UNIVERSITÀ
DEGLI STUDI
DI MILANO

Finding Similar Items

Algorithms for massive data course project

Angelina Khatiwada
MSc, Data Science and Economics
angelina.khatiwada@studenti.unimi.it

Department of Economics, Management and
Quantitative Methods

December 12, 2021

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.

Abstract

In this project, we studied Locality Sensitive Hashing (LSH) technique to run similarity analysis on a large-scale data. We detected pairs of similar questions in the "StackSample" dataset which contains 1.26 million questions from the StackOverflow website. Spark environment was used to run computations and scale the results. We performed an exploratory analysis of the dataset, detected duplicate rows and pre-processed questions text. We extracted features from text using Term Frequency-Inverse Document Frequency (TF-IDF) method and then performed similarity analysis using LSH approximate similarity join with Jaccard distance and Cosine similarity join. We ran the algorithms on 11 chunks of different size extracted from the original dataset (from 100 to 102400 rows). Finally, we compared algorithms' performance in terms of computation speed and ability to detect similar pairs. Experiment has shown that TF feature extraction with LSH-based similarity join is the most efficient algorithm.

Contents

1 Introduction

Apache Spark emerged as an efficient framework that fulfills computational requirements for massive data analysis. It implements advanced in-memory programming and provides libraries for structured data processing and scalable machine learning. Spark's core data abstraction, Resilient Distributed Dataset, is used to design and run high-performance data algorithms [1].

The objective of this project is to detect pairs of similar items in the "StackSample" dataset which contains 1.26 million questions from the Stack Overflow website. We used PySpark as an interface for Spark to run massive data analysis and explored the following modules:

- **Spark SQL**, a relational framework for structured data processing
- **Spark MLlib**, a scalable machine learning library for feature extraction and similarity analysis

We ran TF-IDF transformation of preprocessed and tokenized question text into feature vector using hashing function. To perform questions similarity analysis in a high-dimensional space, we applied an approximate similarity join based on locality-sensitive hashing (LSH). This algorithm provides a good solution while significantly reducing the processing time [2].

2 Dataset

The "StackSample" dataset published on Kaggle was used to perform similarity analysis. The dataset is released under the CC-BY-SA 3.0 license. At the time it was published in 2016, it contained texts of 10% of questions and answers from the StackOverflow programming Q&A website [3]. The dataset is organized in three tables: Questions, Answers and Tags on the questions.

To find the pairs of similar questions, we analyzed the description of the questions asked. The entire dataset weights about 3.6 GB (the file Questions.csv accounts for 1.92 GB). The dataset was downloaded directly to the Google Colab notebook during code execution via the Kaggle API.

We analyzed only the table "Questions" which contains the following fields:

- Id
- Title
- Body
- Creation date (questions asked from August 2008 till October 2016)
- Closed date
- Score
- Owner Id

3 Data pre-processing

3.1 PySpark dataframe

As a first step, we created a PySpark dataframe from the Questions table. The dataframe represents a distributed collection of data grouped into named columns [4]. "ISO-8859-1" encoding was used to read all the rows correctly and the first line was set as a header. The input schema is automatically inferred from data and the records that span multiple lines are parsed as one record. The dataframe consists of 1,264,216 records and 7 columns.

For the further analysis, we dropped the columns OwnerUserId, CreationDate, ClosedDate, Score keeping only Id, Title and Body (Table 1).

Id	**Title**	**Body**
80	SQLStatement.exec...	<p>I've written a...
90	Good branching an...	<p>Are there any ...
120	ASP.NET Site Maps	<p>Has anyone got...
180	Function for crea...	<p>This is someth...
260	Adding scripting ...	<p>I have a littl...
330	Should I use nest...	<p>I am working o...
470	Homegrown consump...	<p>I've been writ...
580	Deploying SQL Ser...	<p>I wonder how y...
650	Automatically upd...	<p>I would like t...
810	Visual Studio Set...	<p>I'm trying to ...

Table 1: Questions dataframe created for the analysis

3.2 Missing and unique values

As a next step, we checked if there are any missing values in the columns and counted the numbers of unique values. There are no missing values in the dataframe. No duplicates were found in the Id column (unique identifier), whereas there are 221 and 12 pairs of duplicates in the Title and Body columns respectively (Table 2).

