Finding Similar Items

Sainey Manga - ID 943874

University of Milan, Data Science and Economics

sainey.manga@studenti.unimi.it

January 25, 2022

I/We declare that this material, which I/We now submit for assessment, is entirely my/our own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my/our work. I/We understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me/us or any other person for assessment on this or any other course of study.

I. Introduction

With advances in data collection techniques and extended storage media volume of stored data is growing constantly. As an example, all pages stored in internet can be noted. With the increasing volume of data stored, it is not possible retrieving them with conventional techniques easily. Therefore, methods for processing massive data are required. A fundamental data-mining problem is to examine data for "similar" items. The problem of finding textually similar documents is one of the problems of finding similar items.

The aim of this project is to implement a detector of pairs of similar items, analyzing the stack sample dataset published on Kaggle. The project explores Local sensitive hashing approach design for Jaccard distance.

II. DATA DESCRIPTION AND PREPROCESSING

The data set stacksample downloaded from Kaggle where it is published and released under the CC-BY-SA 3.0 license. It is downloaded during code execution via the Kaggle API. The entire data set consists of three different tables, namely:

- Questions which contains the title, body, creation date, closed date (if applicable), score, and owner ID for all non-deleted Stack Overflow questions whose Id is a multiple of 10.
- Answers which contains the body, creation date, score, and owner ID for each of the answers to these questions. The Parent-Id column links back to the Questions table.
- Tags which contains the tags on each of these questions.

In this this project, we are only interested on only the table "Questions" where we considered the attribute "body" as the set of documents or items we will mined for similarity. A function was implemented using spark to read and curate only the body and id of the required table, other attributes were filtered out from the table.

After loading the data set, a function called **clean_text()** was defined to remove all characters, html links, numbers and other unwanted features found on the text of the body attribute.

The first 10 rows of cleaned data are shown below:

Id	Body
80	ive written a dat...
90	are there any rea...
120	has anyone got ex...
180	this is something...
260	i have a little g...
330	i am working on a...
470	ive been writing ...
580	i wonder how you ...
650	i would like the ...
810	im trying to main...

Figure 1: *First ten rows of the data set*

A tokenization process was caried on the cleaned data set to distinctly separate the text into pieces (tokens). This was done with the help of the RegexTokenizer function in pyspark.ml.feature.

To remove the low-level information from our text, we also removed stop words from the corpus so that we can focus on the more important words in the text. We fed the tokens generated from the previous step onto the function StopWordsRemover in pyspark.ml.feature.

Id	Body	tokens	removed_st_wrd
80	ive written a dat...	[ive, written, a,...	[ive, written, da...
90	are there any rea...	[are, there, any,...	[really, good, tu...
120	has anyone got ex...	[has, anyone, got...	[anyone, got, exp...
180	this is something...	[this, is, someth...	[something, ive, ...
260	i have a little g...	[i, have, a, litt...	[little, game, wr...
330	i am working on a...	[i, am, working, ...	[working, collect...
470	ive been writing ...	[ive, been, writi...	[ive, writing, we...
580	i wonder how you ...	[i, wonder, how, ...	[wonder, guys, ma...
650	i would like the ...	[i, would, like, ...	[like, version, p...
810	im trying to main...	[im, trying, to, ...	[im, trying, main...

Figure 2: *Tokenized with removed stop words data*

III. ALGORITHMS AND IMPLEMENTATION

i. Jaccard Similarity

In this project, the Jaccard similarity algorithm was implemented to find the pair of questions that are similar in the data set. The Jaccard similarity and Jaccard distance are differentiated by the mathematical definition. The similarity is subtracted from one (1) to have the distance. If the similarity approach is to be considered, then more similar items have a score closer to one and a score closer to zero if otherwise.

The Jaccard similarity is defined as:

$$SIM(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{1}$$

ii. Local Sensitive Hashing (LSH) for Minhashing

One general approach to LSH is to "hash" items several times, in such a way that similar items are more likely to be hashed to the same bucket than dissimilar items are. This is applied in large data sets to reduce the computational costs, so that items that are more likely to be similar are considered.

