ISSN (Print) 2313-4410, ISSN (Online) 2313-4402

http://asrjetsjournal.org/

Closest Match Based Information Retrieval and Recommendation Engine using Signature-Trees and Fuzzy Relevance Sorting

Ali Sohani[a], Rafi Ullah[b] *, Athaul Rai[c], Owais Karni[d]

[a,b,c,d] *Data Science Department, Cubix.co*

[a]*Email: ali.sohani@cubix.co*

[b]*Email: rafiullah.khan@cubixlabs.com*

[c]*Email: athaulrai@cubixlabs.com*

[d]*Email: owais.karni@cubixlabs.com*

Abstract

This paper proposes a recommendation technique to avoid exhaustive search to be ran on the database with thousands of records, before coming to a conclusion or inference, where it can be said that recommended thing is matching up to a significant percentage of what was initially desired. Often such searches involve not just the simple full-match search based on indexes, but also the partial or nearby match searches where which percentage of match between entities is relevant enough for ultimate recommendation. Usually these problems are tackled by various methods like Fuzzy operations, Reg-Ex searches, Clustering, Similarity Analysis each having its own set of effectiveness as well as efficiency. Our goal here was to create a search and recommendation system which can perform fuzzy-search and fuzzy-similarity-analysis with near-match percentages in an effective, efficient as well as user-friendly manner on thousands of records/ files/ rows with 100s of attributes/ features/ columns. Inspired from Google's Image Searching Algorithm, that search on the basis of signatures based on feature-extraction from each image, we have created Match engine, that read schema of data or files, compiles encoded signature and store them as an index. That index is then converted into a tree (S-Tree), on the basis of relevance of each field/ column and data frequency observed. After compilation done, system can now search and recommendation of best matches in very efficient manner. For further optimization we use heuristics like dividing feature sets into hard-filters and soft-filters, former demands full match and later demands fuzzy match. On arriving even one best match, we can retrieve other matches without searching.

---

* Corresponding author.

Our technique though not that modern and actually inspired, but based on ensemble methods used to provide fast and efficient results. We have proved quicker than full scan searches. In future we plan to make signature comparison engine on variety of advanced data types of features like Geo-coordinates and synonyms. And storing compiled signatures trees into distributed database/grid, query will run concurrently to match the results, or signatures passing through machine learning techniques. Currently system used for recipe recommendation and in future this will be used in applications like dating system's, film and music recommendation.

Keywords: Signature tree matching; Recommendation system's; fuzzy recommendation system; Fuzzy Relevance Sorting.

1. Introduction

Problems of recommendation often require an exhaustive search (search through complete solution space) to be ran on the database with thousands/millions of records, before coming to a conclusion or inference, where it can be said that recommended thing is matching up to a significant percentage of what was initially desired. Often such searches involve not just the simple full-match search based on indexes, but also the partial or nearby match searches where which percentage of match between entities is relevant enough for ultimate recommendation. Usually such problems of matching and recommendations are tackled by various methods like Fuzzy operations, Reg-Ex (regular expression) searches, Clustering, Similarity Analysis each having its own set of effectiveness as well as efficiency.

Our goal here was to create a search system and recommendation system which can perform fuzzy-search and fuzzy-similarity-analysis with near-match percentages in an effective, efficient as well as user-friendly manner on thousands of records/ files/ rows with 100s of attributes/ features/ columns. In searching each of the attributes has their owns importances or weight-age. Each of these having individual score and contributes to global score. May can diminished or enhance the importance of attributes/feature/columns as per requirements.

Inspired from Google image searching based on signatures of images (features of image), we have created Match-Engine. Match Engine created signature tree (Sig-Tree) after creating encoded signatures. Sig-tree is created on the basis of relevance of each filed or column and data frequency observed. After compilation, system can now make search and recommendation. Query is converted into encoded signature and then find best match in compiled signatures.

