# Simple Search Engine in Spark

**Assignment**

Palyonov Vadim, Telegram: @vAD$_l v$
Konyshev Yan, Telegram: @DorakAlba
Gorbunov Arseniy, Telegram: @ArsenyGorbunov

GitHub Repository

https://github.com/vADLv/BD_SearchEngine Innopolis University

07.10.2019

# Contents

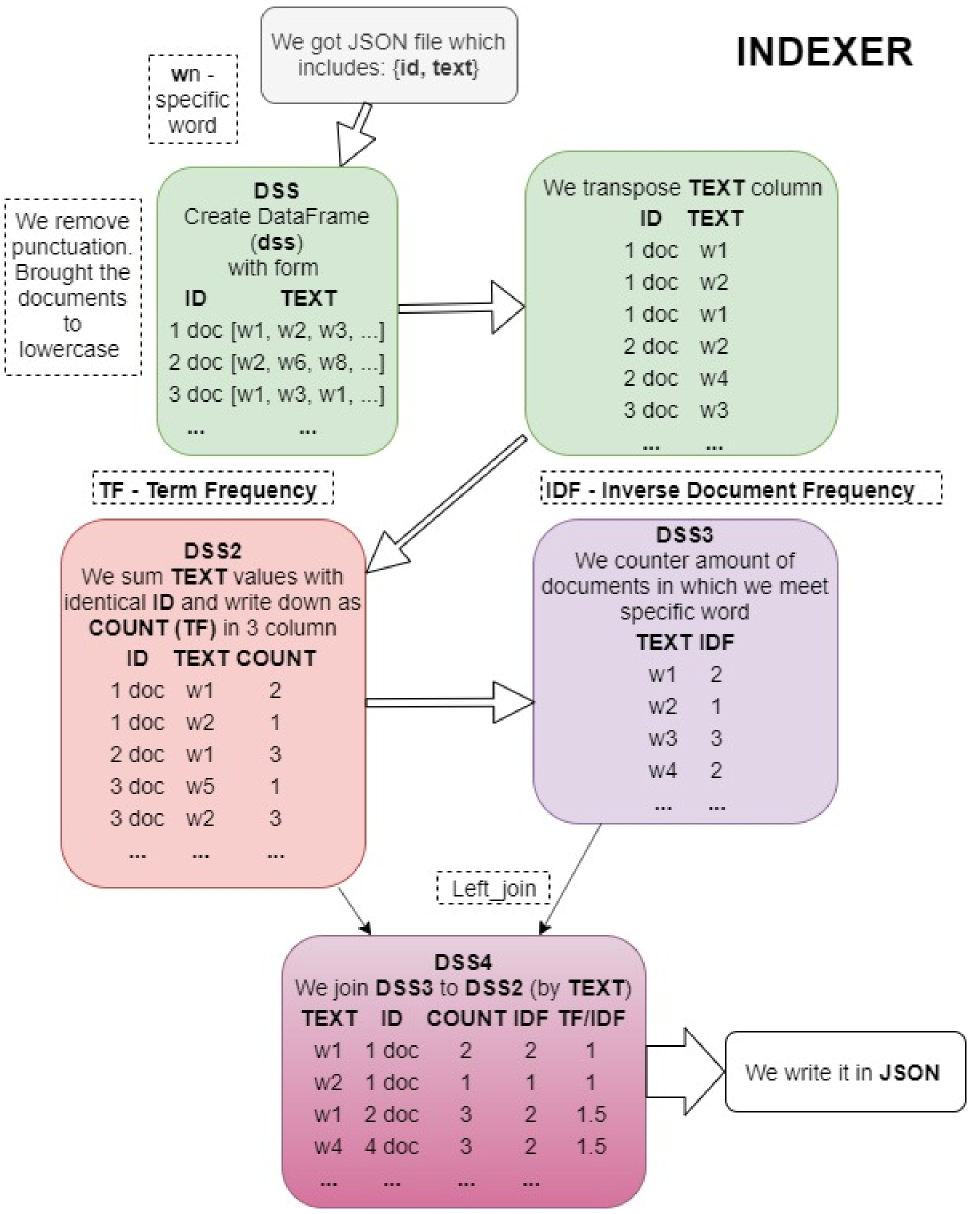
## 1. Objective

During this work, we had to create a search engine and execute it on the university Hadoop cluster. Polish programming in Scalla and work with Spark. Train optimization techniques and usage of Git.
These technologies allow as to parallelize computation on Big Data on the cluster by using DataFrame, RDD, DataSet types.
As a result, we have got a significant profit in time by using the cluster (indexer on full EnWikiMedium - 15min, ranker - 5 min) instead of the local machine (indexer on 1 file - 4min, ranker - 3 min).