IV. EXPERIMENTS AND RESULTS

The experiments carried out on this project are done through the help of spark. Firstly, a pipeline model was set to implement two very key functions. The function HashingTF from pyspark which maps a sequence of terms to their term frequencies using the hashing trick. It takes in as input the text where stop words have been removed and outputs a column of vectors to be used for hashing. Finally, the LSH class for Jaccard distance was also implemented in the pipeline model through the MinHashLSH function from pyspark.

The function takes in as input the sparse vector generated from the previous step and outputs the hashes which we shall use for the implementation of our Jaccard distance.

Id	Body	tokens	removed_st_wrd	vectors	hash
80	ive written a dat...	[ive, written, a,...	[ive, written, da...	(262144,[3564,461...	[[4.7238581E7]]
90	are there any rea...	[are, there, any,...	[really, good, tu...	(262144,[62427,68...	[[3.9862902E7]]
120	has anyone got ex...	[has, anyone, got...	[anyone, got, exp...	(262144,[386,3564...	[[1.8689312E7]]
180	this is something...	[this, is, soneth...	[something, ive, ...	(262144,[10345,37...	[[4.027477E7]]
260	i have a little g...	[i, have, a, litt...	[little, game, wr...	(262144,[5352,126...	[[1.2793449E7]]
330	i am working on a...	[i, am, working, ...	[working, collect...	(262144,[8538,978...	[[4.1105192E7]]
470	ive been writing ...	[ive, been, writi...	[ive, writing, we...	(262144,[9781,158...	[[9.3702936E7]]
580	i wonder how you ...	[i, wonder, how, ...	[wonder, guys, ma...	(262144,[2306,298...	[[1.3542467E7]]
650	i would like the ...	[i, would, like, ...	[like, version, p...	(262144,[11680,12...	[[7.2622284E7]]
810	im trying to main...	[im, trying, to, ...	[im, trying, main...	(262144,[2259,126...	[[1.1757093E7]]

Figure 3: *Data showing the vectors and hashes*

To limit computational time, we implemented the final model on a sample of 40,000 from the data set. This is done after conducting the LSH minhashing. The approach is indeed scalable to large data sets. All rows were filtered out to take only the rows that have questions with at least a word.

Finally, to implement the Jaccard distance model, the approxSimilarityJoin function in pyspark was invoked with a negative one [-1] to make it a distance. It takes as input the hashed data, where the data set has been divided into two parts with respect to the ids. The results are filtered to show only the pairs that have a similarity score below 0.7. This is to show only a reasonable number of pairs of similar items.

id_A	id_B	distCol
30800	513330	0.6929133858267716
103460	1251620	0.6666666666666667
144530	1712670	0.6666666666666667
144530	155780	0.6666666666666667
154630	343230	0.625
218450	727180	0.6732673267326732
290080	654450	0.65625
344460	344550	0.6415094339622642
406700	844410	0.6923076923076923
446500	446600	0.5714285714285714
503310	835280	0.375
606820	623990	0.625
612820	634630	0.5
741900	994270	0.5714285714285714
897770	905410	0.5061728395061729
1006060	676920	0.5555555555555556
1107670	1107820	0.5866666666666667
1192680	1479100	0.6666666666666667
1257910	1544880	0.6944444444444444
1287340	647750	0.625

Figure 4: *Pairs of similar items*

The table 4 shows two sets of ids representing to various texts with their corresponding similarity score. The closer the score is to zero, the more similar

the texts are. We can see that the ids 503310 and 835280 are the most similar questions in the data set. This is followed by the score of 0.5.
We show below a sample of similar text as provided by the solution of the model implemented.

