Data Science

Documentation

Nearest Neighbors - Advanced (LSH)

January 21, 2018

Eicker Niklas, Halastra Szymon

Contents

1 Introduction

Using GraphLab Create we will load and analyze data on Wikipedia articles about persons. In this advanced project we will use Locality Sensitive Hashing (LSH) to achieve fast and efficient approximate nearest neighbor searches.

1.1 The data set

Each element of the original data set consists of a link to a Wikipedia article, the name of the person it is about and the text of the article (in lowercase and without punctuation). There are 59071 entries in total. An excerpt of the data "as is" can be seen in figure 1.1.

Figure 1.1: Excerpt of the data set.

URI	name	text
<http://dbpedia.org/resource/Digby_Morrell> ...	Digby Morrell	digby morrell born 10 october 1979 is a former ...
<http://dbpedia.org/resource/Alfred_J._Lewy> ...	Alfred J. Lewy	alfred j lewy aka sandy lewy graduated from ...
<http://dbpedia.org/resource/Harpdog_Brown> ...	Harpdog Brown	harpdog brown is a singer and harmonica player who ...
<http://dbpedia.org/resource/Franz_Rottensteiner> ...	Franz Rottensteiner	franz rottensteiner born in waidmannsfeld lower ...
<http://dbpedia.org/resource/G-Enka> ...	G-Enka	henry krvits born 30 december 1974 in tallinn ...
<http://dbpedia.org/resource/Sam_Henderson> ...	Sam Henderson	sam henderson born october 18 1969 is an ...
<http://dbpedia.org/resource/Aaron_LaCrate> ...	Aaron LaCrate	aaron lacrate is an american music producer ...
<http://dbpedia.org/resource/Trevor_Ferguson> ...	Trevor Ferguson	trevor ferguson aka john farrow born 11 november ...

2 Data preparation

As preparation we add a new column to the data set containing the TF-IDF values as well as a row count acting as unique article ID.

The GraphLab Create method 'text_analytics.tf_idf' calculates TF-IDF values for all our articles. Adding the ID id done via an in-build method of the SFrame 'add_row_number'. A short preview of the modified data set can bee seen in figure 2.1.

Figure 2.1: Preview of the data set with an additional column for TF-IDF and unique IDs.

id	URI	name	text	tf_idf
0	<http://dbpedia.org/resource/Digby_Morrell> ...	Digby Morrell	digby morrell born 10 october 1979 is a former ...	{'selection': 3.836578553093086, ...
1	<http://dbpedia.org/resource/Alfred_J._Lewy> ...	Alfred J. Lewy	alfred j lewy aka sandy lewy graduated from ...	{'precise': 6.44320060695519, ...
2	<http://dbpedia.org/resource/Harpdog_Brown> ...	Harpdog Brown	harpdog brown is a singer and harmonica player who ...	{'just': 2.7007299687108643, ...
3	<http://dbpedia.org/resource/Franz_Rottensteiner> ...	Franz Rottensteiner	franz rottensteiner born in waidmannsfeld lower ...	{'all': 1.6431112434912472, ...
4	<http://dbpedia.org/resource/G-Enka> ...	G-Enka	henry krvits born 30 december 1974 in tallinn ...	{'they': 1.8993401178193898, ...
5	<http://dbpedia.org/resource/Sam_Henderson> ...	Sam Henderson	sam henderson born october 18 1969 is an ...	{'currently': 1.637088969126014, ...

For the rest of the assignment we will use sparse matrices instead of the SFrame. A sparse matrix is a matrix with only a few non zero elements. SciPy supports this format and allows us to easily handle our TF-IDF values in matrix form. First we have to convert the original dictionary structure to the new SciPy sparse matrix format.

We transform the TF-IDF dictionaries in one 59071 x 547979 sparse matrix, where each row is a document and each column a word. Now every document has a TF-IDF value for every word in any document. If the word does not appear in the document, which is the case for most words, the value is zero.

3 Locality Sensitive Hashing

LSH is a hashing method that aims to maximize the probability of a "collision" for similar items. Via randomly generated vectors we will assign each document to a specific bin. The total number of bins should be chosen in a way, that the bin of a queried document and it's surrounding bins include the "real" nearest neighbors. Due to the smaller amount of the documents in surrounding bins (compared to the complete sample), it will be possible to calculate the closest neighbors faster then by brute force.

3.1 Sorting data into bins

For the beginning we have to decide on a number of random vectors. This number will define the number of bins. With each vector we will generate one bit for every document by multiplying it with the document's TF-IDF vector and checking the sign of the solution. Each unique combination of bits represents one data bin.

