Finding Similar Questions
for Stack Overflow

Junqian Zhang - 944566

University of Milan
Data Science and Economics

Abstract. This project aims to use Locality Sensitive Hashing to provide an efficient solution for finding similar question on Stack Overflow. It carries out experiments with traditional exhaustive search and Locality Sensitive Hashing by PySpark respectively, over a series of dataset with different size. Finally it compares the computation time and similar pairs detected on both methods. Through these experiments LSH proves its advantage in finding similar items in large-scale dataset.

Keywords: Massive Data · PySpark · Finding Similar Items · Locality Sensitive Hashing.

1 Introduction

This work aims to explore the solutions of similarity analysis on large-scale question set from Stack Overflow over Google Colaboratory. In traditional exhaustive way, we are able to compare the items one by one. But as the number of items increases to million or even billion level, the simple similarity comparison needs approximately $\frac{n^2}{2}$ times computation, implying that as the size of dataset grows, the time cost increase in an exponential way, consuming too much time. However, Locality Sensitive Hashing (LSH) is able to perform approximate near neighbor search combined with PySpark in a smart way and thus find similar questions in the large dataset in an more time-saving way.

In order to find similar questions one Stack Overflow in this work, all the questions are represented by sets of token words, and Jaccard similarity are used to compare the word sets. A series of experiments are held on datasets with different size with both methods to prove the advantage of LSH.

2 Data

2.1 Data Description

The dataset used for the analysis is StackSample: 10% of Stack Overflow Q&A, available on Kaggle. It contains 3 tables, but only *Questions.csv* is analyzed here. *Questions.csv* is 1.92 GB with 1,264,216 samples and is composed of 6 variables, which are:

- **Id**: unique Id for each question
- **Title**: the title of each questions
- **Body**: the content of the questions
- **Creation Date**
- **Closed Date**
- **Score**
- **Owner Id**

Body are regarded as the document items analyzed in this work and **Id** is kept as index.

2.2 Data Preprocessing

Before performing similarity analysis, it is necessary to clean the raw data. Firstly, I checked the missing values and fortunately, there is no missing ones. Secondly, to gain a better representation of question body, I remove the all the HTML tagging, such as "$< p >$", and replace all the non alphabetic character, such as numbers, with white space. The next step is to take tokens from the document body and then remove the stop words from the token lists to reduce the noise on similarity comparison. **Table 1** represents the first 5 example documents after data preprocessing.

Id	Body	Tokens	No_StopWords	No_Duplicates
80	I've writt...	[i, ve,...	[ve, written,...	[ve, written,...
90	Are there ...	[are, there,...	[really, good,...	[really, good,...
120	Has anyone...	[has, anyone,...	[anyone, got,...	[anyone, got,...
180	This is so...	[this, is,...	[something, ve,...	[something, ve,...
260	I have a l...	[i, have,...	[little, game,...	[little, game,...

Table 1: Summary Examples of Data Preprocessing

2.3 Feature Extraction

To remove the tokens which provide less information for document and also reduce the number of tokens for computation, **Term Frequency** (TF) is adopted for feature extraction. *TF(t,d)* is the number of times that term *t* appears in the documents *d*. The number of features retained, i.e., the number of rows in the characteristic matrix is set as **1,024**. Another useful method for feature extraction Term Frequency - Inverse Document Frequency (TF-IDF) is not concerned here because, this work will run a series of experiments on different size of dataset, so it makes no sense to consider the number of documents in feature selection.

3 Methodology

3.1 Locality-Sensitive Hashing

The main algorithm used for finding similar items in this work is **Locality Sensitive Hashing** (LSH), which performs approximate search instead of exhaustive search. After the data preprocessing and feature extraction, we get a sparse characteristic matrix M in which rows are 1,024 features retained, and columns are the all the documents for comparison. The first issue is that the matrix is sparse and there are a large number of rows. To solve it and thus represent large documents into a smaller matrix, minhash signature $[h_1(S), ..., h_n(S)]$ is used to reconstruct the matrix based on the connection that $P(h(S) = h(T)) = SIM(S,T)$. This remarkable connection conveys the idea that a hash function returns the same values when the two documents have high similarity. As a result, all the similar documents will have similar minhash signatures.

Although the signature matrix has less rows, it has the same number of columns as the original characteristic matrix. However it is time consuming to compute the similarity of each pair as the size of dataset grows to million or billion, since it produces an at best complexity of $O(n^2)$. So besides trying to get the exact similar pairs, we can focus on the pairs which are more likely to be similar. LSH makes it possible to hash items several times in such a way that similar documents are more likely to be hashed to the same bucket than dissimilar documents. And the pairs that are hashed to the same bucket for any of the hashings are regarded as *candidate pairs*. In this way, our attention is moved from each pairs in the dataset to only candidate pairs and therefore we will compute similarity less times and achieve the task more quickly.