We have also optimize technique by dividing features into hard filters and soft filters, former used for strict match I-e 100% and later used for fuzzy match I-e (0%-100%). Using hard filters we traverse to bins or clusters, then perform fuzzy match on soft-filters. To make searches faster we have used match-sort and match-drop techniques. Hence if we have got even a one successful match crossing a threshold, then based on compiled index of signature similarity analysis, we will be able to retrieve similar signatures. Proposed techniques proves efficient in searching and recommendation in huge solution space, retrieval is quicker than full scan searches.

Rest of the paper discussed the Related work, Similarity measures, signature compilation, match-sort and match-drop technique, proposed methodology, Results, Conclusion, Future work.

2. Related work

Multi-level signatures files can be constructed to support relational queries having many AND predicates and generic text queries in databases [4].

Signature file used as filtering mechanism to reduce the query text when searching, but the signature files is searched exhaustively which degrade performance. In [6] they have proposed deterministic algorithms to get rid of such exhaustively search of signature file.

S-tree technique is used in [7] for the efficient organization of fuzzy signatures. Used content-based image retrieval system. Images from the dataset are described using a fuzzy set of features.

For set of d-dimensional tuples or sets having textual descriptions, a technique is used to facilitate complex queries. And efficient algorithm is used names Keyword-Matched Skyline search is proposed that uses IR^2-tree as an index structure. To retrieve query it uses nearest neighbor search in the branch and information of node. Technique has been proven very effective and efficient in term of combinatorial cost and Input-Output cost [8].

Hierarchical tree is used to clustering fuzzy data (they named it ET i-e extensional tree). Proposed technique in [11] has been compared with some of the algorithm used in literature review.

3. Similarity Measures

For find similarity between two data types, we can use different similarity measures mainly depend upon the type of data type. For example one can use simple subtraction to find similarity between two integers or float types, this measure cannot be use in case of tow strings or sets or dates. Some of the similarity measures found in literature are

A. Jaccard Similarity

This measure is usually used for finding similarity between two sets. Let suppose “A” and “B” are two sets, then similarity score between “A” and “B” is given by

$$\text{Similarity}(A, B) = (\| A \text{ intersection } B \|) / (\| A \text{ Union } B \|) \qquad (1)$$

B. Cosine similarity

“A” and “B” are two vectors, then similarity between “A” and “B” given by

$$\text{Similarity}(A, B) = \cos(Q) = (\| A.B \|) / (\| A \| . \| B \|) \qquad (2)$$

C. Euclidean distance

Euclidean distance is the measure usually for finding similarity between two vectors or points. Distance between two vectors “A” and “B” given by

$$\text{Distance (A, B)} = \text{sqrt}((x2 - x1)^2 + (y2 - y1)^2) \quad (3)$$

$$\text{Similarity (A, B)} = 1 / \text{Distance(A, B)} \quad (4)$$

D. Manhattan distance

Manhattan distance between two points (or vectors) given by

$$\text{Distance (A, B)} = | x2 - x1 | + |y2 - y1 | \quad (5)$$

Some of the techniques used for string matching used for many years are Levenshtein distance formula, Damerau- Levenshtein, Hamming distance, Jaro-Winkler and strike a match.

4. Proposed Methodology

Proposed technique is consist of two modules

1: Making Signature tree and signature files

2: Records retrieval and recommendations

Module 1 is a one time process for setup of the algorithm, it will only be recalled when underlying data has caught new feature. Module 2 is a recurring one, will be used time and when Match operation is called given input parameters.

E. Making Signature tree and signature files

This is one time process, user have to given either configuration data about data i.e. hard filters and soft filters, minimum, maximum of the values etc.

If user will not provide, system is capable to do that. The setup algorithm globally analyzes the whole data in form of stats, like for unique values and data-type detection, it takes time depending up on the data size but eventually arrives at conclusions like based on typed values, what features can be considered hard/ categorical and features that can be considered soft/ quantitative. Here the signature tree is created in some similar fashion like decision tree.

