ISSN:2249-5789
Prabhjot Singh et al , International Journal of Computer Science & Communication Networks,Vol 5(2),107-114

Implementation of Support Vector Machine based ranking metrics

Narina Thakur
Associate Professor,
Bharati Vidyapeeth's College of Engineering
Computer Science Department,
New Delhi, INDIA
narina.thakur@bharatividyapeeth.edu

Prabhjot Singh
Student,
Bharati Vidyapeeth's College of Engineering
Computer Science Department
New Delhi, INDIA
prabh.me@outlook.com

Sumit Dhawan
Student,
Bharati Vidyapeeth's College of Engineering
Computer Science Department,
New Delhi, INDIA
sdhawan999@gmail.com

Shubham Agarwal
Student
Bharati Vidyapeeth's College of Engineering
Computer Science Department
New Delhi, INDIA
shubhamagar05@gmail.com

***Abstract*—** This paper discusses the implementation of Support Vector Machine (SVM) based ranking metrics to compute the similarity and rank the documents based on a user query using Fuzzy Logic. TREC dataset has been used for the same purpose. The dataset is parsed to generate keywords index which is used for the similarity comparison with the user query and ranking is done using SVM. Each query is assigned a score value based on its fuzzy similarity with the index keywords. The performance and accuracy of the proposed retrieval model is compared with TF/IDF similarity model using Precision-Recall curves. The SVM ranking precision and recall are calculated by classifying the documents based on a threshold value by using SVM alpha measures.

***Index Terms*—**Fuzzy Logic, Information Retrieval, Ranking, Similarity, Precision, Recall, Lucene, SVM.

I. INTRODUCTION

An information retrieval system stores and indexes documents such that when users express their information need in a query the system retrieves the related documents associating a score to each one. The higher the score the greater is the importance of the document. Usually an information retrieval system returns large result sets and the users must spend considerable time until they find the items that are actually relevant. Moreover, documents are retrieved when they contain the index terms specified in the queries. However, this approach will neglect other relevant documents that do not contain the index terms specified in the user's queries. When working with specific domain knowledge this problem can be overcome by incorporating a knowledge base which depicts the relationships between index terms into the existing information retrieval systems[1].

To deal with the vagueness typical of human knowledge, the fuzzy set theory can be used to manipulate the knowledge in the bases. The expectation is that the indexed terms can improve the quality of retrieved documents bringing the most relevant and more semantically related to the initial query. Full-text search is still the most popular form of search and is very useful to retrieve documents for which we know the keywords to search for. Indeed, full-text search is not suitable for finding relevant documents about a specific topic in the context of a given task.

Another problem relies in the huge number of retrieved documents/data. It is difficult for a user to deal with thousands of electronic documents. The techniques often used by search engines are based on statistical methods and do not permit to take account of the semantics contained in the user's query as well as in the documents. Some approaches have been developed to extract semantics and so, to better answer user queries. On the other hand, most of these techniques have been designed to be applied to the whole web and not on a particular domain[2].

Apache Lucene has been used to retrieve the relevant documents. Lucene is a text search engine library suitable for nearly any application that requires full-text search. Lucene is composed of many text processing tools. Each tool within it is a heuristic, an algorithmic shortcut in lieu of true linguistic comprehension[3].

Lucene performs search in two steps. First, Lucene searches for tokens stored in the database that are similar to the query tokens. To determine if tokens are similar, Lucene computes an edit distance (also referred to as a Levenshtein Distance) from the query tokens to the

tokens stored in the database. Second, Lucene uses the similar tokens it finds as new query tokens to retrieve relevant documents[4].

This project implements a Support Vector Machine (SVM) based Ranking system using fuzzy similarity measures. The project has been divided into 3 phases:

1) Fuzzy logic based similarity measure implementation on TREC.
2) Implementation of SVM ranking by using the output of Fuzzy Similarity as training data.
3) Performance evaluation by calculating precision-recall values.