Column names	**Id**	**Title**	**Body**
Count NA values	0	0	0
Count unique values	1264216	1263995	1264204

Table 2: Number of missing and unique values in the columns

3.3 Duplicate values

We verified the number of duplicates in Body column by grouping records and counting values with more than 1 occurrence. Indeed, 12 pairs of duplicates were detected and they

are presented in Table 3. Duplicates have different Ids, but identical question body texts and might have identical or different titles. For example, questions with ids 30993210 and 31037750 have the same description (question body) but a different title. There are only two pairs of questions that have identical values in both Body and Title columns (pair 1: 5087720 & 5090290; pair 2: 2807730 & 2808090).

Body	**Id**	**Title**
<p>How I can disp...	30993210	dompdf - display ...
<p>How I can disp...	31037750	How to determine ...
<p>I'm attempting...	3400470	Create a smooth f...
<p>I'm attempting...	6426300	Android: Smooth f...
<p>I got problem ...	30128150	Can not find post...
<p>I got problem ...	30138690	Can not connect w...
<p>I have Some Pa...	6744140	Store JavaScript ...
<p>I have Some Pa...	6744970	Store JavaScript ...
<p>I'm working on...	38120120	Inserting to sort...
<p>I'm working on...	38124500	Inserting element...
<p>I am using MS ...	23876060	ODBC SQL Server D...
<p>I am using MS ...	24001180	ODBC SQL Server D...
<p>PayPal IPN sen...	2807730	Cannot send value...
<p>PayPal IPN sen...	2808090	Cannot send value...
<p>Anyone know ho...	23401620	how to open and r...
<p>Anyone know ho...	23403060	How to read the u...
<p>Is it possible...	4115730	Iphone simulator ...
<p>Is it possible...	4380590	run asp.net 3.5 s...
<p>I am trying to...	5087720	Android: BitmapDr...
<p>I am trying to...	5090290	Android: BitmapDr...
<p>How can I use ...	4815160	How can I use an ...
<p>How can I use ...	16528470	How can I use an ...
<p>When I tried t...	35740350	How do I position...
<p>When I tried t...	35757150	CSS how to positi...

Table 3: Pairs of duplicate rows in the Body column

3.4 Text pre-processing

We removed Title and used only description text (Body) to detect pairs of similar questions. The following transformations were performed over the questions text:

- Text tokenization
- Transforming to lowercase
- Removing HTML tagging
- Removing punctuation
- Removing numbers
- Removing stopwords

We transformed the text to lowercase and applied regular expressions to remove markup language tags, punctuation and numbers, so that only alphabetic characters are kept. Then we split sentences into sequences of words using Tokenizer from Spark machine learning module.

As a next step, we removed stop words from the list of tokens using StopWordsRemover function. We used a default list of stop words proposed by Spark, which contains 181 stopwords such as 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', 'your' etc.

We also counted the number of tokens in each question with and without stop words removal: 'Sw.r. count' and 'Count' columns in Table 4. We can see that in some cases stop words account for 40-50% of the tokens in the question descriptions. The lists of tokens for both cases are shown in the table as well.

Id	**Body**	**Tokens**	**Count**	**Tokens sw. r.**	**Sw. r. count**
80	ive written a dat...	[ive, written, a,...	349	[ive, written, da...	317
90	are there any rea...	[are, there, any,...	24	[really, good, tu...	14
120	has anyone got ex...	[has, anyone, got...	58	[anyone, got, exp...	40
180	this is something...	[this, is, someth...	41	[something, ive, ...	21
260	i have a little g...	[i, have, a, litt...	181	[little, game, wr...	96
330	i am working on a...	[i, am, working, ...	170	[working, collect...	89
470	ive been writing ...	[ive, been, writi...	54	[ive, writing, we...	31
580	i wonder how you ...	[i, wonder, how, ...	256	[wonder, guys, ma...	142
650	i would like the ...	[i, would, like, ...	156	[like, version, p...	82
810	im trying to main...	[im, trying, to, ...	188	[im, trying, main...	98

Table 4: List of tokens and their count with and without stop words removal

4 Research methodology

4.1 Feature extraction

Term frequency-inverse document frequency (TF-IDF) method is applied to extract features from the pre-processed text. TF-IDF vectorizes text and indicates the importance of a term in a specific document. Term frequency TF (t,d) represents the number of times term t appears in document d. IDF is used to down-weight the terms that appear often in the corpus and highlight the document-specific terms [5]. IDF is 0 when a term appears in all the documents. TF-IDF is the product of TF and IDF:

$$TFIDF(t, d, D) = TF(t, d) \cdot IDF(t, D) = IDF(t, D) = \log \frac{D' + 1}{DF(t, D) + 1}$$