Sig-tree levels and signatures in each cluster i.e. leaves of tree mainly depend upon the hard filters found in data. There will be as many levels as number of hard filters. And the number of children at each node depends upon the product unique values of each hard filters.

A. Records retrieval and recommendations

After Sig-tree is created now we can make queries for searching and recommendations. User query must be in same format as signatures. Apply hard filters first for strict match and then soft filters for fuzzy match.

Algorithm :

1. *Count unique values* in *each column/attributes*
2. *for all columns*
   ……*if count* (*unique column values*)<*filter threshold*
   ……*keep column* in *hard filters*
   ……*else*
   ……*keep column* in *soft filters*
3. *sort hard filters count values* in *ascending order*
4. *make combinations of hard filters*
5. *place signatures* in *each combination* (*cluster*)
   if it contains hard− filter values
6. *for all clusters*
   ……*Encode feilds of records* to *make signatures*
   ……*Apply vertical sort* in *cluster*
   ……*Apply horizontal sort* in *cluster*
7. *Sig− tree is complete*

Figure 1: Algorithm for Making Signature tree and signature files

Algorithm :

1. *Arrange user query* in *same format as signature*
2. *Apply hard− filters* one *by* one *go* to *specific cluster*
3. *for signature* in *cluster / signaturefile*
   ……*for each component* in *signature*
   ……$score_{total} = 0$
   ……$score_{remaining} = 100$
   ……$if sign_{component}$ *not match* $query_{component}$
   ……$score_{remaining} = score_{remaining} - component_{weight}$
   ……$if\ score_{total} + score_{remaining} < threshold$
   ……*break*
   ……*else*
   ……*continue*
   ……$if\ sign_{component}$ *matched* $query_{component}$
   ……$score_{total} = score_{total} + component_{weight}$
   ……*if score≥ threshold*
   ……*store this record* in $matched_{records}$
   ……*break*
   ……*break*
4. $return matched_{records}$

Figure 2: Algorithm for Records retrieval and recommendations

The very first step of our proposed technique is signature compilation. When a thousand or millions of records. System automatically read the schema of data or files and try to extract the data types of each field. Different statistics like minimum, maximum etc are calculated, all the values are set in some standard format, data fields are encoded to achieve fast searching, hard filters and soft filters are decided from the nature of data. Or the

same configuration can be inputed to system along with data. Consider following data records,

Table 1: An Example dataset

Name	Age	Degree	subjects	Experience	Gender
Ali	38	MBA	(AI, DS)	16	Male
Salman	37	MBA	(DS)	19	Male
Urooj	29	MBA	(OS, AI)	10	Female
Rafi	28	BSCS	(AI, OS)	2	Male
Faraz	27	BSCS	(OS)	3	Male
Athual	25	BSCS	(DS,OS)	1	Male

System encode the records for fast searching and matching. The proposed system detect the data types and encode the fields. After encoding records will be something like

Encoded labels are stored as JSON file for later use when

{ “Degree” : { “BSCS” : 1, “MBA” : 2}, “Gender” : {“male” : 1, “female” : 2}}

Encoded records (Each record is actually a signature)

Table 2: Encoded dataset given in table 1

Name	Age	Degree	subjects	Experience	Gender
1	38	2	(1, 2)	16	1
2	37	2	(2)	19	1
3	29	2	(3, 1)	10	2
4	28	1	(1, 3)	2	1
5	27	1	(3)	3	1
6	25	1	(2,3)	1	2

Now count the unique values in each of the column or attributes and the data types.

Count(Name) = 6

Count(Age) = 6

Count(Degree) = 2

Count(subjects) = 6

Count(Experience) = 6

Count(Gender) = 2

Attributes having count less than threshold (we have 5) are considered as hard filters and count greater than 5 are considered as soft-filters.

Take minimum of the count and split data like decision tree. Data will split into two partition (splitting here is dynamic).