II. Information Retrieval Model

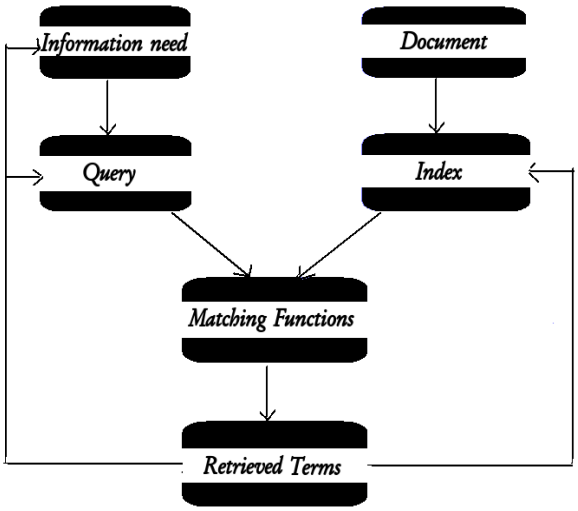

Fig. 1. Visualization of Information Retrieval system process.

A. Issues and solutions

Traditional IR systems assume that a user's queries can precisely be characterized by the index terms. However, this assumption is inappropriate due to the fact that the user's queries may contain vagueness. The reason for the vagueness contained in the user's queries is that the user may not know much about the subject he/she is searching or may not be familiar with the information retrieval system. Therefore, the query specified by the user may not describe the information request properly[5].

Other issues such as performance, scalability and occurrences of paging update are other common information retrieval issues. Relevance is the relational value of a given user query to the documents within the database. Relevance of a document is normally based on a document ranking algorithm. These algorithms define how relevant a document is to a user query by using functions that define relations between the query given and the documents collected in the index. The evaluation of the feedback given by the information retrieval system is another issue with information retrieval. The behavior of the system may not meet the expectations of the user or the documents returned from the system may not all be relevant to a query[6].

Depending on the system and the user, the results of a query should be in a format that most fits the data being searched and returned. Information needs is how the user interacts with the information retrieval system. The data within the system should be able to be accessed easily and in a way that is convenient to the user. Retrieving too much information might be inconvenient in certain systems, also in other systems not returning all relevant information may be unacceptable.

As noted, managing volumes of information from the web can be difficult due to the volume of documents the server contains. This leads to an even larger problem when trying to retrieve the most relevant results to a query. A simple retrieval query can return thousands of documents, many of which are loosely related to the original retrieval criteria. Natural language and word ambiguity can also add distress to information retrieval. To overcome this, an information retrieval system needs to have good query management along with the ability to give weight to documents that are more relevant to the user's query, and present those results first.

III. Fuzzy Logic

Fuzzy Logic is basically a logic approach that allows intermediate truth values to be defined between conventional evaluations of true and false. Notions like rather hard or pretty cool can be formulated mathematically and processed by computers[7].

A. Fuzzy Sets

The very basic notion of fuzzy systems is a Fuzzy set. Fuzzy sets are sets whose elements have degrees of membership. In classical set theory, an element either belongs or does not belong to the set. However, fuzzy set theory permits the gradual assessment of the membership of elements in a set; this is described with the help of a membership function valued in the real interval [0, 1].

Nevertheless, there has not been a formal basis for how to determine the grade of membership. It is a subjective measure that depends on the context, set membership does not mean the same thing at the operational level in each and every context. Dubois and Prade have showed in that degree of membership can have one of the following semantics: a degree of similarity, a degree of preference, or a degree of uncertainty[7].

IV. Support Vector Machine (SVM)

We use a support vector machine (SVM) when your data has exactly two classes. An SVM classifies data by finding the best hyperplane that separates all data points

of one class from those of the other class. The *best* hyperplane for an SVM means the one with the largest *margin* between the two classes.

Margin means the maximal width of the slab parallel to the hyperplane that has no interior data points.

The *support vectors* are the data points that are closest to the separating hyperplane; these points are on the boundary of the slab. The following figure illustrates these definitions, with + indicating data points of type 1, and – indicating data points of type –1.

