsequently, we calculated the number of True Positives and False Positives from these predictions. By utilizing these values, we computed the precision. This process was repeated for 1000 randomly selected headlines, and the average precision was calculated. Finally, this average precision value was updated in the SLO.csv file, representing the precision for that particular combination of tokenizer and string-matching technique.

Recall

Recall is a performance metric that measures the ability of a model to correctly identify all relevant instances from the total number of instances in the dataset. It quantifies the completeness of the model's predictions. Mathematically, recall is defined as the ratio of

the number of true positive predictions to the total number of actual positive instances in the dataset.

It is calculated using the formula:

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

Where:
True Positives (TP) are the number of correctly predicted positive instances.
False Negatives (FN) are the number of positive instances that were incorrectly classified as negative.

A high recall score indicates that the model is effectively capturing a large portion of the positive instances, while a low recall score suggests that the model is missing a significant number of positive instances.

In the context of our solution, the recall is the number of matching headlines successfully identified by our solution divided by the total number of matching headlines for a particular target headline.

The process by which we calculated the recall is similar to the one followed for obtaining our model's precision score. This recall score was then updated in the SLO.csv file, representing the recall for that particular combination of tokenizer and string-matching technique.

F1 Score

The F1 score is a metric commonly used to evaluate the performance of a model. It considers both the precision and recall of the model to provide a single score that balances between them.

The formula for the F1 score is:

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

The F1 score attains its optimal value of 1 when both precision and recall are perfect, and its lowest value of 0 when either precision or recall is absent. It represents the harmonic mean of precision and recall, thereby assigning equal significance to both metrics. This characteristic renders the F1 score particularly valuable in scenarios characterized by class imbalances or when both precision and recall are equally crucial. In our context, where the goal is to avoid irrelevant headlines while also ensuring matching headlines are not overlooked, this balance between precision and recall is paramount.

The process by which we calculated the F1-score is similar to the one followed for obtaining our model's precision and recall scores. This F1-score was then updated in the SLO.csv file, representing the F1-score for that particular combination of tokenizer and string-matching technique.

Finally, we computed the aforementioned metrics for all combinations and saved them in a file named SLO.csv. This file serves as the basis for selecting the optimal combination according to the user's Service Level Objective (SLO) requirements. Further details on this process are outlined in the solutions section.

Challenges

Handling foreign/corrupted news article headlines

While working on the UCI Machine Learning News Aggregator Dataset, we ran into an issue while trying to tokenize the news headlines. We found out that some of the news headlines contained characters that couldn't be represented using the UTF-8 encoding which is used by TikToken by default. This could be the case due to the character either being from some foreign language that isn't represented using UTF-8 or due to some data

corruption of the news headline. Here are some example news headlines for which we encountered this problem:

i) *Paris car ban set to start after pollution hits highŸœ€åÿ±ÿ≥ ŸÖ€å/Øÿß/ë€åŸà/©€íÿ±Ÿàÿ≤ÿßŸÜ€Å*

ii) *I–≤–Ç,Ñ¢m Not Dead –≤–Ç,Äú Wayne "Newman" Knight*

In order to resolve the problem of handling foreign/corrupted news article headline representations, we decided to first perform data cleaning on the input news article headlines. We wrote a Python script that iterated over all the headlines present in the dataset. For each headline, we attempted to encode each character in the headline using the UTF-8 encoding within a try/catch block. For all the cases for which the exception was triggered, we stored the problematic headlines in an exception.csv file. We then inspected these headlines and if they were valid exceptions, we decided to remove these headlines in order to obtain a cleaned dataset suitable for our project. In total, there were about 80 headlines that we removed during the data-cleaning process.

Extracting matching news article headlines

Upon receiving predictions, we encountered a challenge in deciding between employing a threshold or relying solely on the top K results. Both methods offer their own benefits and limitations. Utilizing only a threshold approach poses the risk of either yielding numerous unrelated results or providing too few results if the threshold is set too low or too high for a particular scenario. Conversely, opting solely for the top K headlines could result in instances where K is insufficient compared to the total number of pertinent headlines, resulting in high precision but low recall.