The indexer module represents all documents as a TF-IDF vector and saves the result to JSON files. Ranker module reads that files as well as a search query and provides as a list of result documents descending ordered by estimated relevance. Both Indexer and Ranker modules were written in Spark Scala to use all advantages of parallelism in Big Data.
MAP module helps to evaluate the results and accuracy of the rank algorithms. Also, it gives a comparison Basic Vector Space Model and BM25 algorithms. MAP was written in Python to quickly gather and visualize all results. The following sections provide details how it was implemented

# 2. Implementation

## Indexer

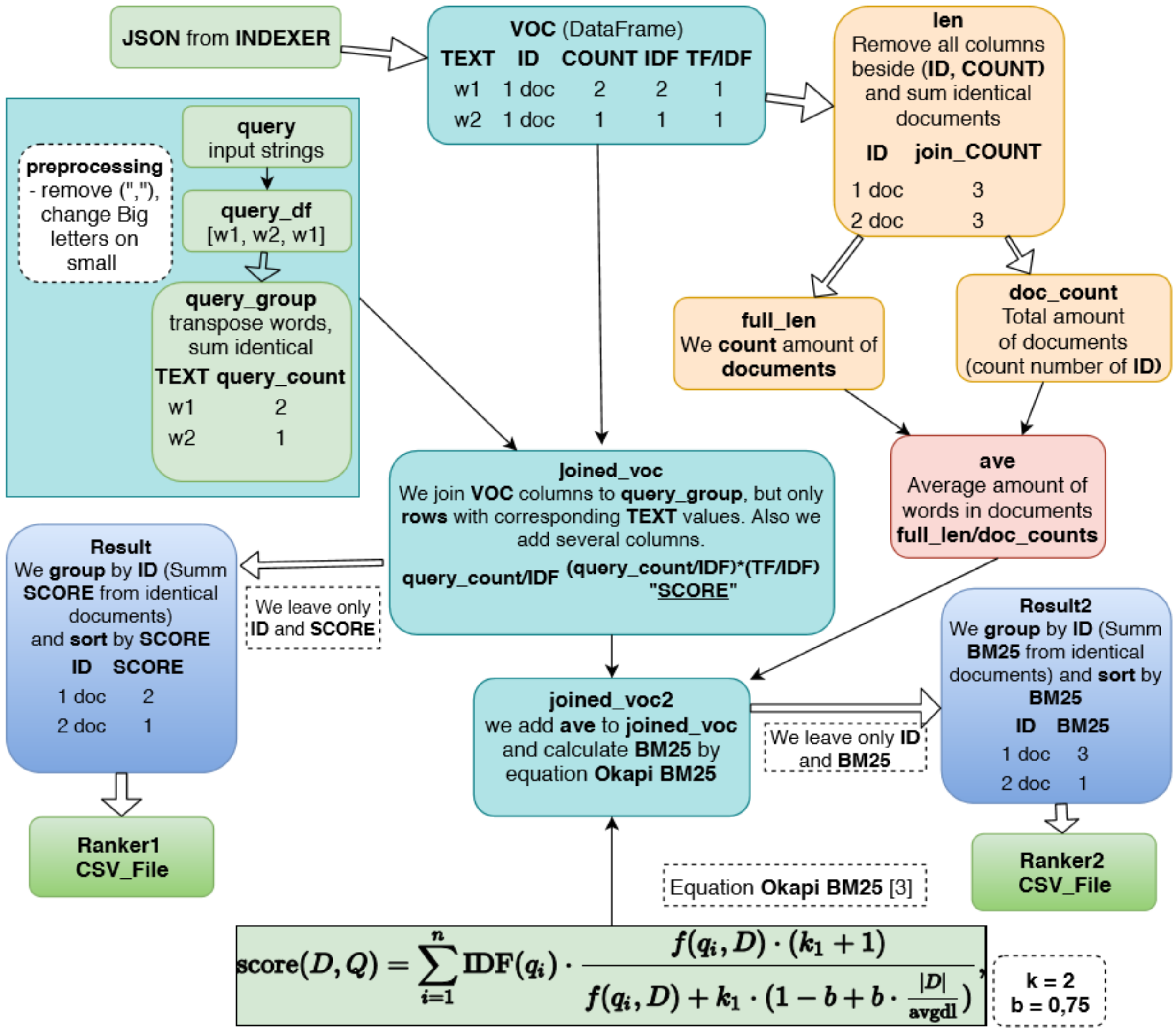
Our first objective was to represent documents in vector format and evaluate it with TF/IDF Vector Space Model.



```
spark-submit –deploy-mode cluster –master yarn
–class Indexer path/to/indexer/scala.jar
/nigeria/BD_assignment_1/ /nigeria/BD_assignment_1/indexer_result
```