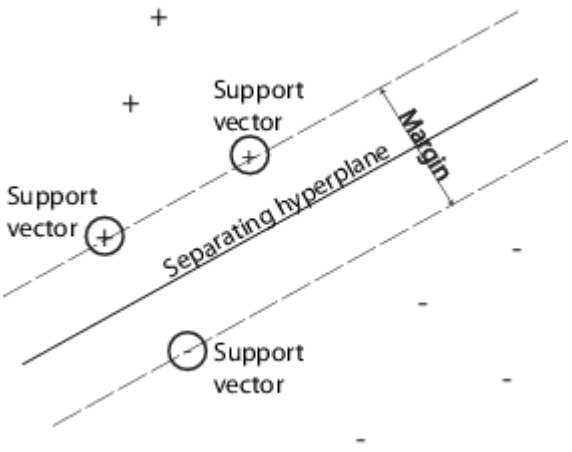

Fig. 2: Example of SVM hyperplane

- Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other, making it a non-probabilistic binary linear classifier.

- An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible.

- New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on.

- In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

- More formally, a support vector machine constructs a hyperplane or set of hyperplanes in a high-or infinite-dimensional space, which can be used for classification, regression, or other tasks.

- Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training data point of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier.

- Whereas the original problem may be stated in a finite dimensional space, it often happens that the sets to discriminate are not linearly separable in that space.

- For this reason, it was proposed that the original finite-dimensional space be mapped into a much higher-dimensional space, presumably making the separation easier in that space.

- The hyperplanes in the higher-dimensional space are defined as the set of points whose dot product with a vector in that space is constant. The vectors defining the hyperplanes can be chosen to be linear combinations with parameters of images of feature vectors that occur in the data base.

- With this choice of a hyperplane, the points in the feature space that are mapped into the hyperplane are defined by the relation: Note that if becomes small as grows further away from, each term in the sum measures the degree of closeness of the test point to the corresponding data base point.

- In this way, the sum of kernels above can be used to measure the relative nearness of each test point to the data points originating in one or the other of the sets to be discriminated.

- This function uses the maximal margin in this higher dimensional space. There is a penalty parameter which uses for the error term.

We use SVMStruct parameters to rank the documents on the basis of alpha, beta measures and bias that exceed a specified threshold value[8].

V. Existing Similarity techniques

The problem of fuzzy string searching can be formulated as follows: “Find in the text or dictionary of size n all the words that match the given word (or start with the given word), taking into account k possible differences (errors)."

For example, if we requested for ‘machine’ with two possible errors, find the words ‘marine’, ‘lachine’, ‘martine’, and so on.

Meanwhile, in most cases a metric is understood as a more general concept that does not meet the condition above, this concept can also be called distance. Among the most well-known metrics are Hamming, Levenshtein and Damerau-Levenshtein distances. The Hamming distance is a metric only on a set of words of equal length, and that greatly limits the scope of its application.

- **Levenshtein distance:** The Levenshtein distance, also known as "edit distance", is the most commonly used metric, the algorithms of its computation can be found at every turn. Nevertheless, it is necessary to make some comments about the most popular algorithm of calculation - Wagner-Fischer method.

$$\mathrm{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \mathrm{lev}_{a,b}(i-1,j)+1 \\ \mathrm{lev}_{a,b}(i,j-1)+1 \\ \mathrm{lev}_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Fig. 3: Levenshtein distance formula

The original version of this algorithm has time complexity of O(mn) and consumes O(mn) memory, where m and n are the lengths of the compared strings.

