ENTITY RESOLUTION:
A STUDY OF LOCALITY SENSITIVE HASHING & MACHINE LEARNING BASED APPROACHES

1. Introduction

1.1 An entity is defined as unique person, event or object and may have been referred to in different ways in multiple data records. The task of 'resolution' refers to reducing or separating something into its constituents. Thus, Entity Resolution (ER) in its very basic definition refers to the task of identifying and linking/grouping various manifestations of the same Entity across and within datasets or data sources. For example, there could be different ways of addressing the same person in text, addresses for businesses, or photos of a particular object. The goal of ER is to resolve entities, by identifying the records that represent the same entity and reconcile them to obtain one record per entity.

1.2 Despite the fact that the world is saturated in profound amounts of data, the unfortunate reality is that data was never designed to be used in conjunction with other datasets for informed decision making. As such, datasets containing different aspects of information about a particular Entity are often found across the storage landscape in different places, within different schemas, and without consistent join keys. Moreover, the problem that arises when we search for similar items of any kind in a large dataset is that there may be far too many pairs of items to test each pair for their degree of similarity. This is where ER is useful, to methodically integrate the large, dissimilar data source and provide a all-inclusive view on the entity of interest. (Huang, 2020).

2. Motivation for ER

2.1 The motivation for ER stems from its ability to unlock value from data through data integration. A complete data picture, created by integrating different pieces of data about the entity of interest, is much more effective at informing decision making than a partial or an incomplete view. For example, an investor can be much more effective at selecting the best investment if she had an integrated view of all the data on potential companies, starting from the quality of its team, to its traction in the market and the buzz surrounding it. (Huang, 2020)

3. Methods for ER

3.1. A large number of ER libraries exist both in python and R, however, Locality Sensitive Hashing technique forms the basis of almost all algorithms

used for ER. Recently, Machine Learning techniques have been utilized for Entity Resolution or its close process of Named Entity Resolution (NER). Many new approaches including use of single layer perceptron, crowdsourcing etc. are also being developed to improve the efficiency and also to reduce the time of entity resolution.

4. **Objective**

4.1. The paper aims at studying ER as an application of LSH and implementation of a working program for ER in python utilization libraries incorporating LSH as well as shallow machine learning techniques. The paper has been structured as follows: -

a. **Part I**. Study of LSH and the methodology of carrying out ER using LSH.

b. **Part II**. Implementation of an ER program in python using LSH.

c. **Part III**. Implementation of ER program utilizing customized python libraries incorporating machine learning aspects.

5. **Part I. Study of LSH and the Methodology of carrying out ER using LSH**

5.1 LSH probably owes its origins to the problem of finding similar documents – those that share a lot of common text. It's important to note that finding similarity does not refer to extracting semantic meaning of the documents, but implies looking at whether they contain the same words. This task of finding similarity in documents can be divided into three broad parts as follows (Leskovec, 2019) :-

a. Shingling. Firstly, we convert documents into sets in a way that lets us view textual similarity of documents as sets having a large overlap. To measure the similarity between these two sets, we use Jaccard similarity, the ratio of the sizes of their intersection and union.

b. Minhashing. This is a way to convert large sets into much smaller representations, called signatures that still enable us to estimate closely the Jaccard similarity of the represented sets.

c. Locality Sensitive Hashing. The family of functions (known as LSH families) to hash data points into buckets so that data points near each other are located in the same buckets with high

probability, while data points far from each other are likely to be in different buckets. Thus, LSH makes it easier to identify observations with various degrees of similarity. (Gupta, 2018)

5.2. **Shingles**.

5.2.1 Shingles is a very basic broad concept. For text, shingles could be characters, unigrams or bigrams. A set of categories can be used as shingles.

5.2.2 Documents to Sets. Shingles simply reduce our documents to sets of elements that so we can calculate similarity between sets. It is more common to parse the document by taking, for example, each possible string of three consecutive words from the document (e.g., "A small detail", "small detail here", "detail here is", etc.). This retains a little more of the document structure than just hashing the individual words. (McCormick, 2015)

5.2.3 Set of Integers. Instead of using substrings directly as shingles, we can pick a hash function that maps strings of length k to some integer and then we can use the integer as the shingle. This technique of hashing substrings is referred to as "shingling", and each unique string is called a "shingle".