Gender = {male, 1}

Table 3: Cluster of male records

Name	Age	Degree	subjects	Experience	Gender
1	38	2	(1, 2)	16	1
2	37	2	(2)	19	1
4	28	1	(1, 3)	2	1
5	27	1	(3)	3	1

Gender = {female, 2}

Table 4: Cluster of female records

Name	Age	Degree	subjects	Experience	Gender
3	29	2	(3, 1)	10	2
6	25	1	(2,3)	1	2

Then iteratively split records in each of the partitions above with second minimum count, that is Degree. Repeat same process for all hard filters and make separate signature file.

Gender = 1(male) and Degree = 1(MBA)

Table 5: Male and MBA cluster

Name	Age	Degree	subjects	Experience	Gender
4	28	1	(1, 3)	2	1
5	27	1	(3)	3	1

Gender = 1(male) and Degree = 2(BSCS)

Table 6: male and BSCS cluster

Name	Age	Degree	subjects	Experience	Gender
1	38	2	(1, 2)	16	1
2	37	2	(2)	19	1

Gender = 2(female) and Degree = 1(MBA)

Table 7: female and MBA records

Name	Age	Degree	subjects	Experience	Gender
6	25	1	(2,3)	1	2

Gender = 2(female) and Degree = 2(BSCS)

Table 8: female and BSCS records

Name	Age	Degree	subjects	Experience	Gender
3	29	2	(3, 1)	10	2

After all hard-filters signature will be split into different partition, place those signatures in their corresponding files. These files can be viewed as a decision tree when we will be retrieving and matching records. Sig-tree for

above example

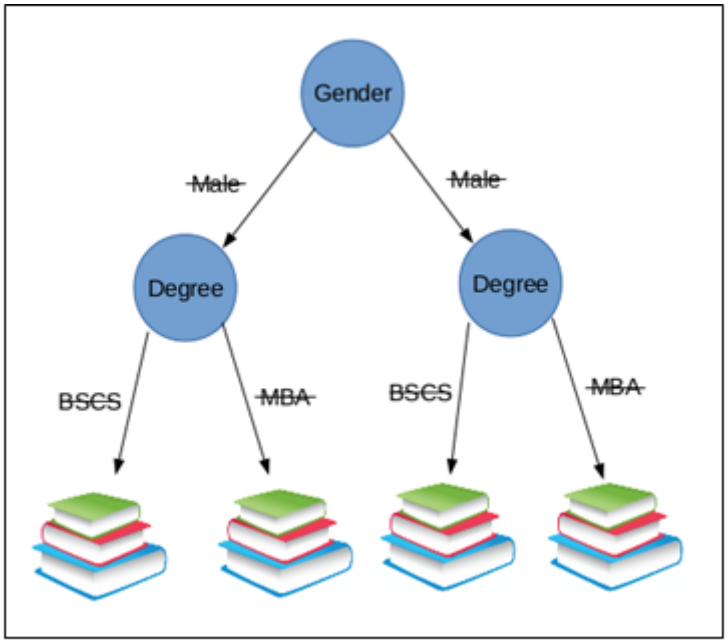

Figure 3: Making clusters from dataset

This calculations can be avoid if we give configuration file at the time of compiling signatures.

After records encoding, identifying hard-filters, place signature placement in their corresponding signature files. Then apply vertical-sort and horizontal sort with respect to weight on each signature file or cluster.

These are the compilation steps.

On user query rearrange user query in same format as signature, then go to that specific signature files with the help of hard-filters. And once we reach that specific signature file, match user query attributes with signature one by one using similarity measure depending on the type of data. Apply match-drop technique here to decide whether to go ahead or not? On finding signature that pass the threshold score. Then records lies near this signature will be recommended with even comparing and checking.

Similarity measures, we have used for

We use following formula for matching integers or float values

$$Si = (upper\_limit - signature\_value) / (upper\_limit - lower\_limit) \quad (6)$$

$$Q = (upper\_limit - user\_value) / (upper\_limit - lower\_limit) \quad (7)$$

$$Result = | Si – Q | \quad (8)$$

$$Result = Result * fuzzyWeight \quad (9)$$

For set or we have Jaccard similarity, or we can use cosine similarity as well. For string if not encoded, we use cosine similarity.

