**Team 1 - Prototype 3 Report:**

Gaurav’s Implementation:

* Framework that enabled you to re-rank top n documents using multiple retrieval algorithms.
* Framework for generating a result file with top n without re-rank to compare with the one generated from the re-ranked framework.
* Caching extended to all the retrieval algorithms (previously only worked on TFIDF Improved and DIRICHLET) now works on BM25 and BM25+
* Caching also extended to cache Tagme results to avoid hitting the server.
* Merged tagme and my results generation pipeline now retrieval algorithms can be used in any combination with query expansion enabled or disabled.
* Tested code on the server provided. (Installed Conda and setup testing framework)

**Generating Cache (Note: Updated in Prototype 3)**

\*\* Caching now works with tagme queries \*\*

tc\_generate\_document\_cache.py [outlines file] [paragraphs file] [no of passages to extracts from paragraph file] [use\_tagme\_enhancement]

[use\_tagme\_enhancement] : enhanced, un\_enhanced

**(top-n re-ranked)Generating trec\_eval compatible results using top n re-ranking algorithm ( Note: New in prototype 3 )**

python tc\_rerank\_document\_framework.py [outlines file] [paragraphs file] [output file] [primary\_retrieval\_algorithm] [re-ranking\_algorithm] [cache] [no\_of\_results\_to\_re-rank] [no of passages to extract] [use\_tagme\_enhancement]

The aforementoned arguments can take the following value:

[primary\_retrieval\_algorithm]: BM25, BM25+, TFIDFIMPROVED

[re-ranking\_algorithm] : DIRICHLET

[cache] : no\_cache, cache ( Note 'cache' only works if tc\_generate\_document\_cache.py is run first on same number of passages )

[no\_of\_results\_to\_re-rank] : an integer (less than no of passages being extracted)

[no of passages to extract] : an integer

[use\_tagme\_enhancement] : enhanced, un\_enhanced

Sample Run Statement:

Note: For a quick result evaluation use the un\_enhanced but if you want to use enhanced generate the cache first.

python tc\_rerank\_document\_framework.py all.test200.cbor.outlines release-v1.4.paragraphs output.top500reranked.run TFIDFIMPROVED DIRICHLET no\_cache 500 50000 un\_enhanced

**(top-n not re-ranked)Generating trec\_eval compatible results top - n without reranking ( Note: New in prototype 3 )**

\*\*\*\* Note:

This is only to compare the top n implementation as results from top n might be a little bit lower than the the entire thing this is a good way to generate a top n only results file without re ranking to compare it with the re-ranked implementation

\*\*\*\*\*

python tc\_rerank\_document\_framework.py [outlines file] [paragraphs file] [output file] [ranking\_function] [cache] [top\_n\_only] [no of passages to extract] [use\_tagme\_enhancement]

The aforementoned arguments can take the following value:

[ranking\_function]: BM25, BM25+, TFIDFIMPROVED, DIRICHLET

[cache] : no\_cache, cache ( Note 'cache' only works if tc\_generate\_document\_cache.py is run first on same number of passages )

[top\_n\_results\_only] : an integer (less than no of passages being extracted)

[no of passages to extract] : an integer

[use\_tagme\_enhancement] : enhanced, un\_enhanced

Sample Run Statement:

Note: For a quick result evaluation use the un\_enhanced but if you want to use enhanced generate the cache first.

python tc\_rerank\_document\_framework.py all.test200.cbor.outlines release-v1.4.paragraphs output.top500notreranked.run DIRICHLET no\_cache 500 50000 un\_enhanced

**Generating trec\_eval compatible results file all results (Note: Updated in Prototype 3)**

tc\_generate\_document.py [outlines file] [paragraphs file] [output file] [ranking function] [cache] [no of passages to extract] [use\_tagme\_enhancement]

The aforementoned arguments can take the following value:

[ranking function] : BM25, BM25+, TFIDFIMPROVED, DIRICHLET

[cache] : no\_cache, cache ( Note 'cache' only works if tc\_generate\_document\_cache.py is run first on same number of passages )

[no of passages to extract] : an integer

[use\_tagme\_enhancement] : enhanced, un\_enhanced

Shilpa’s Implementation

1. Implemented Rocchio algorithm based on Relevance feedback for query and paragraph word expansion.
2. Merged Gaurav's retrieval methods to use Tagme enhanced data,
3. Synced Colin's clustering with Tagme implementation.
4. Merged Gaurav’s cache and re-ranking implementation to use with entity-linking. Performed caching to use the top-100 paragraphs for entity-linking and then re-ranked the top-100 to generate better results.

**How to run the code:**

1. You can use the bash script entitylinking.run.sh

Where $1 – outlines file

$2 – paragraphs file

$3 – number of passages to extract

$4 – enhanced or un\_enhanced

$5 - outlines file (same as above)

$6 - paragraphs file (same as above)

$7 – output file

$8 – retrieval algorithm (BM25, Dirichlet, TFIDFIMPROVED)

$9 – cache or no\_cache

$10 - number of passages to extract (same as above)

$11 – qrels file

$12 – output file (same as above)

1. Separate code run **–**

Uses Gaurav’s implementation for caching the Tagme generated data.

**tc\_generate\_document\_cache.py** for generating cache of the enhanced queries and passages. If this code is run first, then we can avoid load on Tagme server.

This file takes three input parameters:

Outline file, paragraph file and number of passages you want to extract.

Example run:

python tc\_generate\_document\_cache.py all.test200.cbor.outlines release-v1.4.paragraphs 50000

**tc\_generate\_entitylink\_rm\_cache\_results.py** for performing entity linking with query expansion using rocchio relevance model. This will generate output file used in evaluation framework. This file takes 6 parameters:

outlines file, paragraph file, output file, retrieval algorithm, use\_cache and number of passages to extract

Example run:

python tc\_generate\_entitylink\_rm\_cache\_results.py all.test200.cbor.outlines release-v1.4.paragraphs output.run BM25 cache 50000

**eval\_framework.py** This is for evaluating the results. This takes two parameters:

qrels file and output file

Example run:

python eval\_framework.py all.test200.cbor.hierarchical.qrels output.run