We create 16 Gaussian random vectors, which will yield 2^{16} (65536) bins to sort our data into. In this case the number of bins is comparable to the number of documents in the data set.

Each scalar product with a document and all random vectors gives us an array of Boolean with the length of 16 (= number of vectors). We can transform this array into an ID for the corresponding bin, by taking it as a 16-bit binary number and calculating it's single integer value. At this point we have sorted all our documents into bins and can start to compare single documents to other documents inside the same bin and surrounding bins.

3.2 First evaluation of a LSH model

With every document in it's corresponding bin, we can take a look at the actual "closeness" of documents in the same and surrounding bins. To measure the closeness we will be using the cosine distance, which was closer investigated in the primer task.

As an example we will be using the Wikipedia page of Barack Obama. In Obama's bin there are five other pages. Table 3.1 lists the names and cosine distances of these documents to Barack Obama's article. Most of them are far away from Barack Obama with cosine distances above 0.9, but the closest is Joe Biden with a distance of 0.7. It is obvious, that LSH is just an approximation and one has to consider documents from surrounding bins to find the real nearest neighbors.

Table 3.1: Comparison of the documents that share the same bin as Back Obama's article.

Rank	**Name**	**Distance to Obama**
1	Joe Biden	0.703139
2	Mark Boulware	0.950867
3	John Wells (politician)	0.975966
4	Francis Longstaff	0.978256
5	Madurai T. Srinivasan	0.993092

4 Querying the LSH model

4.1 Structure of a query

In the previous chapter we found that a query of our model will have to take surrounding bins into account. One can define a range for such a query as the number of different bits in the bins bit representations. Such a query with the range 3 will look like this:

1. Let L be the bit representation of the bin that contains the query documents.
2. Consider all documents in bin L.
3. Consider documents in the bins whose bit representation differs from L by 1 bit.
4. Consider documents in the bins whose bit representation differs from L by 2 bits.
5. Consider documents in the bins whose bit representation differs from L by 3 bits.

4.2 Query implementation

For an easy implementation of the query we use "itertools.combinations". This method takes the number of our vectors (number of bits in a bin ID) and the query range. It returns a collection of lists, which show all possible bit flips that lead to surrounding bins with the given range as maximum number of flips. Our query method takes these lists of bit flips, applies each one of them and saves all documents found in any of the reached bins in a set. This set then includes all possible candidates for being nearest neighbors and can be further examined.

The first part of the query gives us a set of candidates. From this set we compute the true cosine distances of the candidates to the queried document and then take a look at the top 10. The names and distances of the results for a query radius of 3 can be found in table 4.1 and are a lot better then the results from just looking at the other documents in the same bin.

Table 4.1: Top ten nearest neighbors for Barack Obama. This data is obtained with 16 random vectors and a search range of 3.

Rank	**Name**	**Distance to Obama**
1	Barack Obama	0
2	Joe Biden	0.703
3	Nathan Cullen	0.857
4	Barry Sullivan (lawyer)	0.875
5	Neil MacBride	0.89
6	Vikramaditya Khanna	0.898
7	Herman Cain	0.899
8	Raymond F. Clevenger	0.901
9	Michael J. Malbin	0.903
10	Lowell Barron	0.909

5 Tuning of the LSH model

There are two main aspects of our model that we would like to tune.

1. **Speed**
   The faster the query the better. This is why we use an approximate technique like LSH in the first place.

2. **Accuracy**
   Speed is nice, but the search results should still be close to the exact results by brute force.

5.1 The range parameter

The range parameter affects our queries by defining the number of bins searched. This also affects the number of documents for the exact distance calculation. Thus both speed and accuracy can be tuned with this parameter.

To take a closer look at the affect of the range parameter we will query Obama once more, but this time with ranges from 0 to 16 while recording the query time and results. Figure 5.1 shows how the number of considered documents raises with the search radius. On the other hand, more documents also mean longer query times, as can be seen in figure 5.2. Figure 5.4 finally shows the raising accuracy of our search, while we are approaching the brute force results.

This evaluation is only for an "Obama query". We repeated it for 10 randomly chosen documents and recorded the time per query and how many of the top 10 where in the top 25 of brute force. The later precision measurement can be seen in figure 2.1. The results look similar to the single query results with Obama. With a range of 7 we get precise top 10 documents and still have a time advantage over brute force (<50%).

Figure 5.1: Number of documents searched for different search radii.

