Efficiently comparing search spaces using Locality Sensitive Hashing

Submitted by:-

Kathan Kashiparekh 2014B3A70792G

Guide:-

Prof. Ashwin Srinivasan

ABSTRACT

Search spaces in the Inductive Logic Programming (ILP) domain iterate over a set of all possible clauses that satisfy an example give some background knowledge in the form of facts and predicates. However, these search spaces tend to drastically increase in size and two search spaces are often composed of many similar clauses. Thus if there are two search spaces that are quite similar then time can saved by iterating over just one of them and extrapolating the results to the second search space. Given many such search spaces, with each possibly being of infinte size, typical comparison algorithms find pairs of similar search spaces in $O(n^2)$ time. This can become quite expensive if the number of search spaces increase dramatically. Thus, the aim of this work is to extend a technique known as Locality Sensitive Hashing that uses a MinHash based algorithm to reduce the comparison time to O(n).

Keywords:- Inductive Logic Programming, Search spaces, Locality Sensitive Hashing, MinHashing.

ACKNOWLEDGMENTS

I would like to sincerely thank Prof. Ashwin Srinivasan for providing me this opportunity to work on a project under him. Apart from the thorough and focused guidance by him, the stimulating and encouraging meetings we had were a constant source of motivation and knowledge that will inspire and motivate me for the years to come.

1. INTRODUCTION

Inductive Logic Programming (ILP) based systems are often used to generate clauses or hypotheses that satisfy a given example and some background knowledge. For eg:

father(harin,kathan).
father(gautam,harin).

For the above given background knowledge and a positive example of the form "***grandfather(gautam,kathan)"*** an ILP based system can predict that the clause ***"grandfather(A,B):-father(A,C), father(C,B)."*** satisfies the given background information and the positive example. A collection of such clauses is called a search space. Given more such data and postitive and negative examples, many such clauses can be generated and their relative performance measured by how many positive examples they satisfy.

Aleph (**A** **L**earning **E**ngine for **P**roposing **H**ypotheses) is one such ILP based system that generated such clauses given some background knowledge and postive and negative examples and has been extensively used in this work for search space generation.

However, with the increase in amount of data supplied the number of such clauses generated can increase rapidly and sometimes even go to an infinite size. It generally so happens that many search spaces are quite similar to each other and thus traversing all of them can be quite expensive. Thus it would be beneficial if it can be somehow computed if two search spaces are similar or not and thus extra-polate the results of one for the other. Traditional comparison algorithms take $O(n^2)$ time and it can be quite expensive if there are a large number of such infinite sized search spaced. The problem is two fold. Not only are the search spaces of infinite size but there are a large number of such search spaces as well. Thus in order to get significant imporvement in the computation, it is necessary to improve upon both of these parameters.

Jaccard Coefficient is a similarity based metric that is used to calculate similarity between two entities like documents, web pages etc. It is a simple forumlae given by:

$$J = \frac{A \cap B}{A \cup B}$$

Where J is the Jaccard Coefficient of two sets A and B. However, calculating this score for search spaces would lead to no improvement in performance. MinHashing is a hashing based technique that has been mathematically been proven to be an unbiased estimator of the Jaccard Coefficient. The core idea behing the MinHash technique is to reduce the number of features representing a particular entity.

Typically, a size in the multiples of 100 is chosen depending on how many number of features are there. For example, if we have a document which is represented by 10000 features with each feature being a word present in that document, MinHashing helps reduce the feature space to mere 100 or 200 features. This solves two problems:

(1) Entities of different sizes can be represented by a fixed length feature vector.
(2) It helps reduce the dimension of entities (search spaces here) from an infinite size to just 100-200 features.

MinHashing typically involves choosing distinct hash functions equal to the number of features that we want in the reduced space. Value of each of those features is equal to the minimum of the hash value of each feature in the original feature space when passed through the respective hash function. Here, n is the original feature space size.

$$MinHash_i = Minimum\left(HashFunction_i\left(Feature_j\right)\right) \forall\ j \in [1, n]$$

Now, in order to determine similar entities, it is still necessary to make roughly n^2 comparisons. Locality Sensitive Hashing (LSH) reduces this time to an order of n.

Consider an example. We have a matrix of order m*n where m is the number of entities and n is the number of hash functions choosen. Let n be 100. LSH divides this matrix into bands with each band having a give number of hash functions. Let us divide the 100 hash functions into 5 bands of 20 hash functions each. LSH further hashes these 20 hash fuctions for each entity in our dataset into buckets. The core idea revolves around the fact that similar documents will hash to the same buckets. If they don't hash to the same bucket, they still have 4 other bands to hash to a similar bucket and thus be pairs.

Thus during the process, the entire corpus of entities is iterated through just once and such buckets are created. Entities belonging to the same buckets are considered to be candidate pairs for being similar and thus only these entities are compared in full. Whenever a new entity arrives, we minhash it and then further hash it to buckets and find similar documents from the bucket it hashes to.

Thus, in a nutshell, LSH coupled with MinHashing reduces the two problems described above and reduces the complexity of the problem from $O(n^2)$ to $O(n)$.

The rest of the works if presented as follows. **Sections 2** talks about the experimental set up and a brief descriptions of the various experiments that have been run. **Section 3** consists of the results for the above mentioned experiments with the last section talking about the conclusions and possible future research work.