Document frequency DF (t,D) in the formula above is the number of documents (D) that contains term t and D' is the total number of documents in the corpus. In general, IDF improves the performance of algorithms by reducing the importance of frequent

terms. However, removal of stop words and then computing only TF can also have a similar effect since we eliminate the words that do not carry important information in the pre-preprocessing stage.

HashingTF from Spark ML is used to map a raw feature into an index(term) by applying a hash function and then calculating TF based on the indices. A global term-to-index map is not computed which makes it a good solution for large datasets. One of the challenges of this technique is a potential hash collision (different raw features are mapped to the same hashed value). This can be fixed by increasing feature dimension, i.e. the number of buckets of the hash table [5].

IDF function takes TF vectors and scales each feature according to the importance for the specific document.

4.2 Similarity analysis

Locality sensitive hashing (LSH) is an approximation technique used for similarity search in a high dimensional space (Figure 1). LSH combines similar data points in the same hash buckets with a high probability [2]. LSH family of functions should satisfy the following conditions, where M is a set of documents, d is a distance function on M [5]:

$$\begin{aligned} &\forall p, q \in M, \\ d(p,q) \leq r1 &\Rightarrow Pr(h(p) = h(q)) \geq p1 \\ d(p,q) \geq r2 &\Rightarrow Pr(h(p) = h(q)) \leq p2 \end{aligned}$$

The LSH family is called (r1, r2, p1, p2)-sensitive, where r1 and r2 denote threshold distances; p1 and p2 - threshold probabilities. [5].

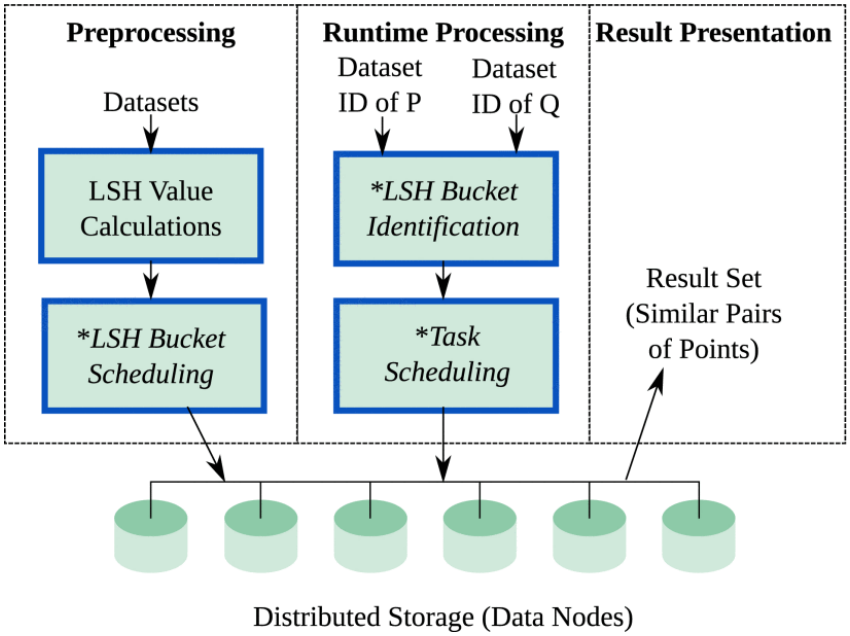

Figure 1: Locality sensitive hashing technique for similarity search. Source: [2]

MinHash function is applied to perform an approximate similarity join and return pairs of documents (rows) whose Jaccard Distance is smaller than the given threshold. Jaccard distance of two documents is defined by the cardinality of their intersection and union. MinHash applies a random hash function g to each element and takes the minimum of all hashed values [5]:

$$h(\mathbf{A}) = \min_{a \in \mathbf{A}}(g(a))$$

5 Experimental results

We ran three algorithms to find similar questions in the "StackSample" dataset:

1. TF feature extraction and LSH approximate similarity join
2. TF-IDF feature extraction and LSH approximate similarity join
3. TF feature extraction and cosine similarity join (inner product of feature vectors)

Before running the algorithms, we removed empty sets of tokens from the dataset, because MinHash function can transform only vectors that have at least 1 non-zero entry.

