Information Retrieval System - Search Engine for Academic Papers

1nd Harsh Kavdikar *Computer Science Department*
Arizona State University
Arizona, USA
hkavdika@asu.edu

2st Hardik Sonetta *School Of Computer Science*
University of Windsor
Windsor, Canada
sonetta@uwindsor.ca

***Abstract*— Research Papers Search Engine is developed to provide an easy to use interface to query and retrieve relevant search papers to the keywords given by the users. The search engine is developed to retrieve papers contained in the Citeseer, VLDB and ICSE paper data-set. The basic functions of a search engine can be described as indexing and query processing. To enhance the functionality of the search engine there are various features included but are not limited to Stopword removal, keyword highlighting, Wildcard query matching, Search statistics, Web hosting, Stemming , Spellcheck and Query Auto-Suggestion. Additional enhancements are accomplished using Word2Vec model to recommend similar words given by user to be searched as a search query. Using Doc2Vec the search engine recommend similar documents to the users on the basis of the recent search query. Various classification algorithms are used to auto classify the documents and are evaluated using various performance metrics. Text Clustering is implemented to groups similar documents in the same cluster. The quality of the clusters are further validated. The dataset utilized to build this search engine is available at http://jlu.myweb.cs.uwindsor.ca/538/ [1]**

I. Introduction

The Information Retrieval System project developed consists of building a search engine that provides basic functionality accomplished in phase-1 like indexing of data, query processing, keyword highlighting, wildcard query matching, and displays search statistics to users. The Search Engine is developed using Apache Solr which is an open-source enterprise-search platform. Apache Solr provides a highly efficient indexing functionality to store and index dataset. [2] Further in phase-2, additional enhancement were done like stemming and spellcheck for the user query, to improve the overall user experience of the search engine. A Doc2Vec model was trained using Citeseer dataset, the input given to the model were 10,000 research papers, each tagged to the title of the paper for training. Additionally, Word2Vec model was trained using VLDB and ICSE dataset to recommend similar search queries. Additionally, a comparative study is done between five classification algorithms to evaluate and classify documents using Support Vector Machine (SVM), k-Nearest Neighbour (KNN), Gradient Boosting Machine (XGBoost), Naive Bayes and Random Forest classifier. The trained classification algorithms will aid in performing binary classification and will classify into categories of articles related to field of Software Engineering and Database Systems, namely. The performance of the algorithms will be evaluated using mentioned metrics like Precision, Recall and Accuracy. Lastly, Text Clustering is performed using Centroid clustering models which leverages K-means Clustering algorithm to generate clusters from unlabelled datasets. The quality of the clusters generated is evaluated using RAND index, Purity and silhouette co-efficient.

Fig. 1: Search Engine User Interface

II. Indexing and Searching

Search Engine indexing is facilitated using Apache Solr library. Solr indexing collects, parses, and stores data to facilitate fast and accurate information retrieval. Solr 8.41. have been utilized to build the index. Indexing procedure includes:

1. Obtain content from the directory which is to be indexed and analyzed the content using the standard analyzer of Solr. It preprocesses the query by lower casing each token and removes stop words and punctuation, if any.

repository.saveAll(researchPaper);
System.out.println("No. of papers done = " + count + " file = " + file.toFile().toString());
researchPaper = new ArrayList≤ResearchPaper≥();

2.) Created three fields namely: Title, Author, Content, and Filename to model the application.

3) Assigned these document to the index for indexing and searching.

User query via the front-end interface is being held in the content tag, which further passes the query to searchText variable. The query is preprocessed using trim function that removes stopwords and any punctuation in the user input

text.
String searchText = content.trim();
String searchQuery = "";
if (searchText.contains(" ")) searchQuery = "content:" + searchText + "";
QueryResponse response = solr.query(query);
This query is further passed to solr which retrieves relevant results that matches the keywords.

III. Improvements

The Search Engine is made user-friendly by enhancing basic functionalities and enabling Keyword highlighting, displaying search statistics, wildcard query matching, stemming and spellcheck module. The enhancements are described in further sections with intricate details:

A. Keyword Highlighting:

In order, to provide more context and confidence to user about the results retrieved, the keywords in the user query are highlighted in the retrieved results.

```
// Set Highlight and field to be highlighted
query.setHighlight(true);
query.addHighlightField("content");
```

Fig. 2: Wildcard query matching — Input Query: "tech" — Results include : 'technologies','techniques','technology'.