5.2.4 Matrix Representation of Sets. It is helpful to visualize a collection of sets as their characteristic matrix. The columns of the matrix correspond to the sets, and the rows correspond to elements of the universal set from which elements of the sets are drawn. As an example, the following two sentences demonstrate the similarity problem.

S1: "I enjoyed my stay during summer at hotel Lalit"
S2: "I enjoyed my stay during winter at hotel Umrao"

We now have the following sets (using the words with ignore case):

S1 = {i, enjoyed, my, stay, during, summer, at, hotel, Lalit}
S2 = {i, enjoyed, my, stay, during, winter, at, hotel, Umrao}

	Word	**Set 1**	**Set 2**
1	i	1	1
2	enjoyed	1	1
3	my	1	1
4	winter	0	1

5	summer	1	0
6	stay	1	1
7	lalit	1	0
8	during	1	1
9	at	1	1
10	umrao	0	1
11	hotel	1	1

Fig 1.Characteristic Matrix M

5.2.5 Similarity of Sets. The similarity of these two sets is defined as the number of rows in which the sets agree divided by the total number of rows. So, in this case we conclude that the similarity is 7/11 which is 0.63. The problem is that storing sets sets which could have huge number of words and size could be in tera bytes will take huge memory (or disk space).

6. **MinHash Signatures**

6.1 Why do we need Signatures? Sets of shingles are large. Even if they are hashed to four bytes each, the space needed to store a set is still roughly four times the space taken by the document. The objective is to replace large sets by much smaller representations called "signatures." The two sets are then compared by estimating Jaccard similarity of the underlying sets from their signatures alone. (Petridean, Issue 37)

6.2 A simple minhash would be to minhash a set represented by a column of the characteristic matrix M, by picking a permutation of the rows. The minhash value of any column is the number of the first row, in the permuted order, in which the column has a 1. Example, take the order existing in the characteristic Matrix shown in Fig 2.

Permutation	**Word**	**Set 1**	**Set 2**
4	winter	0	1
10	umrao	0	1
8	during	1	1
5	summer	1	0
3	my	1	1
11	hotel	1	1
2	enjoyed	1	1
9	at	1	1
6	stay	1	1

1	i	1	1
7	lalit	1	0

Fig 2. Permuted Matrix

6.3 A permutation defines a minhash function h that maps sets to rows. Computing the minhash value of set S1 according to h. The first column, which is the column for set S1, has 0 in the first row, so we proceed to row 2, the second in the permuted order. There is again a 0 in the column for S1, so we proceed to row 3, where we find a 1. Thus. h(S1) = "during" and h(S2) = "winter"

6.4 Minhash Signature Vector. To represent a large collection of documents, represented by their characteristic matrix M, n random permutations of the rows of M are taken. For example, 100 permutations are taken and the the minhash functions determined by these permutations be h1, h2, . . . , hn. From the column representing set S, the minhash signature for S, the minhash vector is thus obtained as :-

[h1(S), h2(S), . . . , hn(S)].

6.5 Computing Minhash Signatures**.** It is not feasible to permute a large characteristic matrix explicitly. Fortunately, it is possible to simulate the effect of a random permutation by a random hash function that maps row numbers to as many hashes (or buckets) as there are rows. Thus, instead of picking n random permutations of rows, we pick n randomly chosen hash functions h1, h2, . . . , hn on the rows. We can then construct the signature matrix by considering each row in their given order. Example of a probable method to find minhash signatures is as follows: -

a. Let SIG(i, c) be the element of the signature matrix for the ith hash function and column c.
b. Initially, set SIG(i, c) to ∞ for all i and c.
c. We handle row r by doing the following:
    - Compute h1(r), h2(r), . . . , hn(r).
    - For each column c do the following:
        - If c has 0 in row r, do nothing.
        - if c has 1 in row r, then for each i = 1, 2, . . . , n set SIG(i,c) to smaller of current value of SIG(i, c) and hi(r).

d. This gives us a similarity of 0.5 which is fairly acceptable when compared to the actual similarity of 0.63.