As the dataset contains more than 1.26 million rows, a lot of resources are required to perform computations on the entire set. Therefore, we decided to compare the performance of the algorithms on the chunks of different size extracted from the original dataset: 100, 200, 400, 800, 1600, 3200, 6400, 12800, 25600, 51200, 102400 rows. We studied performance on different chunks in terms of ability to find similar questions and time to run computations.

5.1 Feature extraction and similarity join

Algorithm 1: TF and LSH approximate similarity join. For each question description, we used HashingTF to transform the list of tokens into a feature vector. The output is the sparse vector, for example, SparseVector(1024, {3: 1.0, 6: 1.0, 11: 1.0, ... 1015: 1.0}), i.e. there are 1024 elements in the space and the document contains elements 3, 6, 1015 etc. Then we applied MinHash function that computes the locality sensitive hashes and approximately returns pairs of questions with Jaccard distance smaller than 0.6. The dataset was joined on itself within approximate similarity join procedure, therefore, we introduced a condition that the document Ids in the pair should be different. This condition removed duplicate Id pairs with 0 distance.

We ran the analysis on 11 chunks of data increasing the size from 100 (0.00008%) to 102400 (0.08 %). Results are the following:

- Computation time increased from 1.081 seconds for 100 rows to 1862.97 seconds for 120400 rows
- The first pair of similar questions with distance 0.5728 (less than 0.6 threshold) was detected when we analysed the chunk of 6400 rows. No similar questions within the threshold given were found when we considered smaller chunks.

Table 5 shows the top ten most similar pairs of questions identified for a 51200-row chunk (left) and a 102400-row chunk (right). The pairs are sorted in ascending order according to the Jaccard distance (0 indicates that the items are identical or very similar).

id1	id2	JaccardDistance
1999930	2012430	0.169
2412400	2413480	0.193
1041520	1042370	0.303
52550	1479100	0.333
1182220	2062480	0.333
503310	835280	0.375
2222470	2327120	0.376
715630	2142730	0.378
2046970	2065370	0.437
1722750	1722860	0.439

id1	id2	JaccardDistance
2311160	4115730	0.0
3669960	3676110	0.0
2576770	2577140	0.0
2807730	2808090	0.0
2311160	3633320	0.0
3633320	4115730	0.0
4000000	4000780	0.071
3748710	3776900	0.148
1999930	2012430	0.169
2412400	2413480	0.193

Table 5: The most similar pairs: 51200-row chunk (left); 102400-row chunk (right)

As we increase chunk size, the algorithm is able to find more pairs of similar documents and also pairs with a smaller distance. The analysis of a 102400-row chunk detected 6 pairs of questions with 0 distance. The pair with ids 2807730 and 2808090 was also highlighted as a pair of duplicates in section 3.3, Table 3.

We do not have a ground truth to evaluate the accuracy of the results, however, we can manually verify if the top pairs are relevant. We extracted Body, Title and UserId for some pairs detected (presented in Table 6). We can see that questions 2311160 and 4115730 have a different title and were asked by different users. However, question texts are the same once we remove the stop words. It is the case when description is contained in the Title of the question and not the Body. If we look at the questions 3669960 and 3676110, we can see that they have the same title and were asked by the same user. However, question bodies are not identical as one of them has extra spaces between the words, and therefore these questions are not highlighted as duplicates in section 3.3.

Algorithm 2: TF-IDF and LSH approximate similarity join. We used exactly the same algorithm as before, the only difference is that we calculated TF-IDF in the feature extraction part. We applied IDF to rescale feature vectors and then computed MinHash function and performed approximate similarity join. In our case, adding IDF did not improve performance: Algorithm 2 detected the same pairs of questions for different chunks as the Algorithm 1. Moreover, IDF computation required more time: from 1.753 seconds for 100 rows to 3336.834 seconds for 102400 rows.

Algorithm 3: TF and cosine similarity join. In order to compare the performance of MinHash implementation, we also ran cosine similarity join on the same chunks of data. We normalized TF vectors and then calculated cosine similarity as an inner product of normalized vectors. Similarity measure was computed for all the pairs of questions in the chunk. This algorithm required much more time for all the computations: time increased from 2.429 seconds for 100 rows to 1345.675 seconds for 3200 rows (we did not run the algorithm on larger chunks).