5. Hard and Soft Filters

To further optimize it we use heuristics like dividing feature-sets into hard-filters and soft-filters, former demanding full-match and later one demanding the fuzzy ones. All soft-filters are granted relevance score, % weight-age.

Algorithm that traverses the tree, traverses it on the basis of most relevant features first (hard filters), jumping directly into the bin/ tree cluster where nodes of specific relevance are located per index. User query is compare against signature using hard filters. These hard filters must be fulfilled to go signature file, otherwise signature file or bin will not be selected.

Once we reach signature file, attributes that are soft filters are compared as fuzzy search. Each attributes has its own weight. Then values in user query and signature are matched and multiplies by their corresponding weights or fuzzy values.

$$Result = weight * (query\_1 ** signature\_1) \quad (10)$$

Weight is the corresponding fuzzy value and ** is the similarity measures we used. Then final score is the sum of individual score of each component of signature.

$$Score\_final = Score\_individual1 + Score\_individual2 + \ldots\ldots \quad (11)$$

Once we found a signature that score pass the threshold value , we can recommend similar signature making not just exhaustive fuzzy-search but even fuzzy recommendations easier in result.

6. Match-sort and Match-drop

Match-sort and Match-drop are the two optimization techniques used for fast and efficient records retrieval. Signatures in every cluster or bin is sorted with respect to weight-age of each attribute or components of signature. By default weights of all attributes are equal. The heuristics behind this technique is to reduced processing time.

B. Match-Sort:

Signatures in each cluster/bin/file are sorted I-e vertical sort. Consider following are signatures in the any of the clustering

$signature_1 = A_{11} \mid A_{12} \mid A_{13} \mid A_{14} \mid A_{15}$

$signature_2 = A_{21} \mid A_{22} \mid A_{23} \mid A_{24} \mid A_{25}$

$signature_3 = A_{31} \mid A_{32} \mid A_{33} \mid A_{44} \mid A_{45}$

$signature_4 = A_{41} \mid A_{42} \mid A_{43} \mid A_{44} \mid A_{45}$

$signature_5 = A_{51} \mid A_{52} \mid A_{53} \mid A_{54} \mid A_{55}$

Compare all signatures with first signature-tree, consider following scores with $signature_1$

score { $signature_1$, $signature_2$} = 0.68

score { $signature_1$, $signature_3$} = 0.75

score { $signature_1$, $signature_4$} = 0.98

score { $signature_1$, $signature_5$} = 0.84

Sort signatures in descending order with respect to score against signature 1. Sort cluster file will be.

$signature_1 = A_{11} \mid A_{12} \mid A_{13} \mid A_{14} \mid A_{15}$

$signature_4 = A_{41} \mid A_{42} \mid A_{43} \mid A_{44} \mid A_{45}$

$signature_5 = A_{51} \mid A_{52} \mid A_{53} \mid A_{54} \mid A_{55}$

$signature_3 = A_{31} \mid A_{32} \mid A_{33} \mid A_{44} \mid A_{45}$

$signature_2 = A_{21} \mid A_{22} \mid A_{23} \mid A_{24} \mid A_{25}$

All similar signatures are clustered together using simple sorting. Now it's time to apply Match-sort (Horizontal)

Following is the one of the signature from cluster or bin

$Compiled\_Signature = Attr_1 \mid Attr_2 \mid Attr_3 \mid Attr_4 \mid Attr_5$

Weights = [0.3, 0.1, 0.4, 0.1, 0.1]

Sort each of the compiled signature in each cluster or bin w.r.t to weigh-age given.

$Sorted\_Signature = Attr_3 \mid Attr_1 \mid Attr_2 \mid Attr_4 \mid Attr_5$

After Match-sort (both vertical and horizontal), signatures in the file cluster are now sorted and well structured.