3.2 Similarity Metrics

The similarity metrics employed to compare two question document is **Jaccard Similarity**. The Jaccard similarity of any two token sets of questions S and T is calculated as equation (1). $|S \cap T|$ means that the number of the tokens share by both set S and set T while $|S \cup T|$ represents the the number of all the tokens who appear in either set S or set T. **Jaccard Distance** is defined as equation (2), ranging from zero to one. The larger the distance is, the lower possible the two set are similar. When the *d(S,T)* is zero, representing that S and T are exactly the same, while *d(S,T)* is one, representing that S and T are totally different. The threshold to select similar items in this work is **0.5**, meaning that if the Jaccard distance between two sets are smaller than 0.5, they are picked as similar items.

$$SIM(S,T) = \frac{|S \cap T|}{|S \cup T|} \tag{1}$$

$$d(S,T) = 1 - SIM(S,T) \tag{2}$$

4 Experiments and Results

The experiments are performed on a sequence size of dataset to see the performance, ranging from 2,000 to 44,000, 2,000 as steps, **22** sample sizes in total. Two different ways are used for finding similar documents. One uses the traditional exhaustive search , and all the computation are performed without *PySpark*, where *sklean* package is used for feature extraction and Jaccard similarity is calculated in Python environment . The other one uses LSH with *PySpark*.

4.1 Computation Time

In the case of the traditional exhaustive search which is able to find the exact similar pairs, limited to the time, the experiment is carried out on only on dataset of size 2,000 and 4,000. Since there is no any hashing, all the pairs in the dataset are compared with the original characteristic matrix. In the experiment of 2,000 samples, there are **1,999,000** pairs, and it takes **745.45** (12 minutes) seconds to finish the feature extraction and the similarity computation for ALL pairs. When it comes to the experiment of 4,000 samples, double of last experiment, there are **7,998,000** pairs, around **four** times of last experiment, and takes **2991.04** seconds (49 minutes), almost more than **four** times of last experiment.

Compared to exhaustive search, LSH brings a far more efficient way to finish the computation. As depicted in **Fig.1**, both experiment under sample size 2,000 and 4,000 with LSH spend less time than that with exhaustive search. Obviously, as the size of data size increases 2,000 at each step, the computation time cost increases by no more than 50 seconds, unlike the exponential way in the exhaustive search.

The computation time summarized in **Table 2** and **Fig.2** imply that when the data size reaches million or billion level, LSH with PySpark can achieve the task in a more time saving way.

4.2 Similar Items Found

Spending great much time in Jaccard Similarity, the two experiments of exhaustive search do not find any similar pairs who have Jaccard distance lower than threshold 0.5.

The first three experiments with LSH do not find any similar pairs either. In the experiments with data size from 8,000 to 16,000, it successfully find the first similar pair with Jaccard distance 0.47. The experiment with 18,000 find two similar pairs. As sample size increase from 20,000 to 26,000, each time gets 4 pairs. **Table 3** lists the similar pairs detected with the time order.

Limited to the resource, it is impossible to calculate the *False Negatives* which are the pairs not picked as similar ones but in fact are similar ones. Also, it is

Sample Size	LSH	Exhaustive Search
2,000	10.13	745.45
4,000	8.84	2,991.04
6,000	15.32	-
8,000	21.95	-
10,000	30.19	-
12,000	41.15	-
14,000	52.71	-
16,000	68.63	-
18,000	85.90	-
20,000	102.00	-
22,000	119.07	-
24,000	143.22	-
26,000	161.32	-
28,000	187.33	-
30,000	209.76	-
32,000	236.00	-
34,000	269.23	-
36,000	299.59	-
38,000	332.19	-
40,000	358.62	-
42,000	400.39	-
44,000	437.55	-