Distance	Id	UserId	Title	Body
0.0	2311160	116875	Is There a way to update/refresh only one record in delphi dbgrid?	Is it possible? How? \n
	4115730	495315	Iphone simulator without iphone SDK?	Is it possible? \n
0.0	3669960	32816	How can I debug an Actionscript project in Flex builder?	I am trying to use Flex builder for the first time in years. I haven't used the "Run Application" option before, and when I do that now it tells me "Errors exist in required projects" and whether I should proceed. I would like to debug those errors. \n\n Does anyone know how I can do that? \n\n PS: When I click the "Debug" button, it does exactly the same thing. I don't see error output in the console views. \n
	3676110	32816	How do I debug an Actionscript project in Flex builder?	I am trying to use Flex builder for the first time in years. I haven't used the "Run Application" option before, and when I do that now it tells me "Errors exist in required projects" and whether I should proceed. I would like to debug those errors. \n\n Does anyone know how I can do that? \n\n PS: When I click the "Debug" button, it does exactly the same thing. I don't see error output in the console views. \n
0.15	3748710	303492	jquery form submission	Can someone show me a tutorial of using jquery to display successful form submission without refreshing the page. Something like that happens on gmail when a message is delivered and the yellow overlay that shows that you message was delivered and then fade outs. \n
	3776900	455257	Form submit without page refresh	Can someone show me a tutorial of using jquery to display successful form submission *without refreshing the page*. Something like that happens on gmail when a message is delivered and the yellow overlay that shows that you message was delivered and then fade outs.\nI want the message to be displayed depending on the result of the form submission. \n

Table 6: Examples of the similar question pairs

5.2 Performance comparison

We compared the performance of three algorithms in terms of computation time on 11 chunks of different size. The results are shown in Figure 2. We stopped running analysis at 3200 rows for the Algorithm 3, because it already required a lot of time. There are 5118400 possible pairs of questions in the 3200-row chunk, whereas for the entire dataset of 1.26 million rows the number of chunks increases to 799120415220.