To address the challenge of filtering matching headlines to the target headline with high recall, we adopted a hybrid approach. In this approach, we ensure that the user receives at least K results. We accomplish this by first calculating the number of headlines surpassing a certain threshold t. If the number of headlines exceeding the threshold t is greater than K, we return all headlines that meet or exceed the threshold. This guarantees that no relevant headlines are overlooked. However, if the threshold t eliminates numerous headlines and the count of headlines surpassing the threshold is less than K, we resort to returning the top K headlines. In summary, we return the maximum value

between K and the count of headlines surpassing the threshold t, ensuring that the user always receives at least K headlines.

Determining the optimal threshold for each matching algorithm

Determining the optimal threshold for each matching algorithm posed a significant challenge. To address this, we executed the model with various thresholds, selecting a random headline and observing the resulting counts of n and K, where n represents the number of headlines exceeding the specified threshold t, and K is set to 20 (top-20). This process allowed us to identify the threshold t that yielded the most desirable balance between returning either the top-20 headlines or the list of headlines that exceeded the threshold t. Once this optimal threshold was determined, we updated it in the SLO.csv file. This threshold value serves as a crucial parameter when generating results using the specific combination of tokenizer and string matching algorithm, ensuring the accuracy and relevance of the returned headlines.

Solutions

Pipeline Inputs

The pipeline to get the list of news article headlines which refer to the same target headline includes taking inputs such as the list of news article headlines from which the target list of headlines is supposed to be found, the target headline to which the list of headlines are supposed to be referred against, the service-level objectives like the maximum time that can be taken by the algorithm, the minimum precision and recall measures of the algorithm to find the matching target headlines.

Data Cleaning

When utilizing the TikToken tokenizer, it's imperative to tokenize using UTF-8 encoding. However, we encountered certain foreign language/corrupted headlines in the input list of news article headlines that couldn't be encoded in UTF-8. To address this issue, we employed the cleaning process outlined in the challenges section to rectify these corrupt headlines.

Top-K or Threshold Hybrid Approach

Opting for either a top K or threshold-based approach to filter results presents its own set of advantages and disadvantages. To ensure that no relevant sentences are overlooked while also avoiding the risk of providing too few results, we determined that employing a hybrid approach, which incorporates elements of both methods, is the most prudent course of action. This hybrid approach guarantees the provision of at least K results, striking a balance between comprehensiveness and precision in the output. More information on the hybrid approach is detailed in the challenges section.

Matching Algorithm Selection

After taking SLOs as input, our pipeline chooses and employs the best combination of tokenizer and string matching technique which accommodates the user requirements. The set of tokenizers that are considered for tokenizing the news article headlines is n-gram(*ng*), TikToken(*tt*), and simple whitespace-based tokenizer(*ws*). The set of string matching techniques that are considered to match the input list of news article headlines with the target headline is TF-IDF score-based similarity(*tfidf*), Jaccard score-based similarity(*jaccard*), and an Open-Source LLM model called Distil-RoBERTa(*llm*). Currently, we measure four performance metrics for each tokenizer and string-matching technique pair. These metrics are the time taken to compare a pair of news headlines and the precision, recall, and F1-score measures with which the matching news article headlines are found. These metrics are measured and then stored in an SLO.csv file. Using this file we find the best combination that satisfies a given user SLO requirements. These are the combinations/models documented in the SLO.csv file:

{ng_tfidf, ng_jaccard, tt_tfidf, tt_jaccard, ws_tfidf, ws_jaccard, llm}

A combination is chosen if the three SLO parameters requirements are being met by the tokenizer, string-matching technique pair. If none of the combinations in the set fulfill the requirement we fall back to the model with the highest F1 Score, which is *llm.*

Tokenization

Once the combination of tokenizer and string-matching technique selection is finalized, the chosen tokenizer is utilized to tokenize both the input news article headlines and the target headline into units suitable for the string matching techniques. These units known

as tokens serve as input to the string-matching technique. The tokenizers employed in this step include word-based n-gram (n=2), TikToken, and whitespace-based tokenizers. Each tokenizer provides a distinct method of breaking down the text into meaningful tokens for comparison.