6.6 MinHash signatures are calculated for each set in the data To approximate the Jaccard Similarity between two sets using MinHash signatures, the number of equal components are summed and divided by the signature length. This provides a fairly good approximation to the Jaccard Similarity between those two sets.

7. **Locality-Sensitive Hashing for Minhash Signatures.**

7.1 One 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. Consider any pair that hashed to the same bucket for any of the hashings to be a candidate pair. Thn check only candidate pairs for similarity.

7.2 Banding Technique. After finding minhash signatures for the items, an effective way to choose the hashings is to divide the signature matrix into b bands consisting of r rows each. For each band, there is a hash function that takes vectors of r integers (the portion of one column within that band) and hashes them to some large number of buckets.

8. **LSH Checklist**. Having studied aspects of LSH a summary of the steps involved in carrying out ER using LSH is as follows: -

a. Create shingles from your available data set (e.g. unigrams, bigrams, ratings, etc.)

b. Build an 'm by n' shingle matrix where every shingle that appears in a set is represented with a 1 otherwise 0.

c. Permute the rows of the shingle matrix from step 2 and build a new 'p by n' signature matrix where the number of the row of the first shingle to appear for a set is recorded for the permutation of the signature matrix.

d. Repeat permuting the rows of the input matrix and complete filling in the p×n signature matrix.

e. Choose a band size b for the number of rows you will compare between sets in the LSH matrix.

9. **Part II. Implementation of an ER program in python using LSH.**

9.1 **Implementation Objective**. It is intended to look for related computer programming books for learning a specific computer science topic. The topic is the 'Named Entity', related information about which is required to be obtained from within the dataset.

9.2 **Dataset**. The dataset being used comprises of 270 rated computer science programming books. The dataset has been obtained from Kaggle and the link is: - https://www.kaggle.com/thomaskonstantin/top-270-rated-computer-science-programing-books?select=prog_book.csv

9.3 **LSH Methodology**. For the programming, I have used unigrams extracted from the books title and description. Code attached as Appendix A.

9.4 **Machine Learning Methodology using Spacy**. The same database has been analysed through the use of the python ML based language processing package called 'Spacy', which uses a deep neural network and a machine learning library called thinc to perform its tasks. It needs a englsih dictionary/lexicon to be downloaded as well to enable use of the ML model. Code given in Appendix A.

10. **Part III. Implementation of ER program utilizing customized python libraries incorporating machine learning aspects**.

10.1 The use of 'Spacy' has been demonstrated for different tasks include a named entity resolution. Code given in Appendix B.

11. **Conclusion**

11.1 LSH was one of the earliest algorithms for similarity assessment and ER. It's variations with varied distance measures and improvements have given satisfactory results. However, the current development of ML models in language processing provide very effective tools for ER which have surpassed the efficacy of LSH based methods.

References

Gupta, S. (2018). *Locality Sensitive Hashing*. Retrieved from towards data science: https://towardsdatascience.com/understanding-locality-sensitive-hashing-49f6d1f6134

Huang, Y. (2020, Aug). *Practical Guide to Entity Resolution — part 1*. Retrieved from towards Data Science: https://towardsdatascience.com/practical-guide-to-entity-resolution-part-1-f7893402ea7e

Leskovec, R. U. (2019). *Mining of Massive Datasets.*

McCormick, C. (2015). *MinHash Tutorial with Python Code*. Retrieved from Chris McCormick : https://mccormickml.com/2015/06/12/minhash-tutorial-with-python-code/

Petridean, O. (Issue 37). *Finding similar entities in BigData models*. Retrieved from Today Soft Magazine: https://www.todaysoftmag.com/article/1553/finding-similar-entities-in-bigdata-models