If the matching threshold is let suppose 70%, and following query occurs

UserQuery = A_1 | A_2 | A_3 | A_4 | A_5

C. Match-Drop Case 1:

If Attribute 3 ($Attr_3$ and A_3) didn't matched, and all other attributes are same. Proposed technique will check Attribute 3 first it didn't match and 40% score is lost, Now remaining score is 60% in overall signature we can achieve, this score I-e 60% plus score achieved is less than threshold (70%), simply drop this signature without checking other attributes. This is faster the retrieval process. Condition for dropping signature is

$$Score\_achieved + Score\_remaining < threshold \quad (12)$$

D. Match-Drop Case 2:

If Attribute 3 ($Attr_3$ and A_3) and Attribute 1 matched, and all other attributes are different.

Proposed technique will check Attribute 3 first it matched and 40% score is gained, Now remaining score is 60% in overall signature, this score I-e 60% plus achieved score 40% is not less than threshold (70%), so continue matching second attribute.

Now check Attribute 1, this also matched, score achieved is 70% (40% + 30%), so stop matching and this signature is candidate match against query. No need to compare other parts of signature. This also reduced processing time. Criteria for match-stop (stop comparing same signature) further is

$$Score\_achieved + Score\_remaining >= threshold \quad (13)$$

7. Results

Our techniques though not that modern and actually inspired, but based on ensemble methods used prove to provide fast and efficient results. They prove retrieval is quicker than the full-scan searches. Here the proposed technique is not traversing the whole search space for best match. Match-sort and Match-Drop enable us to detect whether a record is good match or not early with out spending time on comparison. We have proved proposed technique is efficient and effective in a very large data records. We have test our proposed technique on different sizes of data.

Experiment is done on computer system having following specifications

Operating System : Ubuntu 16.04

RAM : 8 GB

Hard Drive : 150 GB

Intel core i5

Table 9: Different statistics related to algorithm

	Data Size (No of Records)	
	1000	100000
Sig-Tree creation time	3.2559 seconds	413.2589 seconds
Searching in Sig-Tree	0.025 seconds	3.0235 seconds
Number of clusters or Sig-Files	Product of uniques values of hard-filters	Product of uniques values of hard-filters
Number of attributes	7	10
Secondary Memory required	59.8 KB	6.2 MB
Number of levels in tree	3	4

The number of Sig-Files or clusters equals to the product of hard-filters unique values.

Worst case Searching time T = O(h+n), Where "h" are number of hard-filters and "n" is the number of records in that specific clusters or bin.

8. Conclusion

We have successfully implement our proposed technique and prove to be efficient and effective methods in a very large collection of data records. The encoding i-e making signatures reduced the processing time. The only operation that is time consuming is making signature tree and signature file (signatures in the signature files are vertically and horizontally sorted), but this is one time operation. Records retrieval is very fast in Sig-tree. Match-drop and Match-sort techniques further optimize the retrieval, hence retrieval in a very large or huge records storage is very fast. The only problem we observed was, that this algorithm works perfect in case of records having at least some discrete data. It cannot works in case of only continuous data. There at-least must be some of the discrete fields in dataset to take advantages of proposed system.

9. Future Work

In future plan is to make signature comparison engine able to compare on the variety of advanced data-types of features, like not just string, number, date but also Geo-coordinates and synonyms. Next in line is to do a distributed query on the signatures, by splitting compiled signature trees into sub-trees and then storing them into a distributed database/ grid, query will run concurrently to match the results. Work after that will be passing the compiled signatures through a machine learning method, and then train it to recognize patterns in signatures, so that based on partial signatures prediction of items that may append in signature can happen. This can let us know what items in the signature have most likelihood to go along with. This will also improve recommendations by 10 folds. We have already used current system on live service used for Recipe Recommendation, next this will be tried on a Dating System, Film and Music Recommendation system, where on the basis of few selected preferences and interests, best matching profiles/ media content will be recommended respectively.