## Ranker

Second module reads results from previous one and converts it to dataframe type to futher implement two ranker alghoritms.
First algorithm is a simple Basic Vector Space Model, where we multiply tf/idf vaectors from each documents with the search query.


JSON from INDEXER
VOC (DataFrame)
TEXT ID COUNT IDF TF/IDF
w1 1 doc 2 2 1
w2 1 doc 1 1 1
len
Remove all columns beside (ID, COUNT) and sum identical documents
ID join_COUNT
1 doc 3
2 doc 3
query
input strings
preprocessing
- remove (","), change Big letters on small
query_df
[w1, w2, w1]
query_group
transpose words, sum identical
TEXT query_count
w1 2
w2 1
full_len
We count amount of documents
doc_count
Total amount of documents (count number of ID)
ave
Average amount of words in documents
full_len/doc_counts
Joined_voc
We join VOC columns to query_group, but only rows with corresponding TEXT values. Also we add several columns.
query_count/IDF
(query_count/IDF)*(TF/IDF) "SCORE"
Result
We group by ID (Summ SCORE from identical documents) and sort by SCORE
ID SCORE
1 doc 2
2 doc 1
We leave only ID and SCORE
joined_voc2
we add ave to joined_voc and calculate BM25 by equation Okapi BM25
We leave only ID and BM25
Result2
We group by ID (Summ BM25 from identical documents) and sort by BM25
ID BM25
1 doc 3
2 doc 1
Ranker1 CSV_File
Ranker2 CSV_File
Equation Okapi BM25 [3]
$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i,D)\cdot(k_1+1)}{f(q_i,D)+k_1\cdot\left(1-b+b\cdot\frac{|D|}{\text{avgdl}}\right)},$
k = 2
b = 0,75

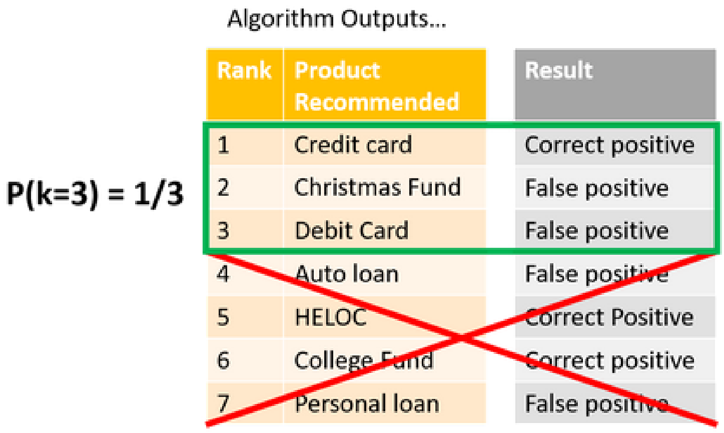
spark-submit –deploy-mode cluster –master yarn –class Ranker
path/to/ranker/scala.jar /nigeria/BD_assignment_1/indexer_result
/nigeria/BD_assignment_1/ranker_result "GOOGLE"

## Map

In order to assess the quality of the search algorithms, we used the Mean Average Precision (IDA) metric. This metric is a good choice for quality assessment if the algorithm produces ranked ordering of items by their priority, where every item is either relevant or irrelevant [1].
The metric is calculated based on averaging Average Precision (AP) for all search queries. AP calculated by the next formula:

$$AP = \frac{1}{N_{rel}} \sum_{k=1}^{N_l} P(k) \cdot \mathrm{rel}(k)$$

P(k) – precision of a cutoff k:



Algorithm Outputs...