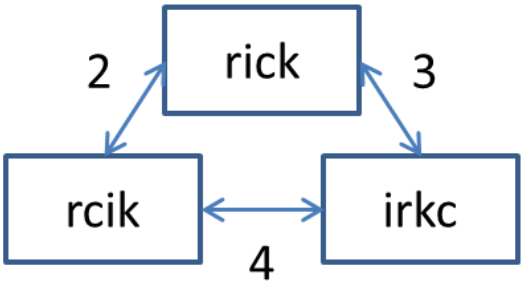

Fig. 4: Example of Levenshtein distance

- **Prefix Distance:** Usually it is necessary to calculate the distance between the prefix pattern and a string, to find the distance between the specified prefix and nearest string prefix. In this case, you must take the smallest of the distances from the prefix pattern to all the prefixes of the string. Obviously, the prefix length cannot be considered as a metric in the strict mathematical sense, what limits its application. Often, the specific value of a distance is not as important as fact that it exceeds a certain value.

- **Damerau-Levenshtein distance:** This variation contributes to the definition of the Levenshtein distance one more rule - transposition of two adjacent letters are also counted as one operation, along with insertions, deletions, and substitutions. Therefore, this metric gives the best results in practice[9].

$$d_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} d_{a,b}(i-1,j)+1 \\ d_{a,b}(i,j-1)+1 & \text{if } i,j > 1 \text{ and } a_i = b_{j-1} \\ d_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} & \text{and } a_{i-1} = b_j \\ d_{a,b}(i-2,j-2)+1 \end{cases} \\ \min \begin{cases} d_{a,b}(i-1,j)+1 \\ d_{a,b}(i,j-1)+1 & \text{otherwise.} \\ d_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} \end{cases}$$

Fig. 5: Damerau-Levenshtein distance implementatation process

To calculate this distance, it suffices to slightly modify the regular Levenshtein algorithm as follows: hold not two, but the last three rows, and add an appropriate additional condition - in the case of transposition take into account its cost[10].

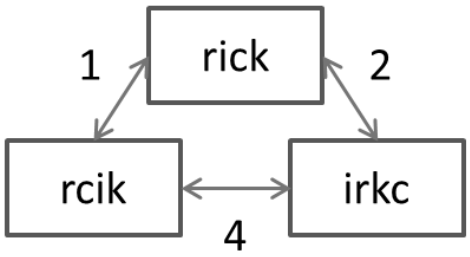

Fig. 6: Example of Damerau-Levenshtein Distance

V. Detailed Implementation

The metric that has been used by queries to determine a match is the Levenshtein distance formula. Simply put, the Levenshtein distance between two pieces of text is the number of insertions, deletions, substitutions, and transpositions needed to make one string match the other. For example, the Levenshtein distance between the words “ax” and “axe” is 1 due to the single deletion required. That means that when performing vague queries, the query text may be compared to an unanticipated term value as a result of analysis, leading to sometimes confusing results.

The source code is written in Java in which Lucene packages are imported. The first step is to create an index of all keywords appearing in the Ohsumed dataset followed by searching. A fuzzy retrieval method is implemented similar to the inbuilt FuzzyQuery method with varying parameters.

The parameters specified are:
1) Query terms: The string query specified by the user is converted to query terms which are then searched for in the generated index.
2) Fuzziness: The allowed difference between the keywords and query term that are to be retrieved.
3) Affix Length: The maximum possible character length common to both the query and keyword.

The fuzziness argument specifies that the results match with a maximum edit distance in terms of percentage. It should be noted that fuzziness should only be used with values less than 1. The official documentation still refers to setting fuzziness to float values, like 0.5, but these values are in-fact deprecated, and are harder to reason about[11][12].

The documents are retrieved on the basis of their score values depending upon the user query. The output documents are displayed in decreasing order on the basis of the score values. The relevance of retrieved documents are compared with the Ohsumed QRELs data to calculate the precision and recall using trec_eval script.

Further, the results are compared with the TF/IDF similarity retrieval results and SVM ranking results are comapred with BM25 (Okapi) results. The classification accuracy is calculated and the precision-recall values are measured on the basis of alpha, beta parameters in SVM.