Acknowledgment

I am a strong believer of fact, that man needs a solid team in surrounding to achieve great results. And that the difference between good and great is in the capability of your team. Thanks to my team, who has worked with me to follow the vision and lead, also for them to bear my many requests of making different suggested updates in algorithm in pursuit of accuracy and performance. I am very thankful for their respect and hard-work.

References

[1] Deppisch, Uwe. "S-tree: a dynamic balanced signature index for office retrieval." In Proceedings of the 9th annual international ACM SIGIR conference on Research and development in information retrieval, pp. 77-87. ACM, 1986.

[2] Frakes, William B., and Ricardo Baeza-Yates. "Information retrieval: data structures and algorithms." (1992).

[3] Faloutsos, Chris, and Stavros Christodoulakis. "Signature files: An access method for documents and its analytical performance evaluation." ACM Transactions on Information Systems (TOIS) 2, no. 4 (1984): 267-288.

[4] Chang, Walter W., and Hans-Jörg Schek. A signature access method for the Starburst database system. IBM Thomas J. Watson Research Division, 1989.

[5] Zezula, Pavel, Fausto Rabitti, and Paolo Tiberio. "Dynamic partitioning of signature files." ACM Transactions on Information Systems (TOIS) 9, no. 4 (1991): 336-367.

[6] Lee, Dik Lun, and Chun-Wu Leng. "Partitioned signature files: Design issues and performance evaluation." ACM Transactions on Information Systems (TOIS) 7, no. 2 (1989): 158-180.

[7] Snasel, Vaclav, Zdenek Horak, Milos Kudelka, and Ajith Abraham. "Fuzzy signatures organized using S-Tree." In Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on, pp. 633-637. IEEE, 2011.

[8] Choi, Hyunsik, HaRim Jung, Ki Yong Lee, and Yon Dohn Chung. "Skyline queries on keyword-matched data." Information Sciences 232 (2013): 449-463.

[9] Olaru, Cristina, and Louis Wehenkel. "A complete fuzzy decision tree technique." Fuzzy sets and systems 138, no. 2 (2003): 221-254.

[10] https://content.iospress.com/articles/journal-of-intelligent-and-fuzzy-systems/ifs710

[11] Yazdi, Hadi Sadoghi, Mohammad GhasemiGol, Sohrab Effati, Azam Jiriani, and Reza Monsefi. "Hierarchical tree clustering of fuzzy number." Journal of Intelligent & Fuzzy Systems 26, no. 2 (2014): 541-550.

[12] De Felipe, Ian, Vagelis Hristidis, and Naphtali Rishe. "Keyword search on spatial databases." In Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on, pp. 656-665. IEEE, 2008.

[13] Faloutsos, Christos. "Signature-based text retrieval methods: A survey." IEEE Data Eng. Bull. 13, no. 1 (1990): 25-32.

[14] Faloutsos, Christos. "Signature Files." (1992): 44-65.

[15] Chen, Yangjun. "Signature files and signature trees." Information Processing Letters 82, no. 4 (2002): 213-221.

[16] Helmer, Sven. "Evaluating different approaches for indexing fuzzy sets." Fuzzy Sets and Systems 140, no. 1 (2003): 167-182.

[17] Tousidou, Eleni, Alex Nanopoulos, and Yannis Manolopoulos. "Improved methods for signature-tree construction." The Computer Journal 43, no. 4 (2000): 301-314.

[18] Lee, Dik Lun, and Chun-Wu Leng. "A partitioned signature file structure for multiattribute and text retrieval." In Data Engineering, 1990. Proceedings. Sixth International Conference on, pp. 389-396. IEEE, 1990.

[19] Tousidou, Eleni, Panayiotis Bozanis, and Yannis Manolopoulos. "Signature-based structures for objects with set-valued attributes." Information Systems 27, no. 2 (2002): 93-121.