P(k=3) = 1/3

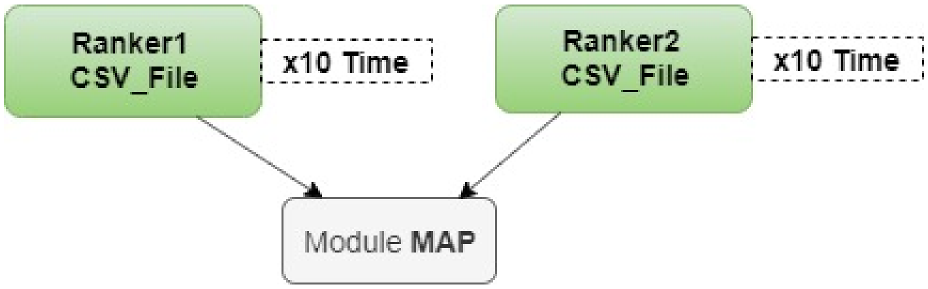
| Rank | Product Recommended | Result |
|---|---|---|
| 1 | Credit card | Correct positive |
| 2 | Christmas Fund | False positive |
| 3 | Debit Card | False positive |
| 4 | Auto loan | False positive |
| 5 | HELOC | Correct Positive |
| 6 | College Fund | Correct positive |
| 7 | Personal loan | False positive |

Pic. 1. Example of calculation of P(k)

AP rewards us for having a lot of relevant recommendations and for putting the most likely correct result at the top (if we put our best guesses at the top we can't decrease AP score by tacking on more).
The program was written in python. The algorithm automatically opens the Wikipedia site and offers to evaluate its relevance according to the query. Relevance assessment is subjective.
The queries were chosen in such a way that they were diverse, i.e. included known/unknown to the General public and long/short queries from different areas. In total, we used 11 queries and 3 recommendation for each. The following queries were selected to evaluate the quality of the model:

- Ananas Hamburg
- Anarchism today
- Beethoven - 7th Symphony
- Butterfly effect film
- demographics of russia 1917
- how often camels drink water in Sahara
- Machine Learning and Neural Networks
- Master student Innopolis University
- Motion Design Disney vs pixar
- new year traditions in different countries
- Spark Scala Hadoop
- Zamira zouker

The results of the rankers are sent to the MAP module, which gives their assessment:



# 3. Results

Rusults
http://10.90.138.32:9870/explorer.html/nigeria/BD_assignment_1
GitHub Repository
https://github.com/vADLv/BD_SearchEngine

## Analyze

We compared MAP metric of Simple algorithm and BM25

| Indexer | MAP |
|---|---|
| Simple | 0.514 |
| BM25 | 0416 |

For our Wikipedia queries Simple indexer gives a better result than Bm25.
The search algorithm does not take into account the sequence of words in the sentence. Also, the algorithms do not take into account synonyms. A subjective assessment of relevance leads to a subjective assessment of the metric.

## Participation

Work was conducted in team with all people involved But we shared rensponsibilities between different parts of work:

- Setting GIT infrastructure and work with Cluster - Vadim
- Files Diagrams - Yan
- Work under Indexer module - Yan, Vadim, Arseniy
- Module Ranker - Arseniy (1th algorithm), Vadim (2th algorithm)
- Testing Final module on locale machine, on local and university cluster - Vadim
- MAP module - Arseniy
- Report, LaTeX - Yan

## 4. Conclusion

Program shows a valid result and we can use it for search documents in Wikipedia. Module structure allows us to launch Indexer module only once and separate it from our Ranker/ Search queries.
Also We can further increase our result accuracy by:
- improve preprocessing, for example delete nonsense words etc
- improve our algorithms, for example choose the different method of IDF computation
- filter out results that don't include all word from search query

## 5. References

1. http://sdsawtelle.github.io/blog/output/mean-average-precision-MAP-for-recommender-systems.html
2. newline Spark SQL guide. JSON files
3. Ranker BM25
4. Spark programming guide
5. For examples, troubleshooting
6. Patterns for regular expressions
7. Mean average precision