B. Search Statistics:

This features displays the number of matching results retrieved to the user. It also displays the time in milliseconds taken to retrieve the results.

```
// To find query execution time
long responseTime = response.getElapsedTime();
System.out.println("Query Resonse time = " + responseTime);
// To find total number of records
FieldStatsInfo stats = response.getFieldStatsInfo().get("id");
long numberofmatches = stats.getCount();//WildCard Search else searchQuery = "content:" + searchText + "*"; System.out.println("Total Number of records = " + numberofmatches);
```

C. Wildcard Query Matching:

Wildcard matching enables the flexibility to users to query the search engine and retrieve search results. This is specifically useful when users have less domain knowledge about the papers they are searching. Wildcard searches are based on character pattern matching. The character pattern matching takes place between the preprocessed user search query and indexed files.

```
//WildCard Search
searchQuery = "content:" + searchText + "*";
```

D. Web hosting:

Additional functionality was incorporated within the phase-1 of project by hosting the search engine application on Amazon EC2 instance. Amazon Elastic Compute Cloud is a cloud-computing platform. It is a Web Service that allows users to create virtual instances on which computer applications are hosted[3]. I created an Amazon EC2 instance, this instance is a Windows(64-bit) system with 1GB of memory and 30GB of storage drive. The remote system was configured and accessed locally via remote application. A war file was created for search engine application, this war file was migrated to the EC2 instance. Numerous configurations were done with security groups, user access control, and domain name system. Considering the memory capacity of the virtual instance it retrieves results comparatively better than the local server. This lessens the search time for users to fetch results and display on the user interface. As the number of users increases and the dataset get large in size, this instance can be easily scaled to handle increased traffic. This helps to avoid the overhead of manually configuring system from scratch and re writing the application code.

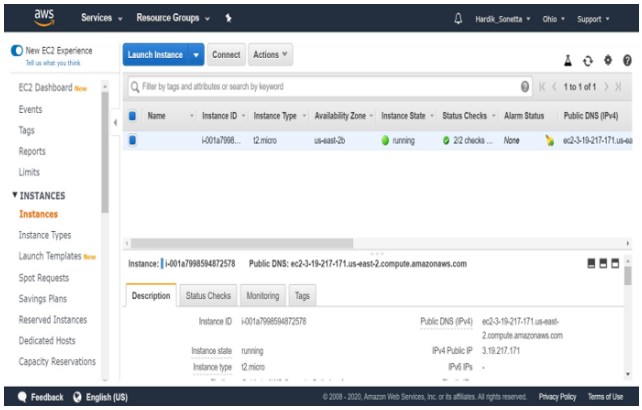

Fig. 3: Amazon EC2 instance

E. Stemming:

It is the process of reducing inflected words to their word stem, base or root form. It finds not only an exact match to the typed search query, but also other variations of the search query. Solr Porter stemmer is used to implement the

functionality of stemming. In porter stemmer a criterion is defined under which inflected letters in word are removed from a word to leave a valid stem.

Fig. 4: Stemming — Input Query: ”Connections” — Results include : ’Connection’,’Connected’,’Connects’,’Connections’

F. Spellcheck:

Elementary but most important feature that checks for misspellings in the user query. It auto-corrects spellings and search for intended words desired by the user. Also, gives enhanced flexibility to user in terms of information retrieval. An indexed based spellcheck functionality is used which is provided by the Solr library. Furthermore, a detailed findings were done to understand the working of the indexed based spellchecker. It performs spellchecking by creating a secondary index of the intended words. The index is built by creating character-based n-grams of the words from the original field.