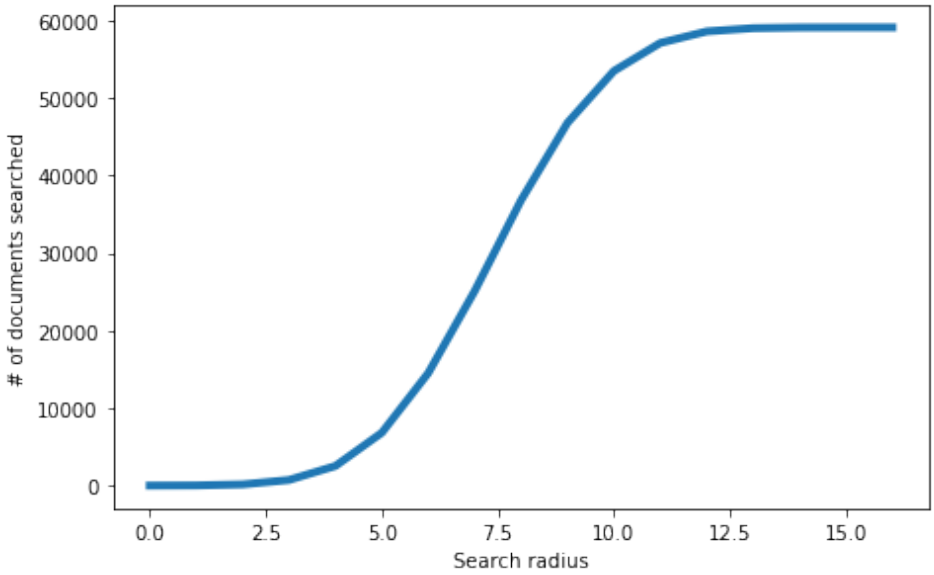


Figure 5.2: Time elapsed for one query with changing search radius.

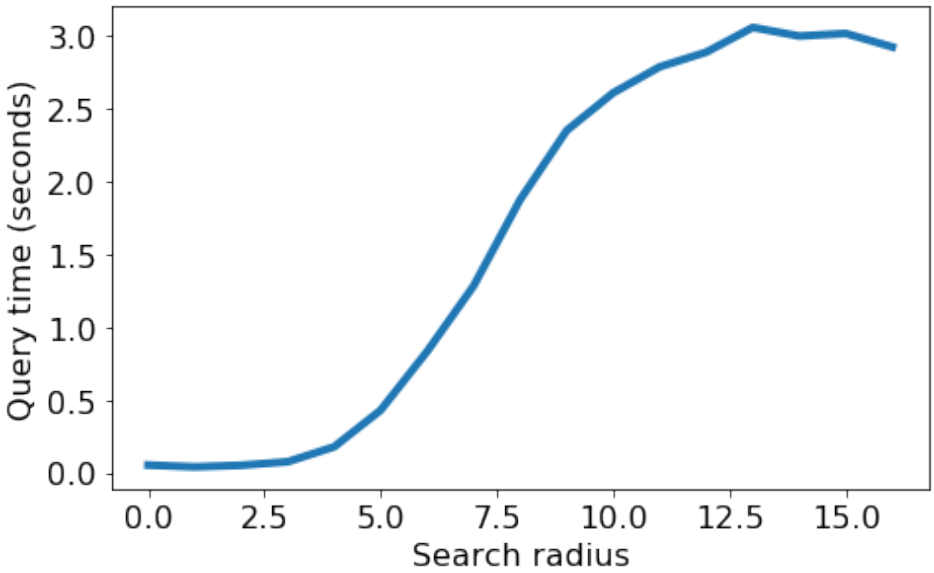


Figure 5.3: Distances of the top 10 documents to the queried article.