The results obtained by entering the query 'intravascular coagulation' are as follows:

```
Enter your query:
intravascular coagulation
92 document(s) found. Time taken: 516ms
Score: 0.4333313.       File: F:\Lucene\Data\File_000219.txt
Score: 0.27083206.      File: F:\Lucene\Data\File_000222.txt
Score: 0.21704145.      File: F:\Lucene\Data\File_000224.txt
Score: 0.15347148.      File: F:\Lucene\Data\File_000307.txt
Score: 0.15146597.      File: F:\Lucene\Data\87058538.txt
Score: 0.15146597.      File: F:\Lucene\Data\File09.txt
Score: 0.13152444.      File: F:\Lucene\Data\87325763.txt
Score: 0.121470876.     File: F:\Lucene\Data\File_000107.txt
Score: 0.11691066.      File: F:\Lucene\Data\87242958.txt
Score: 0.1102244.       File: F:\Lucene\Data\File_000117.txt
Score: 0.092881575.     File: F:\Lucene\Data\File_000196.txt
Score: 0.08661552.      File: F:\Lucene\Data\File_000200.txt
Score: 0.08252516.      File: F:\Lucene\Data\File_000174.txt
Score: 0.07550464.      File: F:\Lucene\Data\File_000306.txt
Score: 0.073064275.     File: F:\Lucene\Data\File_000177.txt
Score: 0.07045632.      File: F:\Lucene\Data\87312037.txt
Score: 0.06460853.      File: F:\Lucene\Data\File_000243.txt
Score: 0.06101697.      File: F:\Lucene\Data\87198965.txt
Score: 0.05711615.      File: F:\Lucene\Data\File_000151.txt
Score: 0.053931884.     File: F:\Lucene\Data\87057614.txt
Score: 0.053931884.     File: F:\Lucene\Data\87309677.txt
Score: 0.05062041.      File: F:\Lucene\Data\File_0024.txt
Score: 0.043841504.     File: F:\Lucene\Data\File_000232.txt
Score: 0.04285372.      File: F:\Lucene\Data\File_0053.txt
Score: 0.04126258.      File: F:\Lucene\Data\File_000330.txt
Score: 0.03773987.      File: F:\Lucene\Data\87210296.txt
Score: 0.03736511.      File: F:\Lucene\Data\File_000314.txt
Score: 0.03711241.      File: F:\Lucene\Data\File_000291.txt
```

