Source recommendation system for fact-checking texts

JOY GHOSH

SAURABH DEOTALE

Background Information

- Lot of sources publishing news articles
  - Difficult to check the factual correctness
- Social media adds a new dimension in spreading false news
- Manual fact checking is time consuming
- Automatic related source recommendation is required
  - Users will be able to verify the news from the sources he trusts

Problem characterization

- Design a system that recommends different sources which have common context with the unverified article from user.
- The problem is to find a solution that can provide sources from datasets with similar context within reasonable time.
- Crucial to select a strategy for ranking documents that not only can be performed fast enough but can also provide reasonable accuracy

Trade-offs

- Discarded 50% of keywords with lower score (from both dataset as well as input file)
- Combine two rudimentary techniques instead of one complex algorithm to reduce computation time
- Reducing the result size by modifying score calculation to avoid memory issues and execution delay

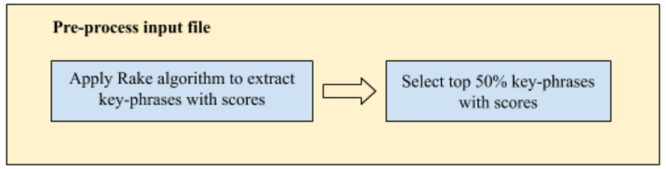
Methodology

There are 3 phases

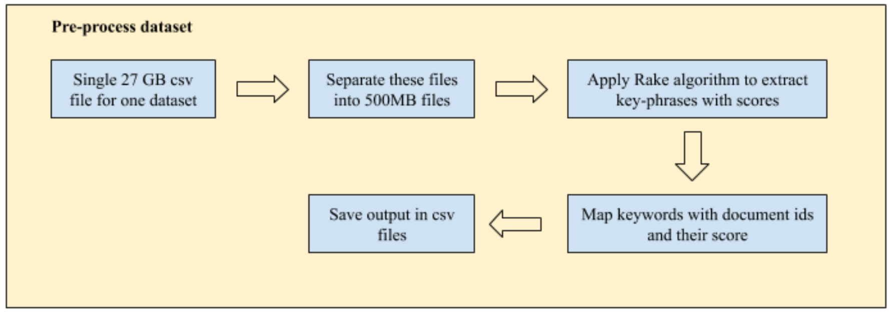
- Preprocessing: Extract key-phrases from the dataset and save them with metadata
- Key-phrase matching: Extract key-phrases from the input file and match them against pre-processed phrases
  - Calculate scores for each document based on common phrases
- Postprocessing: Generate ranked articles and output them with similarity score as well as labeled information from the dataset

Pre-processing

- For the dataset split file into 500 MB chunks
- Extract key-phrases using RAKE
- Select 50% of ranked phrases