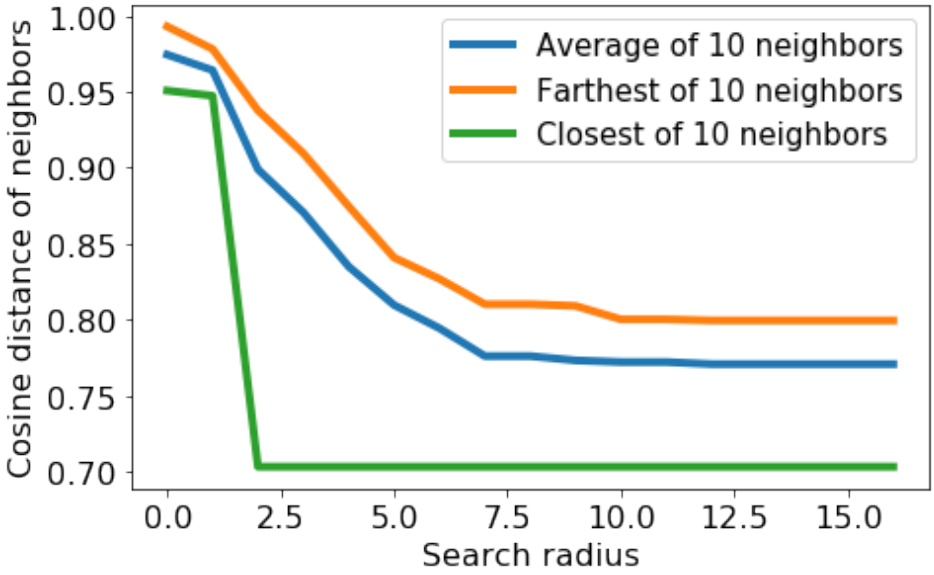
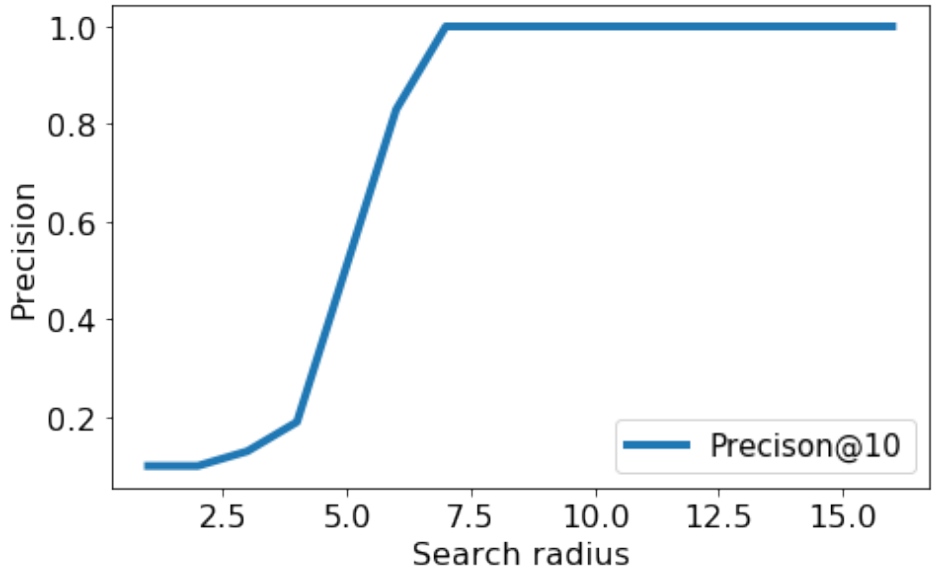


Figure 5.4: Precision of the top 10 query compared to the top 25 of brute force.



5.2 The number of random vectors

To asses the effects of a changing number of random vectors, we pick 10 random documents and query them with a fixed range of 3 and vary the number of random vectors

between 5 and 19.

As with the range parameter there is a trade off between precision and query time. With less vectors, the documents are split into less bins and a search with fixed range will look through more documents. This raises the precision, but also makes the queries slower. Figure 5.5 shows the falling precision with more random vectors, while figure 5.6 visualizes the faster queries. Keep in mind that this all happens at range 3.

Figure 5.5: Precision of the top 10 being in the top 25 of brute force. (Range = 3)

