DAT470/DIT065 Assignment 6

Mirco Ghadri
mircog@chalmers.se

May 23, 2024

This assignment focuses on implementing hyperloglog algorithm using PySpark and testing it on the twitter dataset. Problem 2 focuses on Locality-Sensitive hashing(LSH) and using it to estimate similar words for a list of 10 queries. The full training dataset is found here.

Problem 1

(a) *Implement HyperLogLog in PySpark. Use the skeleton file pyspark_hyperloglog.py as your starting point. You should not reuse the class from the previous assignment; instead, formulate the computation using RDDs and Spark primitives. Use a broadcast variable to broadcast the array A describing the simple tabulation hash function h to all your workers. You may assume that all your input consists of lines in the format x y, i.e., two integers separated by a space.* (8 pts)
Answer: See `assignment6_problem1.py`.

(b) *The file /data/2024-DAT470-DIT065/twitter_edges_full.txt contains the Twitter dataset [1]. It has been formatted to list only one follows relation per line. Each line contains two Twitter IDs separated by a space: x y. This means that y follows x. Each ID can occur multiple times, as the same person can be followed by multiple users, and one user can follow multiple other users.*

The files /data/2024-DAT470-DIT065/twitter_edges_*.txt contain subsampled versions of the data. Use your implementation to estimate the number of distinct users in the dataset. Use the 100M version to evaluate the scalability of your PySpark implementation, and plot the speedup as a function of the number of cores. Also, run your algorithm once on the full dataset (use sufficiently many workers, otherwise it can take a while) and report the estimated cardinality in your report. (4 pts)

Answer: We got the following values for the total running time on the 100M version of the twitter dataset using different number of workers:

Number of Workers (n)	Total Running Time (seconds)
1	5659.808341026306
2	2287.991183757782
4	1210.3078362941742
8	637.0350115299225
16	372.2835500240326
32	192.20570921897888

Table 1: Running Time for Different Number of Workers

To calculate the speedup as a function of the number of CPU cores we used the formula

$$S = \frac{t_1}{t_n} \tag{1}$$

where S is the speedup factor, t_1 is the total running time with 1 CPU core and t_n is the total running time with n CPU cores. This gave us the following table of speedup values:

Number of Workers (n)	Total Running Time (t)	Speedup (S)
1	5659.808341026306	1.00
2	2287.991183757782	2.47
4	1210.3078362941742	4.68
8	637.0350115299225	8.88
16	372.2835500240326	15.20
32	192.20570921897888	29.45

Table 2: Running Time and Speedup for Different Number of Workers

This gives us the following plot:

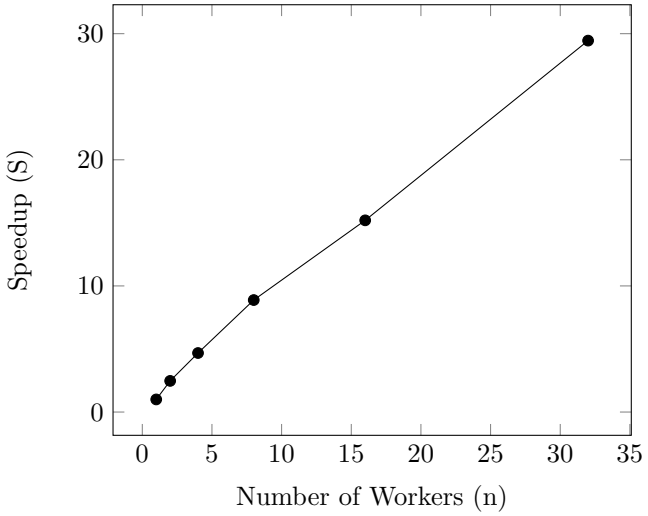

Figure 1: Speedup as a Function of Number of Workers

From the plot, we can observe that there is almost a perfectly linear relationship between the number of CPU cores used and the empirical speedup.

We also ran our PySpark HyperLogLog algorithm on the full dataset `twitter-2010_edges_full.txt`. We got the following results:

Metric	**Value**
Estimated Number of Users	42,038,978.5
Number of Workers	80
Total Time (seconds)	1112.3

Table 3: Performance Metrics on twitter-2010_edges_full.txt

Problem 2

(a) *Normalize all of the data vectors to have unit length. Make sure to use array operations. Use lsh_normalize.py as your starting point for the interface. In your report, record how many seconds it took to normalize the larger dataset (840B).* (1 pt)
Answer: See `assignment6_problem2a` for the code.

Time taken to normalize larger dataset: **5.549781084060669** seconds

(b) The file `queries.txt` contains a list of 10 words. For each word, determine the 3 closest words (not including the word itself) in terms of cosine similarity. Remember that the matrix product AB has the interpretation that $(AB)_{ij}$ is the same as the dot product between the i-th row vector of A and the j-th column vector of B. Construct a $10 \times d$ matrix Q that contains the query vectors, and compute $QX^\top$ where X is the $n \times d$ data matrix. Use `lsh_matmul.py` as your starting point for the interface.