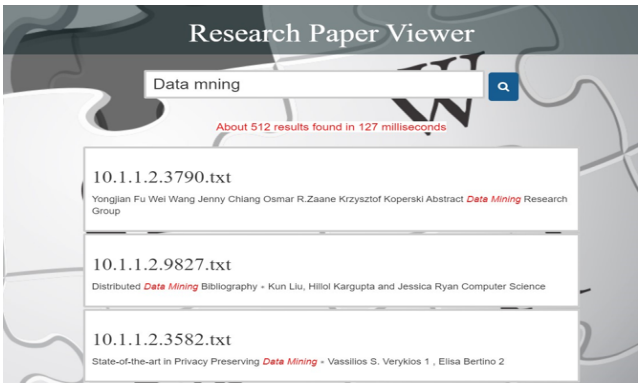

Fig. 5: Spellcheck — Input Query: ”Data Mning” — Results include : ’Data Mining’

At query time, the word to be checked is appropriately analyzed and then searched against this secondary index. Assuming one or more hits are returned, the candidate word is then compared to the original word using a String distance measure. In this case, Levenshtein distance is used which is nothing but the minimum number of single-character edits required to change one word into the other.

```
// To perform spellcheck
query.set(”spellcheck”, ”on”);
QueryResponse response = solr.query(query);

SpellCheckResponse = response.getSpellCheckResponse();
Suggestion suggestion = getSuggestions().get(0); List alternatives = suggestion.getAlternatives();
String alternative = alternatives.get(0);
```

G. Query Auto-Suggestion:

Query Suggestions aid users to navigate search engine more effectively and provide relevant content to the users. The Query Auto-Suggestion is implemented using Solr’s inbuilt Suggester component functionality. While, I have generated similar words and phrases using Word2Vec model which is described in further section in more detail. The Solr’s Auto Suggester functionality enables auto-suggestion by building a secondary index and uses EdgeNGramTokenFilter which generates N-gram for the typed query. I have configured the functionality in a way that it will generate suggestions only when at least 3 characters of query are given. This leads to more accurate suggestions as I have tried configuring a minimum character length of 2 as well. As we can see in Fig. 6 the input query is given as ’datab’ and the following suggestions are generated which are in context with the user query. This was a significant feature addition to the search engine as it greatly helps in enhancing user experience.

Fig. 6: Query-Autosuggestion

IV. Features

A. Word2Vec:

Word2Vec processes text by vectorizing words. It takes a large corpus of text as an input and outputs a set of vectors that represents the word in the corpus. [4] I have used 2 data sets namely ICSE, and VLDB. The steps followed to generate word vectors are as follows:

1.) I merged the two data sets using glob and created a corpus of text.

2.) Performed pre-processing of text by eliminating stop words, special characters and words of length smaller than 4. 3.) Used ”gensim.models.Word2Vec” with continuous skip gram model to convert the words into its vector representation.

4.) After using Word2Vec I generated the vocabulary from the model obtained as an output and saved Word2Vec vectors for further processing.
5.) Used TSNE dimensionality reduction to reduce the vector dimension(50 dimensions) to 2 dimensions and visualized the word vectors using scatter plot.
6.) To find the most similar words using the word vectors we used model.most similar() method.

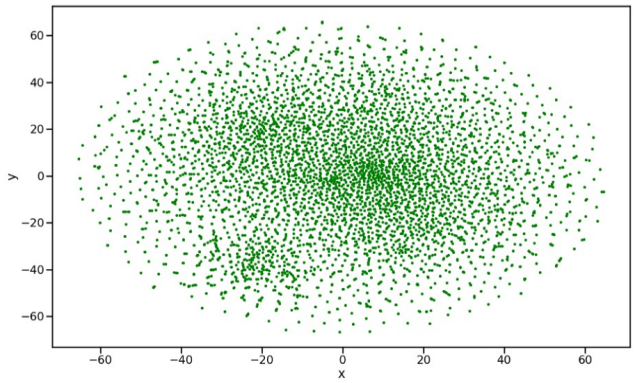

Fig. 7: TSNE - Dimensionality reduction of word vectors

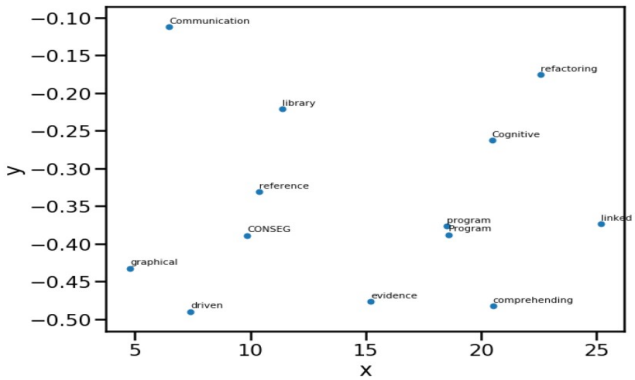

Fig. 8: Refined Word plottings in vector space after word2vec

As we can see in Fig 8. the word vectors are positioned in the vector space such that words that are relatively close in context are located in proximity to each other.