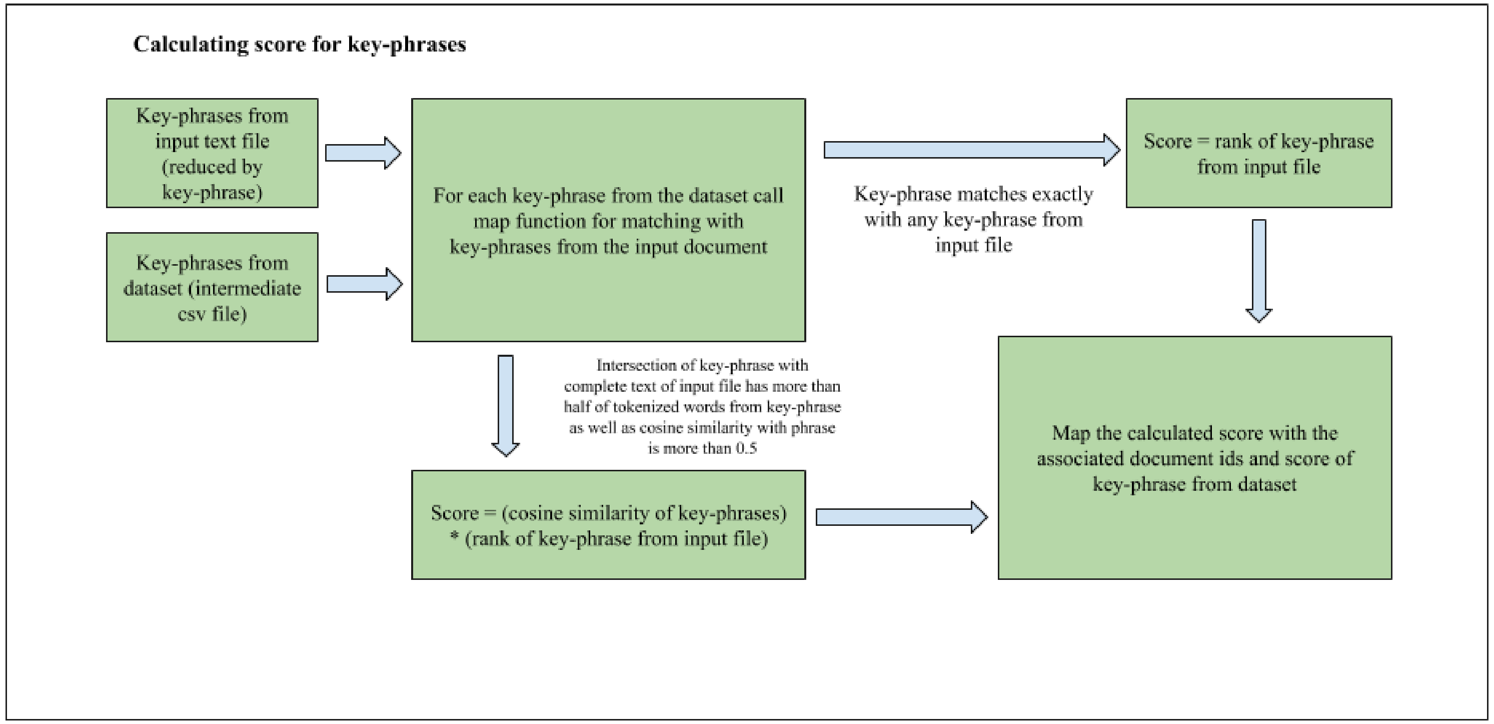


Pre-process input file
Apply Rake algorithm to extract key-phrases with scores
Select top 50% key-phrases with scores



Pre-process dataset
Single 27 GB csv file for one dataset
Separate these files into 500MB files
Apply Rake algorithm to extract key-phrases with scores
Map keywords with document ids and their score
Save output in csv files

Ranking documents


Calculating score for key-phrases
Key-phrases from input text file (reduced by key-phrase)
Key-phrases from dataset (intermediate csv file)
For each key-phrase from the dataset call map function for matching with key-phrases from the input document
Key-phrase matches exactly with any key-phrase from input file
Score = rank of key-phrase from input file
Intersection of key-phrase with complete text of input file has more than half of tokenized words from key-phrase as well as cosine similarity with phrase is more than 0.5
Score = (cosine similarity of key-phrases) * (rank of key-phrase from input file)
Map the calculated score with the associated document ids and score of key-phrase from dataset

Post-processing

- Find the similarity scores for ranked documents
- Extract labeling for the identified documents from the dataset

Performance Benchmarks

- 30 worker machines
- 1 core per executor
- 2 executors per worker
- Driver and Executor memory - 2GB

Performance Benchmarks

Average time for running the jobs -

Job	Time
Partitioning 27G file into 58 512MB file	32 min
Extracting keywords from 512MB file	1.2 - 1.4 min
Extracting similar articles using keywords from articles of 512MB file	2.3 min
Extracting similar articles using keywords from articles of 1GB file	3.5 min
Extracting similar articles using keywords from articles of 2GB file	6 min

Performance Benchmarks

Identifying threshold for finding similarity between documents

% Text similar in the input file	Recommended article containing the original text
100% text	Yes
50% text	Yes
30% text	Yes
25% text	Yes
<15% text	No

Key Innovations

- Combination of two rudimentary techniques for similarity instead of a single complex algorithm
- Partitioning dataset into multiple files for batch processing to overcome pyspark limitation
- Introduce thresholds and checks for filtering out data with less impact