In your report, include:

- The 3 closest words for each word in a nicely formatted table.

Word	**3 Closest Words**
priest	priests, bishop, Priest
fork	forks, spoon, Fork
horse	horses, pony, Horse
beef	pork, meat, chicken
daoist	taoist, daoism, confucian
polish	nail, polishes, nails
vehicle	vehicles, car, automobile
crepe	Crepe, chiffon, crêpe
daytime	nighttime, day-time, night-time
scotland	wales, glasgow, scottish

Table 4: Words and Their 3 Closest Words

- The time it took to compute the matrix product.

  1.1623849868774414 seconds

- The time it took to sort the results.

  3.008972644805908 seconds

- The combined time for the larger dataset (840B).

  4.17135763168335 seconds

(2 pts)

Answer: See `assignment6_problem2b.py` for the code.

(c) Implement the reduction to binary vectors using random hyperplanes. Use `lsh_hyperplanes.py` as your starting point. Suppose the input dimensionality of data is d, and the parameter (number of hyperplanes) is D. Complete the implementation of the class `RandomHyperplanes` such that, in `fit()`, you draw D random Gaussian unit vectors of length d (that is, draw D vectors $x = (x_1, x_2, \ldots, x_d)$ such that $x_1, x_2, \ldots, x_d \sim N(0, 1)$ independently at random, and then normalize them $\hat{x} = \frac{x}{||x||} = \frac{x}{\sqrt{\sum_{i=1}^{d} x_i^2}}$). Let us call the output matrix R of shape $D \times d$. Then, in `transform()`, project the $n \times d$ dataset X into D dimensions by computing $X' \leftarrow XR^\top$, and then compute matrix X'' such that $X''_{ij} = 1$ if $X'_{ij} > 0$ and $X''_{ij} = 0$ otherwise. In your report, include the amount of time it took to transform the larger dataset (860B) using $D = 50$ hyperplanes. (3 pts)

Answer: See `assignment6_problem2c.py` for the code.

Time taken to transform the larger dataset: **2.76** seconds.

(d) Implement the LSH, using `lsh_lsh.py` as your starting point. We will use the following family of elementary hash functions: $H = \{f_i : \{0,1\}^D \to \{0,1\} \mid f_i(x) = x_i, i \in [D]\}$. That is, each hash function simply selects one coordinate of the binary vector and projects it to that. We will choose parameters L and k. We will construct L hash tables by concatenating k such randomly chosen elementary hash functions, and use that hash function for the hash table. That is, for hash table ℓ, we will use a randomly chosen hash function $f^{(\ell)} : \{0,1\}^D \to \{0,1\}^k$ such that $f^{(\ell)} = (f_{i_1}, f_{i_2}, \ldots, f_{i_k})$ where $i_1, \ldots, i_k \in [D]$ have been chosen uniformly at random, and we define $f^{(\ell)}(x) = (x_{i_1}, x_{i_2}, \ldots, x_{i_k})$.

In practice, hash functions may simply be drawn by drawing an $L \times k$ matrix of random integers in the range $\{0, 1, \ldots, D-1\}$. As NumPy can index with arbitrary lists of indices, we can then just project the dataset efficiently by selecting columns by a row of the matrix.

Once we've drawn the hash functions, we will need to be able to fit the data. We do this as follows: we use the random hyperplanes from the previous problem to project the data into binary vectors. Then we hash the data L times by using the L hash functions into hash tables. The hash tables will map binary k-vectors into a set of indices that record which elements in the original data were hashed into that particular tuple.

When we want to query a vector $q \in \mathbb{R}^d$, we do as follows: we use the same random hyperplanes to transform q into a binary vector. Then we hash it L times using the same hash functions. We collect the union of all elements that were hashed in the same buckets with the vector. Finally, we compute the true distance to the vectors by taking the usual inner product with the original vectors (which we can access using the indices).

Try your implementation on the larger dataset (840B) with the following parameters: $D = 50$, $k = 20$, $L = 10$. In your report, include the time it took to fit the data and the time it took to perform the 10 queries. (5 pts)

Answer: See `assignment6_problem2d.py` for the code.

Action	**Time (seconds)**
Time taken to fit the data	224.5547046661377
Time taken to perform 10 queries	0.008310794830322266

Table 5: Time Taken for Fit and Query

However the results of closes words for the 10 queries did not seem very accurate. Here was the results:

Word	**3 Closest Words**
priest	gracious, Deb, taeyang
fork	fork, up4, min33repliesHave
horse	swimlanes, SynJ, Elkridge
beef	logistical, FSAMorsi, MorgageMarvel.com
daoist	daoist, onji, self-limitations
polish	Neldel, 26393, GameScience
vehicle	Non-CDL, Orlando-International, transsiberian
crepe	crepe, crepe, datesSusan
daytime	pretious, sulfonates, Observateur
scotland	braunton, ChiCityFashion, 20119:34

Table 6: Words and Their 3 Closest Words found using LSH approximation