Fig 7: Output Screenshot: Score values of the retrieved documents

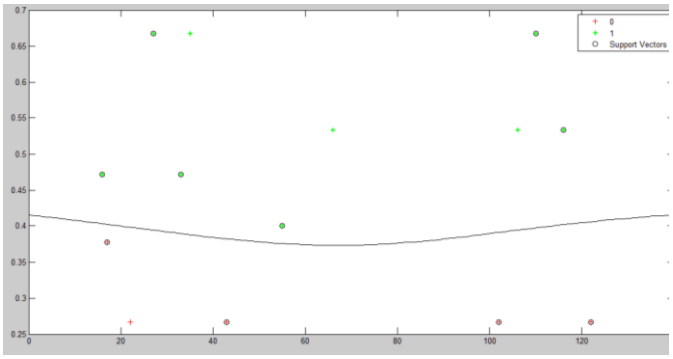

Fig 8: SVMplot: DocumentId v/s Score using RBF Kernel for TF/IDF Similarity

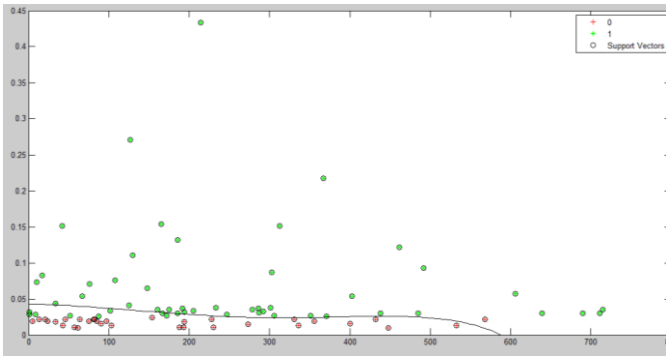

Fig 9: SVMplot: DocumentId v/s Score using RBF Kernel for Fuzzy Similarity

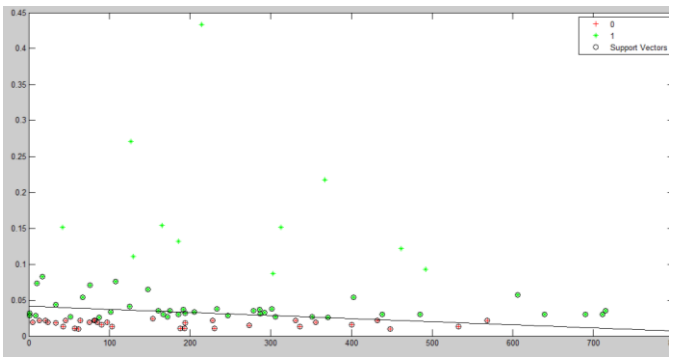

Fig 10: SVMplot: DocumentId v/s Score using Linear Kernel for Fuzzy Similarity

VII. PERFORMANCE EVALUATION

Even though Lucene's Levenshtein distance implementation is state of the art, and quite fast, it is still much slower than a plain match query. The runtime of the query grows with the number of unique terms in the index. That is to say, when performing a fuzzy search the main criteria is not how many documents will be returned, but how many unique terms across the cluster there are for the fields being searched. If there are 100 documents with 10,000 unique words apiece, searching that index will be slower than searching 10,000 documents where the field being searched only has 100 unique words[13][14].

The primary reason for this slowness is that a standard match query can quickly check the terms index, an internal data structure used by Lucene, for a match and find documents extremely quickly using binary search. This process is fast even for large dictionaries since binary searches scale well. Fuzzy queries, on the other hand, use a more advanced algorithm involving a DFA which must process a large number of terms. Processing the much larger number of terms required for a fuzzy search is always slower than a simple binary search.

The fuzziness setting, which defines the maximum number of characters in the query that may differ, can also have dramatic effects on the performance of a fuzzy query. Till now, various researchers have used the given parameters to analyze and evaluate the performance of IR Systems:

A. Precision

It is a fraction of documents that are relevant among the entire retrieved documents. Practically it gives accuracy of result.

$$\text{Precision} = |R_a| / |A|$$

- R_a: Set of relevant documents retrieved
- A: Set of documents retrieved

B. Recall

A fraction of the documents that is retrieved and relevant among all relevant documents is defined as recall. Basically, it gives coverage of result.

$$\text{Recall} = |R_a| / |R|$$

- R_a: Set of relevant documents retrieved
- R: Set of all relevant documents

C. Precision-Recall Curve

This curve is based upon the value of precision and recall where the x-axis is recall and y-axis is precision. Instead of using precision and recall on at each rank position, the curve is commonly plotted using 11 standard recall level 0%, 10%, 20%.....100% [15].

Moreover, average similarity value of documents for individual query and average number of retrieved relevant documents can also be used as parameters to check the performance of IR System. If the values for both of these parameters are high then the performance of IR System will be good. Let us now compare the proposed fuzzy logic based similarity algorithm with the conventional TF/IDF similarity measures to highlight the efficiency of the proposed system[16][17].
For the query 'intravascular coagulation', the output:

- With RBF kernel:

  1) Accuracy = 0.8261
  2) Precision = 0.7273
  3) Recall = 0.8889
- With Linear kernel:

  1) Accuracy = 0.8078
  2) Precision = 0.7192
  3) Recall = 0.8722

Further, we compare the precision-recall graph of the proposed fuzzy similarity measure with TF/IDF measures on three different TREC OHSU queries:

Query Number	Query	Maximum Score Value
1.	`menopausal woman`	`0.19804347`
2.	`intravascular coagulation`	`0.4333313`
3.	`cancer and hypercalcemia`	`0.31233424`

Table 1: Fuzzy queries on TREC dataset and their respective scores

The evaluation parameters of the proposed ranking measure are:

Alpha: Numeric vector of trained classifier coefficients from the dual problem (i.e., the estimated Lagrange multipliers). Alpha has length equal to the number of support vectors in the trained classifier (i.e., sum(SVMModel.IsSupportVector)).