Below is a brief description of the tokenizers employed:

1) **TikToken**: TikToken is a fast open-source Byte Pair Encoding (BPE) [4] based tokenizer developed by OpenAI which is now being used by OpenAI's Large Language Models (LLMs).
2) **n-gram**: The n-gram tokenizer selects n consecutive words as a single token, starting from index 1. It then moves to the next set of n consecutive words, starting from index 2, and continues this process iteratively. This approach allows for the creation of tokens that capture sequential word combinations within the text. We opted for n=2 for the n-gram tokenizer. Using n=1 would result in individual words being matched, lacking context from neighboring words. Conversely, selecting n=3 would require a significant number of words to be matched to constitute a match, potentially reducing the number of matches. By choosing n=2, we strike a balance, capturing meaningful sequences of words without overly restricting the matching criteria.
3) **Whitespace-based**: This basic technique just divides individual words separated by white space into separate tokens.

String Matching Technique

The string matching technique which is being used downstream to the tokenizer takes as input the tokens for the list of news article headlines and the tokens for the target headline. This step runs the string matching technique which was selected as part of the model selection and gives output as the match score for each input news article headline against the target headline. The list of matching algorithms which are being employed as part of this step are TF-IDF score-based similarity, Jaccard score-based similarity and an Open-Source LLM model called Distil-RoBERTa.

Below is a brief description of the string-matching techniques employed:

1) **TF-IDF:** TF-IDF is a popular technique used in the domain of information retrieval, plagiarism detection and document matching. It is combination of two measures - Term Frequency (TF) which measures how frequently a token occurs in a corpus of documents and Inverse Document Frequency (IDF) which measures

how rare or unique a token is across the entire corpus of documents. The TF-IDF score for a token is calculated as the product of its TF and IDF scores. The TF-IDF scores for all the tokens in a document is then represented in the form of a TF-IDF vector. Two documents are then said to match if the cosine similarity score between their TF-IDF vectors exceeds a certain threshold.

2) **Jaccard:** The Jaccard similarity coefficient, often referred to as the Jaccard index or Jaccard similarity, is a measure used to quantify the similarity between two sets. It is defined as the size of the intersection of the sets divided by the size of the union of the sets.
3) **LLM**: We used a pre-trained open-source LLM model called Distil-RoBERTa by HuggingFace trained on the WikiHow dataset. It is a small, fast and lightweight sentence-transformers model that maps sentences & paragraphs to a 768-dimensional dense vector space and can be used for tasks like clustering or semantic search. This model is intended to be used as a sentence and short paragraph encoder. Given an input text, it outputs vector embeddings that capture the semantic information of the text. We can now compare two sentences based on the cosine similarity score between their vector embeddings.

Profiling :

We conducted profiling of tokenizers and string-matching techniques to obtain performance measures for various combinations. The precision, recall, and F1-score for these combinations are summarized in the table below. Additionally, we measured the time taken to match a news headline with a target news headline, and the results are provided in microseconds in the table.

Tokenizer	String Matching Technique	Time (in micro-seconds)	Threshold	Precision	Recall	F1
tik_token	Jaccard	320	0.15	0.511434007	0.362342188	0.391147729
tik_token	TF-IDF	275	0.15	0.540312426	0.576205804	0.521191279
n_gram	Jaccard	6	0.075	0.425103979	0.211639686	0.251854284
n_gram	TF-IDF	10	0.075	0.470935391	0.279470541	0.314715247
whitespace	Jaccard	3	0.2	0.547091154	0.269535364	0.32603229
whitespace	TF-IDF	9	0.2	0.612821409	0.415876008	0.452950883
NA	LLM	4581	0.45	0.706233442	0.697379757	0.688042388

These metrics provide insights into the effectiveness and efficiency of each combination, aiding in the selection of the most suitable tokenizer and matching technique for the task at hand.