Observation: Experimented with CBOW and Skip gram model, the skip gram model gave better results as it tries to predict each context word from its target word. Experimented with the dimension of word vectors, as the text corpus is really huge the vector size was set to 50 and the model was run for 100 epochs.

B. Doc2Vec:

Doc2Vec creates a numeric representation of each document. The training task is similar to word2vec, except that it adds another vector that represents the whole document and assigns it to the words within the documents. I have used Citeseer dataset to work on the doc2vec model. The steps followed to generate document vectors are as follows:
1) Performed pre-processing on each document by eliminating stopwords, special characters and words with length less than 4.
2) Generated a .CSV file that consists of two columns namely: Title, Content of the document.
3) Prepared the dataset for training task by tagging and labelling content of each document from CSV file to the title of document.
4) Used ”gensim.models.Doc2Vec” with Skip grams to convert the documents into its vector representation.
5) To find the documents most similar to the user queried document, an inference from the model was made using output = model.docvecs.mostsimilar() function, which displays 10 most similar documents.

Observation: I have experimented with Distributed memory model in Doc2Vec but it was observed that DBOW is computationally fast and gives more precise similarities between documents.

```
[('DataSet\\citeseerExtracted\\2\\4736\\10.1.1.2.4736.txt',
  0.26451021432876587),
 ('DataSet\\citeseerExtracted\\2\\7369\\10.1.1.2.7369.txt',
  0.2374098300933838),
 ('DataSet\\citeseerExtracted\\2\\9178\\10.1.1.2.9178.txt',
  0.21500340104103088),
 ('DataSet\\citeseerExtracted\\2\\5373\\10.1.1.2.5373.txt',
  0.20437832176685333),
 ('DataSet\\citeseerExtracted\\2\\1835\\10.1.1.2.1835.txt',
  0.20406025648117065),
 ('DataSet\\citeseerExtracted\\2\\706\\10.1.1.2.706.txt', 0.20320871472358704),
 ('DataSet\\citeseerExtracted\\2\\1675\\10.1.1.2.1675.txt',
  0.20218074321746826),
 ('DataSet\\citeseerExtracted\\2\\3172\\10.1.1.2.3172.txt',
  0.19158287346363068),
 ('DataSet\\citeseerExtracted\\2\\5554\\10.1.1.2.5554.txt',
  0.18820132315158844),
 ('DataSet\\citeseerExtracted\\2\\4149\\10.1.1.2.4149.txt',
  0.18236184120178223)]
```

Fig. 9: Doc2Vec — User Query: ’Abstract security inter autonomous systems routing’ — Output: Paths or titles of the 10 most similar documents

C. Comparative study of Word2Vec and Doc2Vec model:

The fundamental concept behind word2vec is that it builds a semantic representation of words by tokenizing the corpus of raw text. Moreover, doc2vec also works with similar concept of tokenizing text but assigns an additional term of document id to the tokens which marks its association with that particular document. In word2vec, I trained the model to figure out word vectors and then run similarity queries between words. In doc2vec, it tags the text and you also get tag vectors also known as document vectors. Word2Vec captures the similarities between words and can suggest similar words to user by running similarity queries on generated word vectors and and returns the most contextually close words. It was observed that is computationally exhaustive to match the query word vector with all the word vectors of corpus. On the other hand, doc2vec captures the similarities between documents and suggests similar documents to the user. It does so by converting the document text query into a document vector and compares it with the all the documents present in the

dataset. Overall experiments and observations shows that the selection of a particular model completely depends upon the task. A word2vec model captures semantic relation between words and can generate word vectors out of it which can be further used for various NLP tasks. There will be times when we need more than just the contextual understanding of words, in fact, we might need to understand contextual relation between sentences, paragraphs or even documents. This is when doc2vec will perform best and lead to efficient implementation of the application.