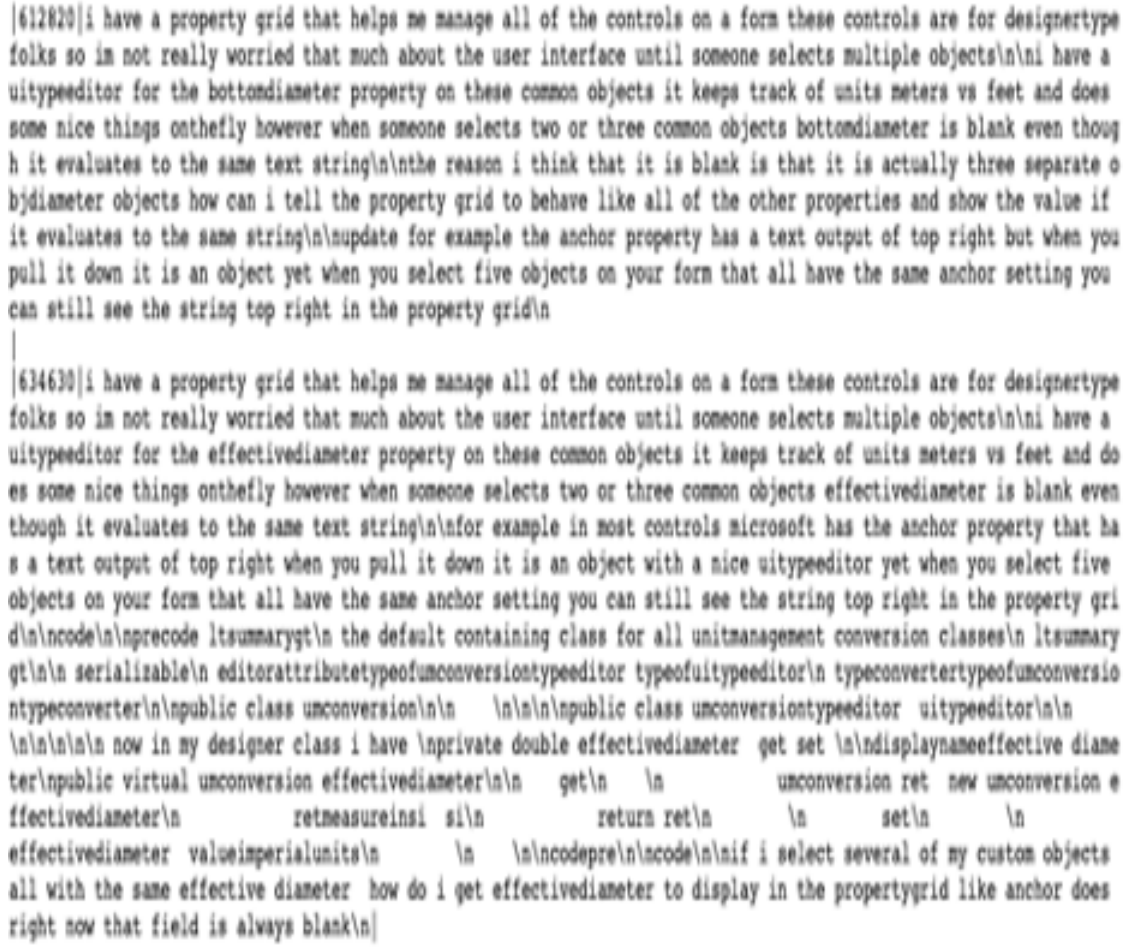

Figure 5: *IDs with score 0.5*

Id	Body
503310	is it possible to map an enum as a string using fluent nhibernate\n
835280	its possible to map a view using fluent nhibernate if so how\n

Figure 6: *IDs with score 0.375*

V. CONCLUSION

So far what we have seen in the two samples of results is that a pair could be classified similar, but the context of the two texts would be entirely different. For example, our most similar pair from the data set asked about two different things, but the questions have a similar format and a good number of similar words. With Jaccard distance, we know pairs with more common words would appear to be similar.

Summarily, the project utilized LSH class of Jaccard distance to find pairs of similar items from the StackSample data set. During model execution, the running time increased as we increased the size of the input data.