The results aren't great with the white space tokenizer because it is a basic tokenizer that tokenizes every word and fails to capture the context in which the word is used.

Utilizing TikToken tokenizer improves the results, as it is a state-of-the-art (SOTA) tokenizer that uses BPE to tokenize sentences. It is trained on a large corpus of data and is able to identify and split words into their root word followed by any prefixes/suffixes.

The results of using Jaccard as the string-matching technique aren't that great because Jaccard does not take into account the importance of the tokens. Any matching token is given the same weightage regardless of whether this is an important token in the context of the news article or a common stopword.

TF-IDF performs better than Jaccard as it accounts for the importance of individual tokens since it uses a combination of the TF and IDF scores of the token. On the other hand, TF-IDF still encounters challenges in matching tokens such as synonyms, abbreviations, and homonyms due to its inability to account for token semantics.

LLM achieves the highest F1 score because it also captures the semantic meaning of the sentences, unlike other string-matching techniques.

Miscellaneous & Future Work

Experiment with other string-matching techniques and tokenizers

Apart from the set of tokenizers and string-matching techniques being used until now as part of the project, we also want to experiment with the performance of the pipeline with other alternative tokenizers and string-matching techniques. Tokenizers like YouTokenToMe [7] is found to be performing better than TikToken in terms of runtime for smaller context lengths. As the average context length for the news article headlines are usually less than 32, YouTokenToMe might perform better for the specific use case with which we are dealing here.

Experiment with news article headlines in different languages

We also want to experiment with our pipeline for matching news article headlines in different languages and see the impact on the performance of the pipeline with respect to different languages. The research literature on tokenizing different languages suggests that specific tokenizers seem to perform better for certain languages than others. This might be challenging as now the string-matching technique selection might also have to take into consideration the language of the news article headline so that it can select an appropriate tokenizer to be used for optimal performance.

Incorporate CPU & Memory utilization SLOs

Apart from the runtime for finding matching new article headlines to the target headline and the precision/recall measures, we could also incorporate CPU and memory usage constraints as supported service level objectives for the users. These can be specifically helpful for cases where the tokenizer and string matching technique has to be run on a resource constrained environment or for cloud cost minimization purposes.

Fine-tune the LLM model

We also want to experiment with the other variants of large language models which are specifically trained on the kind of dataset that we are dealing with here so that the model has more domain-specific knowledge to perform better than general language models. We could also adopt some of the optimization techniques mentioned in DITTO [5] like domain-knowledge injection and data augmentation where we remove/change the order of the words in a news article headline.

Conclusion / Results

In conclusion, we have constructed a pipeline wherein the user submits a list of headlines, a target headline, along with specifications for the maximum response time, minimum precision, and minimum recall desired. Leveraging our profiled statistics, we identify the optimal combination of tokenizer and string-matching technique based on the user's

Service Level Objectives (SLOs). In instances where none of the combinations meet the criteria, we default to utilizing the best F1-score model, which in our case is LLM. The LLM model demonstrates impressive performance, achieving a precision of 70.6%, recall of 69.7%, and F1-score of 68.8%. This comprehensive approach ensures that users receive tailored and accurate results that align with their specific requirements.

Citations

[1] Gasparetti,Fabio. (2016). News Aggregator. UCI Machine Learning Repository. https://doi.org/10.24432/C5F61C.

[2] Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. *ArXiv*. /abs/1910.01108

[3] all-distilroberta-v1., https://huggingface.co/sentence-transformers/all-distilroberta-v1

[4] Sennrich, R., Haddow, B., & Birch, A. (2015). Neural Machine Translation of Rare Words with Subword Units. *ArXiv*. /abs/1508.07909

[5] Li, Y., Li, J., Suhara, Y., Doan, A., & Tan, W. (2020). Deep Entity Matching with Pre-Trained Language Models. *ArXiv*. https://doi.org/10.14778/3421424.3421431

[6] TikToken., https://github.com/openai/tiktoken

[7] YouTokenToMe, https://github.com/VKCOM/YouTokenToMe