2. EXPERIMENTAL DESIGN

2.1 Hashing search spaces:

The problem taken for this work is the "Trains Problem" defined by R.S Michalski in his paper titled **"Inductive Inference of VL Decision Rules"**[1]. Basically, there are eastbound trains (positive examples) and westbound trains (negative examples) and each train has a set of carriages that it pulls. Each carriage has a different design and carries different loads. The aim of the problem is to determine rules/clauses that correctly help define trains that are going east. The data[2] for this work includes the code for background knowledge and positive and negative examples files.

Alpeh[3] has been used to generate the search spaces satisfying each of the given positive examples. Using python, the required clauses are extracted out of the search space and another Prolog code converts these search spaces into hash values which helps in the easy implementation of the LSH based algorithm. At the end of this stage, we have search spaces as a collection of hash values of the clauses. The MinHash and LSH techniques are then applied on these search spaces.

2.2 Preliminary Experiments:

The preliminary experiments involved writing and testing the MinHashing code. Sample pairs of data were generated using random numbers with varying true Jaccard Similiarity and the MinHash Jaccard values were generated and compared to the true values.

2.3 LSH on a set of four trains:

After running the MinHash and LSH code for random data successfully, a set of four trains as provided by Michalski was taken as the positive example and the MinHashing and LSH done to compare results to actual Jaccard similarity. Candidate pairs, if present, were also checked for.

2.4 LSH on a set of 125,000 trains:

Currently under work. Aim to extend the idea in 2.3 to see if there is substantial improvement in execution time and wether the results are actually in line with the true similarity values. Running experiments on just 100 trains due to time contraint and lack of compute power.

3. RESULTS

3.1 Results for preliminary experiments:

	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
5000	[0.0, 'No']	[0.07, 'No']	[0.18, 'No']	[0.31, 'No']	[0.36, 'No']	[0.49, 'No']	[0.58, 'No']	[0.73, 'No']	[0.77, 'No']	[0.93, 'Yes']	[1.0, 'Yes']
10000	[0.0, 'No']	[0.08, 'No']	[0.22, 'No']	[0.3, 'No']	[0.39, 'No']	[0.47, 'Yes']	[0.55, 'No']	[0.6, 'No']	[0.78, 'No']	[0.86, 'Yes']	[1.0, 'Yes']
15000	[0.0, 'No']	[0.06, 'No']	[0.19, 'No']	[0.25, 'No']	[0.38, 'No']	[0.45, 'No']	[0.66, 'No']	[0.71, 'No']	[0.75, 'No']	[0.91, 'No']	[1.0, 'Yes']
20000	[0.0, 'No']	[0.07, 'No']	[0.19, 'No']	[0.25, 'No']	[0.36, 'No']	[0.48, 'No']	[0.59, 'No']	[0.66, 'No']	[0.89, 'Yes']	[0.9, 'Yes']	[1.0, 'Yes']

The above results are for randomly generated dataset with varying true Jaccard Similarity columnwise. Different sized datasets are taken to see if results improve with a larger dataset or not from a range of 5000 elements to 20,000 elements. Values in each block are of the form [A,B] where A indicates the MinHash Jaccard values and B is a binary variable. 'Yes' indiactes that two datasets of a given size are candidate pairs or not.

The results show that the MinHash Jaccard values are quite close to the true Jaccard values thereby reaffirming the fact that MinHash is indeed an unbiased estimator of the true Jaccard values. Apart from that the LSH algorithms correctly predicts two sets to be candidate pairs if their MinHash Jaccard is greater than or equal to 0.9 barring some wrong results in between.

3.2 LSH for a set of four trains:

	1	2	3	4
1	[1.0,1.0]	[0.0195,0.04]	[0.04,0.06]	[0.089,0.09]
2	[0.0195,0.04]	[1.0,1.0]	[0.013,0.01]	[0.067,0.13]
3	[0.04,0.06]	[0.013,0.01]	[1.0,1.0]	[0.026,0.03]
4	[0.089,0.09]	[0.067,0.13]	[0.026,0.03]	[1.0,1.0]

After applying the LSH and MinHash algorithms to the search spaces that were processed in Prolog to generate the hash-values, follwing results are obtained. Each block in the matrix is of form [A,B] where A is the MinHash Jaccard values and B is the true Jaccard values. It can be seen that the MinHash Jaccard values emulate the true Jaccard values and thus can be used as a primary step before applying the LSH algorithms. However, since the trains are quite dissimilar, none of the 4 trains are candidate pairs according to the LSH algorithm.

3.3 LSH for 125K trains:

Currently extending the above model to incorporate 1,25,000 such trains to check for improvement in execution time. Working on just 100 trains due to lack of compute power.

4. CONCLUSIONS

The observed results are sufficient to show that the LSH algorithm not only emulates the true Jaccard Similarity between search spaces, but also can be used to find similar search spaces with improved efficiency as compared to the other comparative techniques. Incorporating a large set of trains to actually test this hypothesis is currently under progress.

Possible future work can revolve around finding appropriate hash functions for the MinHashing and LSH algorithms to further improve the ability of model to categorize between similar and dissimilar search spaces. Apart from that, this model can be incorporated in various ILP algorithms to improve their execution performance.

REFERENCES

[1] Michalski, R. and Larson, J. (1977). Inductive Inference of VL Decision Rules.

[2] https://github.com/friguzzi/aleph

[3] http://www.cs.ox.ac.uk/activities/machinelearning/Aleph/aleph