D. Classification:

Text classification is an important application in NLP domain and very essential at the same time as it aids in classification of documents, calculating similarities between two sentences and as well as sentence completion. In this section, I will describe the binary classification of text documents I have implemented to classify documents into 2 categories. I have utilized ICSE and VLDB dataset for the task of text classification. TF-IDF vectors are generate for the text within the two dataset. The dataset is further split into train and test set with the ratio of 80% to 20%. Furthermore, five classification algorithms are implemented and a comparative study is done to find the best performing model on the dataset for our application. The performance of the models are evaluated on the basis of the contingency matrix which further aid in calculating Accuracy, Precision and F1-Score.

1) Methodology: The methodology includes the fundamental flow of the classification process carried out.

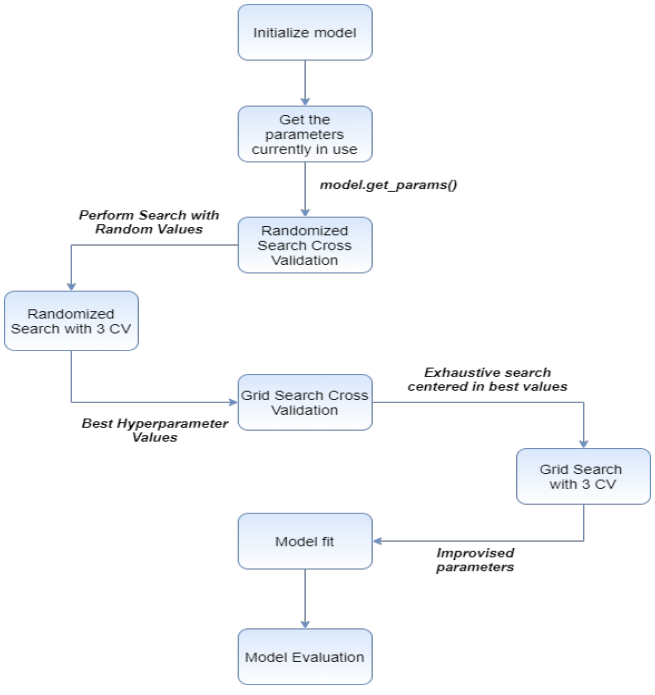

Fig. 10: Classification Process Work Flow

The word vectors are generated using TF-IDF vectorizer with 300 dimension size where a minimum count threshold is set to 3. The context size for the vectorizer is set to 7 which denotes that it considers 7 words in series to capture the context of the word and generate word vectors. Furthermore, the model is initialised and the words vectors are given as an input to the model. The active parameters of the model are accessed using model.get_params function. Once we know what parameters are active we try to fine tune it using random grid search method. After generating the parameters random grid the model is run with all the combinations of the parameter with 3 search cross validation as shown in Fig. 10. After the cross validation step is completed we get the best hyper parameters and the mean accuracy of the model is calculated. Once we have got best parameters with random grid search as the parameters were chosen randomly at the beginning we do a more centered search around these values to find the best hyper parameters. So, going further we again generate a parameter Grid to tune the hyper parameters. The model is again run with all the combinations of parameters in the grid for 3 search cross validation on the training set. Lastly, we get the best parameters for the model and calculate the mean accuracy obtained after performing cross validation for the second time. If we see an increase in the model accuracy we perform a final fit on the model using these hyper-parameters. The model is evaluated further to see if it is performing any misclassification of observations and it is visualised using a confusion matrix. The detailed description of each model and its performance follow in the upcoming sections.

2) Support Vector Machine: SVM are widely known for the kernel trick, which transforms the data into higher dimensions and makes it linearly separable. This helps in maximizing the linear boundary between the two classes and gives good classification score. The TF-IDF vectors are given as an input to the classifier which are generated after pre-processing the text by removing all the stopwords and punctuation signs.

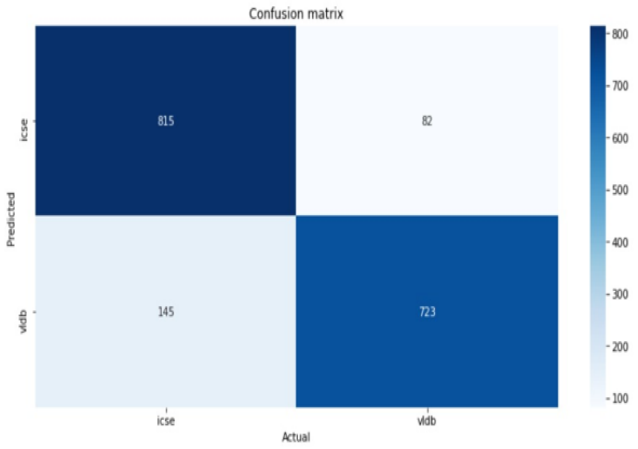

Fig. 11: Confusion Matrix for SVM Classifier

After the training process is completed, predictions are generated on the test set that was split before performing the model training. After the predictions are generated, confusion matrix is built using predicted labels and the test labels.