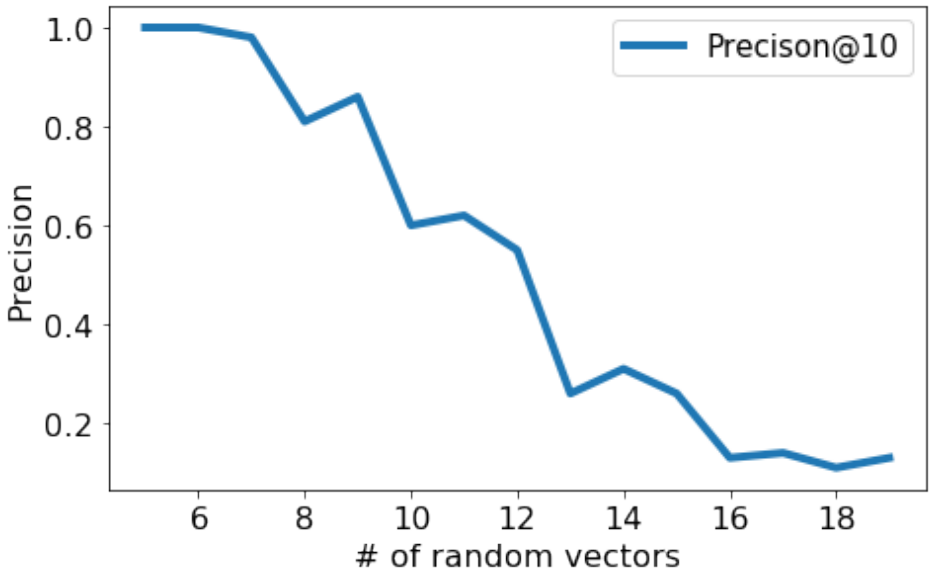
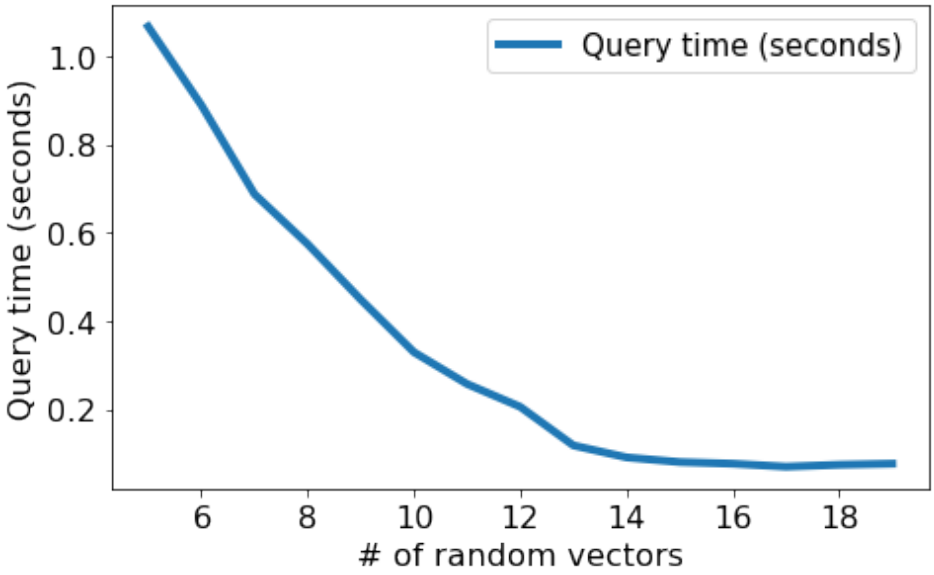


Figure 5.6: Time elapsed during one query. (Range = 3)



5.3 Balancing the parameters

We found that a range of 3 in section 5.2 was a bit low, when looking at the analysis in section 5.1. Out of interest we changed the range to 5 and repeated the analyses for 10 random documents with the number of random vectors between 5 and 19. Figure 5.7 shows, that we still get a precision of 1 for 11 random vectors at an astonishing 0.75s per query (5.8).

Figure 5.7: Precision of the top 10 being in the top 25 of brute force. (Range = 5)

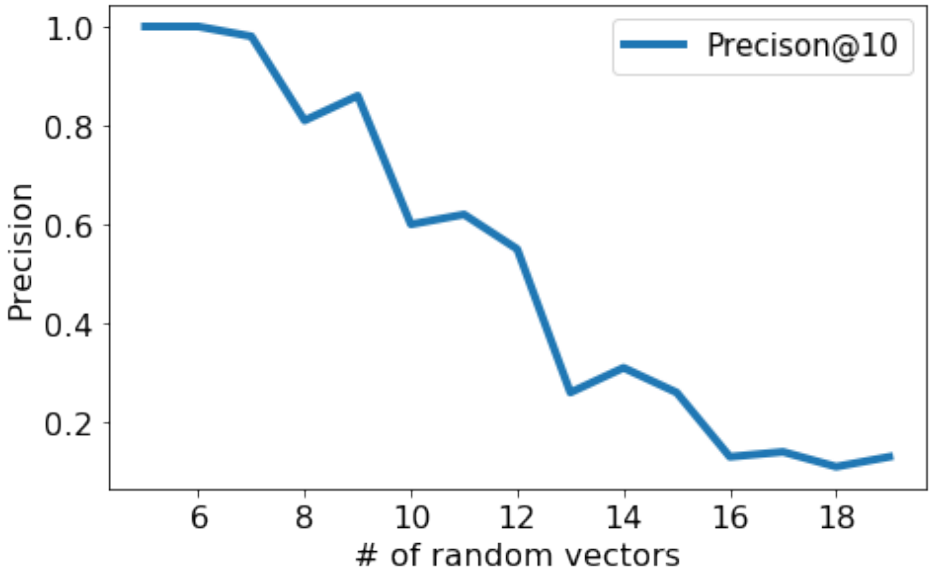


Figure 5.8: Time elapsed during one query. (Range = 5)