Table 2: Computation Time Summary

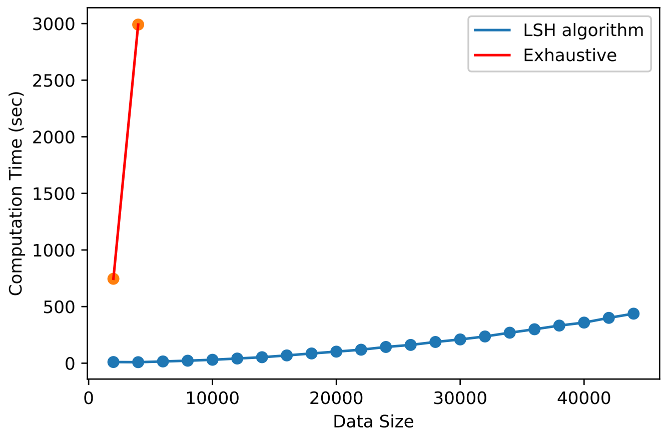

Fig. 1: Computation time curve with different algorithms

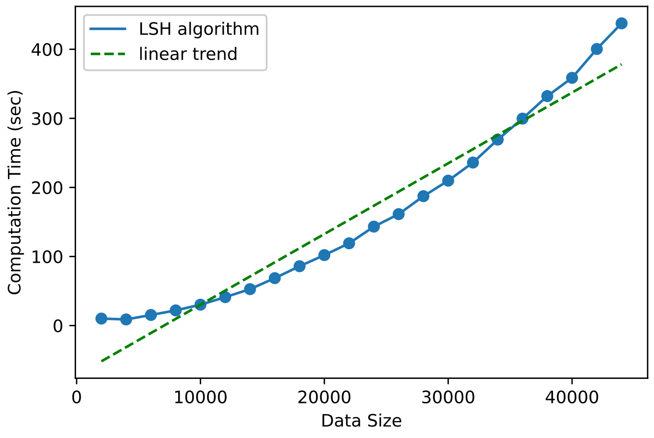

Fig. 2: Computation time curve of LSH algorithm

hard to get the overview of the performance of finding similar questions. But we can have a look at the part of similar pairs found. **Table 4** lists the body of two pairs which are the first one detected and the pair with the lowest distance of the first 10 pairs. **446500** and **446600** are almost the same. The high distance comes from the last several sentences in **446500**. **1041520** and **1042370** are real similar questions talking about the related questions but not duplicates.

IdA	IdB	Jaccard Distance
446500	446600	0.47
950210	950960	0.45
1041520	1042370	0.33
1063190	711820	0.49
1411730	1415520	0.50
1479100	52550	0.33
1572150	1596960	0.50
1709070	777910	0.46
1709070	272270	0.47
1591130	1752440	0.50

Table 3: First 10 pairs picked by LSH

5 Conclusions

This work tries to find a scalable solution to find the similar questions on Stack Overflow. Through a series of experiments, using LSH for Jaccard Similarity has

Id	Body	Distance
446500	I have two DataGridViewComboBoxColumn that I add at run time I need the items of the first DataGridViewComboBoxColumn to stay the same in all the rows of the GridView but I want the items of the second DataGridViewComboBoxColumn to be different from row to another depending on the selected item of the first DataGridViewComboBoxColumn If we say the first DataGridViewComboBoxColumn represents the locations and the second DataGridViewComboBoxColumn represents the sublocations So I want the second DataGridViewComboBoxColumn items to be the sublocations of the selected location from the first DataGridViewComboBoxColumn Like this if Canada is selectedCountry comboBoxItems State Province ComboBox Items USA QuebecCANADA selected OntarioENGLAND Manitoba AlbertaThen if you select USACountry comboBoxItems State Province ComboBox Items USA Selected CaliforniaCANADA New YorkENGLAND Montana Ohio	0.47
446600	i have two DataGridViewComboBoxColumn that i add at run time i need the items of the first DataGridViewComboBoxColumn to stay the same in all the rows of the gridview but i want the items of the second DataGridViewComboBoxColumn to be different from row to the other depending on the selected item of the first DataGridViewComboBoxColumnif we say the first DataGridViewComboBoxColumn represents the locations and the second DataGridViewComboBoxColumn to represent the sublocations so i want the second DataGridViewComboBoxColumn items to be the sublocations of the selected location from the first DataGridViewComboBoxColumn	
1041520	I have a rewrite rule set up in my htaccess file RewriteRule Crocodile Style products display php folder crocodile style amp page L NC http test bradp com drupal Crocodile Style works OK http test bradp com drupal Crocodile Style DOES NOT WORK Apache throws a The PHP logic defaults to page without a page specified so I know the script is fine What can I do ThanksNick	0.33
1042370	Hey everyone I m having rewite issues http test bradp com drupal Crocodile Style works OK http test bradp com drupal Crocodile Style DOES NOT WORK Apache throws a The PHP logic defaults to page without a page specified so I know the script is fine Here s the code RewriteRule Crocodile Style products display php folder crocodile style amp page L NC What can I do Thanks Nick	

Table 4: Body of two similar pairs

proven its advantage on computation efficiency and the example similar pairs found and checked are reasonable to accept as true result. And the traditional exhaustive search shows its defects in the experiments and fail to implement on a large-scale dataset.

6 Appendix

This project is performed through Python language, and code book is shared on Github.

7 Declaration

I declare that this material, which I now submit for assessment, is entirely my 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 work. I 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 or any other person for assessment on this or any other course of study.

References

1. Jure Leskovec, Anand Rajaraman, Jeffrey D. Ullman. "Mining of Massive Datasets", 2014
2. "Machine Learning Library (MLlib) Guide", https://spark.apache.org/docs/2.2.3/ml-guide.html