The confusion matrix in Fig.11 shows that the number of False Positive and False Negative are very less in number comparatively which overall leads to less misclassification of the documents.

3) Random Forest: Random forest classifier is an ensemble model which consists of multiple decision trees that generate predictions and final prediction is generated for a class with majority of votes given by the decision trees. These type of ensemble classifier was experimented to see how it performs on the feature vectors generated for textual data using TF-IDF vectorizer which was given as an input to the classifier. After the training process was completed, predictions were generated on the test set that was split before performing the training of the model. The predictions are generated and the confusion matrix is built using predicted labels and the test labels.

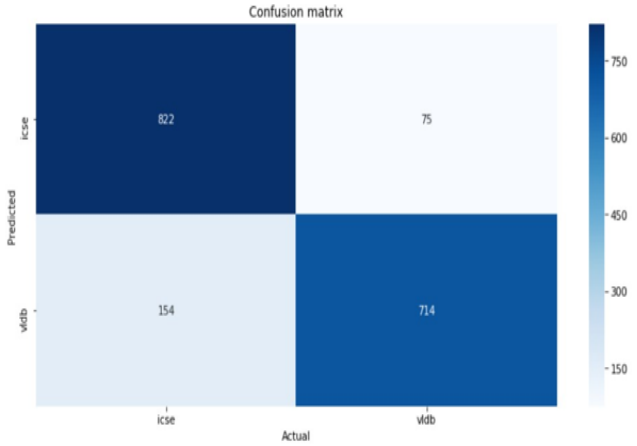

Fig. 12: Confusion Matrix for Random Forest Classifier

The confusion matrix in Fig.12 shows the number of False Negative are comparatively more for VLDB class.

4) Gradient Boosting Machine: Also known as XGBoost, is also another ensemble model which consists of multiple decision trees but the significant difference between XGBoost and Random forest is that they work in different ways with textual data. Random forests assigns higher weights to misclassified class and optimizes it. On the other hand, XGBoost optimizes the gradient of the loss function which give flexibility in achieving greater performance than other classifiers. It was observed that the XGBoost took longer time to train as it take up more computation times to work with 300 dimensions TF-IDF vectors. The predictions were generated on test set and were used to build confusion matrix.

Fig. 13 shows the confusion matrix for GBM classifier and is quite similar to the performance of random forest

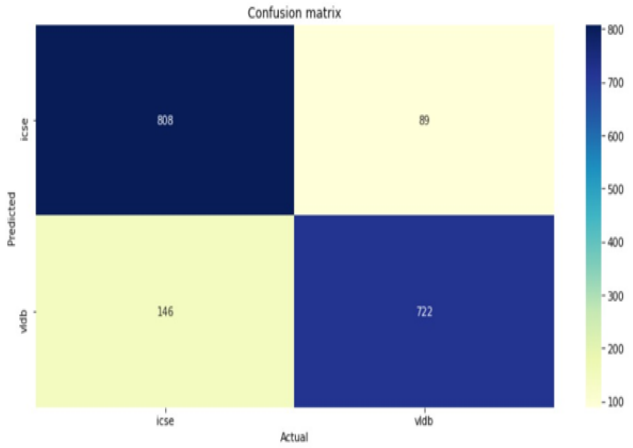

Fig. 13: Confusion Matrix for GBM Classifier

classifier as both are ensemble models and have relatively similar performance.

5) Naive Bayes: The Naive Bayes classifier is a probabilistic classifier and computes the probability of a document d being in a class c. Naive Bayes classifier is the best suitable classifier for the task of binary classification given it's simplicity and the time it takes to generate inference or predictions from the test set. The training time for Naive Bayes classifier is comparatively less and generates the predictions fast as compared to other models.