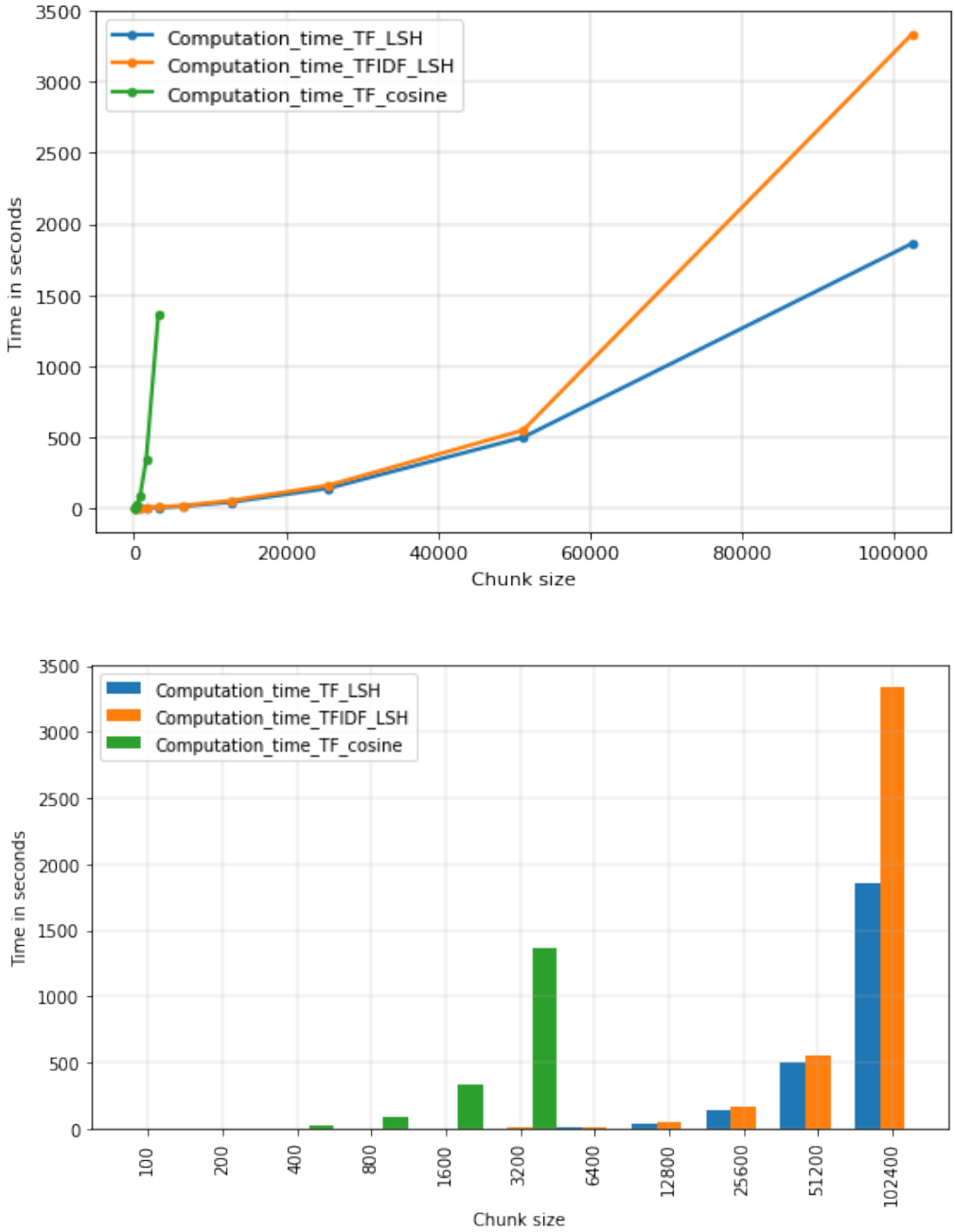

Figure 2: Time required to run the algorithms on different chunks of data

We can see from the figure that TF approximate similarity join (Algorithm 1) ran 2 times faster than TF cosine similarity join (Algorithm 3) on 100 rows and 244 times faster on 3200 rows. Algorithms 1 and 2 show more or less the same performance on

small chunks, however, the difference grows as the chunk size increases. For instance, Algorithm 1 performs computations on 102400 rows 1.8 times faster than Algorithm 2 as the latest requires more time to compute the IDF part. Both of the algorithms detect the same pairs of questions and it might be explained by the fact that we already removed stopwords in the text pre-processing part. In our case, rescaling features with IDF does not add any value, but only increases computation time.

6 Conclusion

Our experiment shows that Algorithm 1 is the most efficient in terms of computation speed and is able to detect relevant pairs of similar questions. To improve performance, we can also consider adding questions titles to the analysis and apply more advanced techniques for feature extraction from text, for instance, word embedding.

As a final step, we decided to predict the time required to run Algorithm 1 on the entire dataset. From Figure 2, we can assume that there is a non-linear relationship between chunk size and computation time. We fitted the running time for 11 chunks in the Polynomial Regression model and predicted the time required to run the algorithm on 1.26 million rows. We ran the model for different degrees and results are the following:

1. 1th degree polynomial (Linear Regression model): 5:56:21.96
2. 2d degree polynomial: 3 days, 1:49:37.65
3. 3d degree polynomial: 2 days, 11:15:32.01
4. 4th degree polynomial: 240 days, 17:03:36.65

If we assume that there is a quadratic relationship between the variables, it would take around 3 days to run the Algorithm 1 on the entire dataset.

References

[1] S. Salloum, R. Dautov, X. Chen, P. X. Peng, and J. Z. Huang, "Big data analytics on apache spark," *International Journal of Data Science and Analytics*, vol. 1, no. 3, pp. 145–164, 2016.

[2] H. Li, S. Nutanong, H. Xu, F. Ha, *et al.*, "C2net: A network-efficient approach to collision counting lsh similarity join," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 3, pp. 423–436, 2018.

[3] "Kaggle. StackSample: 10% of Stack Overflow QA." `https://www.kaggle.com/stackoverflow/stacksample`. [Online; Accessed: 2021-12-11].

[4] "Apache Spark SQL Guide." `https://spark.apache.org/docs/latest/sql-getting-started.html`. [Online; Accessed: 2021-12-11].

[5] "Apache Spark Machine Learning Library Guide: Extracting, transforming and selecting features." `https://spark.apache.org/docs/latest/ml-features.html`. [Online; Accessed: 2021-12-11].