Beta: Numeric vector of linear predictor coefficients. Beta has length equal to the number of predictors (i.e., size(SVMModel.X,2)).

If KernelParameters.Function is 'linear', then the software estimates the classification score for the observation *x* using

$$f(x)=(x/s)'\beta+b.$$

SVMModel stores β, b, and s in the properties Beta, Bias, and KernelParametersScale, respectively.

If KernelParameters.Function is not 'linear', then Beta is empty ([]).

Bias: Scalar corresponding to the trained classifier bias term.

The training data is said to be linearly separable, when data can be separated at two hyper planes of the margins in a way that there are no points between them and then try to maximize their distance. This is the simplest kernel and shows good performance for linearly separable data. In a non linear kernel, used for large set of attributes values and polynomial kernels , the kernel values may go to infinity[18][19].

- **Fuzzy Similarity v/s TF/IDF Similarity:**

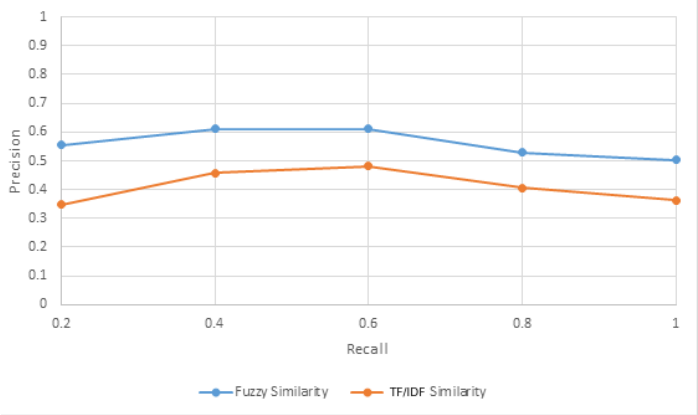

Fig. 11: Comparison of precision recall curve

In Fig. 11, fuzzy similarity based information retrieval system has high precision and recall values.

We calculate the rank of every retrived document such that the user can view the most relevant documents based on the query. The output is stored as follows:

	1	2	3
1	729	35.7245	1
2	596	28.1448	2
3	277	27.9399	3
4	192	26.8367	4
5	657	23.4622	5
6	983	23.4016	6
7	388	22.7990	7
8	32	21.9272	8
9	580	20.1665	9
10	438	16.1893	10
11	133	15.0718	11
12	368	13.5951	12
13	737	12.0790	13
14	48	11.6886	14
15	264	11.4530	15
16	647	11.4032	16
17	955	11.2120	17
18	595	11.0578	18
19	965	10.3578	19
20	831	10.1750	20
21	459	9.7987	21

Fig. 12: Ranking output on TREC data

VIII. CONCLUSION

In this paper, we discussed the implementation and efficiency details of an IR system with fuzzy based similarity measures and SVM based ranking. Experiments performed on TREC Ohsumed data collection using Apache Lucene prove the superiority of the proposed measure. This is a new technique having advantages over the other Information Retrieval systems as it can handle vague and imprecise queries of user very well. The performance of proposed technique is compared with TF/IDF based similarity measure on TREC dataset. Results indicate that proposed similarity measure technique based on vagueness, is better than TF/IDF based similarity measure technique for handling uncertain and imprecise queries. The insight provided by this model makes clear that fuzzy notions describe situations known through imprecise, uncertain, and vague information in a way that neither replaces nor is replaced but that, rather, complements the views produced by other approaches[20][21][22].

IX. REFERENCES

[1] Maria Angelica A. Leite, Ivan L. M. Ricarte, "Using Multiple Related Ontologies in a Fuzzy Information Retrieval
[2] Hajer Baazaoui, Marie-Aude Aufaure, Rania Soussi, "Towards an on- Line Semantic Information Retrieval System based on Fuzzy Ontologies", Journal of Digital Information Management, vol. 6, Oct. 2008.
[3] http://lucene.apache.org/core/
[4] Andrew Cholakian, How to Use Fuzzy Searches in Elasticsearch, https://www.found.no/foundation/fuzzy-search/
[5] TREC-9 Filtering Trackhttp://trec.nist.gov/data/t9_filtering.html
[6] Joseph Lee, Eunjin Kim, Fuzzy Web Information Retrieval System, http://www.micsymposium.org/mics2012/submissions/mics2012_submission_8.pdf
[7] G.N.K. Suresh Babu, S.K. Saravanan, Dr. S.K. Srivatsa, "Information Retrieval Challenges in Web", International Journal of Emerging Technologies in Computational and Applied Sciences (IJETCAS)
[8] Rima Harastani, Information Retrieval with Fuzzy Logic, ftp://ftp.irisa.fr/local/caps/DEPOTS/BIBLIO2010/Harastani_Rima.pdf
[9] Fuzzy String Search, http://ntz-develop.blogspot.in/
[10] Yogesh Gupta, Ashish Saini, A. K. Saxena, Aditi Sharan, Fuzzy Logic Based Similarity Measure for Information Retrieval System Performance Improvement, http://link.springer.com/chapter/10.1007%2F978-3-319-04483-5_23
[11] Saeedeh Maleki Dizaji, Jawed Siddiqi, Yasaman Soltan-Zadeh, Fazilatur Rahman, Adaptive information retrieval system via modelling user behavior http://link.springer.com/article/10.1007/s12652-012-0138-7#page-1
[12] R. Baeza-Yates and B. Ribeiro –Neto, "Modern Information Retrieval", Harlow: ACM Press 1999.
[13] H. Dong, F.K. Husain and E. Chang, "A Survey in Traditional Information Retrieval Models", IEEE International Conference on Digital Ecosystems and technologies, pp. 397-402,2008.
[14] S. Raman, V. Kumar, and S. Venkatesan, "Performance Comparison of Various Information Retrieval models Used in Search Engines", IEEE conference on communication, information and Computing Technology, Mumbai, India, 2012.
[15] F.Silva, R. Girardi, and L. Drumond, "An IR Model for the Web", IEEE International Conference on information technology, Brazil, 2009.
[16] A. Lashkari, F. Mahdavi and V. Ghomi, "A Boolean Model in Information Retrieval for Search Engines", IEEE International Conference on Information Management and Engineering, pp. 385- 389, 2009.
[17] J. Hua, "Study on the Performance of Information Retrieval Models", in 2009 International Symposium on

Intelligent Ubiquitous Computing and Education, pp. 436-439, 2009.

[18] D. Wolfram, "Search characteristics in different types of Web - based IR environments: Are they the same?", in Elsevier Journal of Information Processing and Management, pp. 1279- 1292, 2010.

[19] J. Han, M. Kamber, and J. Pai, "Data Mining: Concepts and Techniques", Morgan Kaufmann Publishers, USA, Third Edition, Elsevier, 2011.

[20] M. Karthikeyan and P. Aruna, "Probability based document clustering and image clustering using content-based image retrieval", in ELSEVIER journal of Applied Soft Computing, pp.959-966, 2012.

[21] Scholer, F., Kelly, D., Wu, W.-C., Lee, H., & Webber, W. (2013). The effects of threshold priming and need for cognition on relevance calibration and assessment. Proceedings of the 36th Annual ACM International Conference on Research and Development in Information Retrieval (SIGIR '13), Dublin, Ireland, 623-632.

[22] Brennan, K., Kelly, D., & Arguello, J. (2014). The effect of cognitive abilities on information search for tasks of varying levels of complexity. Proceedings of the Fifth Information Interaction in Context Conference (IIiX), Regensburg, Germany.