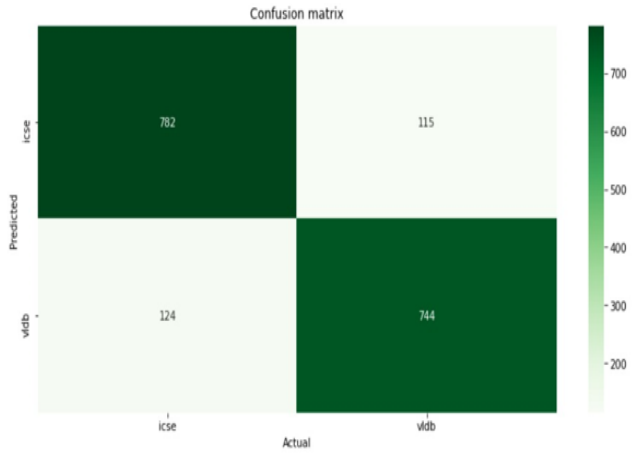

Fig. 14: Confusion Matrix for Naive Bayes Classifier

Fig: 14 shows the confusion matrix for Naive Bayes classifier and we can see that it predicts VLDB class better than any other classifier and high rate for True Positives and True Negatives.

6) K-Nearest Neighbour: KNN classifier assumes that similar things exist in close proximity. KNN works on the idea of similarity also known as closeness by calculating the distance between points in the context space. In KNN, for the text classification task I have loaded the TF-IDF vectors and initialized k to 7 which is the number of neighbours it

considers. For each sample in the dataset it calculates the distance between the query example and the current example from the data. After the training process is completed and a test sample is given it returns the prediction which has the smallest distance to the training samples.

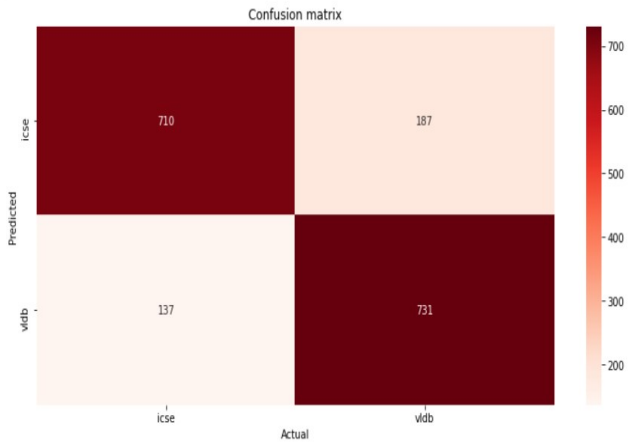

Fig. 15: Confusion Matrix for KNN Classifier

7) Classification Report:

From the table below, we can observe that SVM classifier gives the best performance on our dataset for the task of text classification in terms of Accuracy, Precision and F1-Score. The exceptional performance of the SVM classifier can be attributed to the nature of the dataset, as the dataset is linearly separable it can generate a classifier margin with maximum boundary which leads to accurate classification.

EVALUATION METRICS FOR 5 ALGORITHMS					
Models/ Metrics	SVM	Random Forest	GBM	Naïve Bayes	KNN
Accuracy	**93%**	87%	87%	86%	**82%**
Precision	**92%**	90%	89%	87%	**84%**
F1-Score	**93%**	88%	86%	86%	**81%**

Fig. 16: Classification Report for 5 classifiers

E. Clustering:

Clustering is the task of grouping a set of similar objects together in a cluster. Text clustering has wide applications with which we can cluster similar documents together, recommend similar content to end users as per the given query. To perform the task of text clustering I have utilized K-Means Clustering algorithm on ICSE and VLDB dataset. The labels have been taken off as this is an unsupervised task. TF-IDF vectors are generated for all the observations in the dataset to perform clustering. Following steps are performed to implement KMeans algorithm:

- Specify the desired number of clusters K
- Randomly assign each data point to a cluster
- Compute cluster centroids
- Re-assign each point to the closest cluster centroid
- Re-compute cluster centroids

The algorithm is run for 300 maximum iterations. After the KMeans algorithm is run, dimensionality reduction technique is performed on the resulting cluster samples. The dimensionality reduction technique that I have used is called as Principal Component Analysis. It reduces the observation size to 2 dimensions to visualize clusters on a 2d-plot.

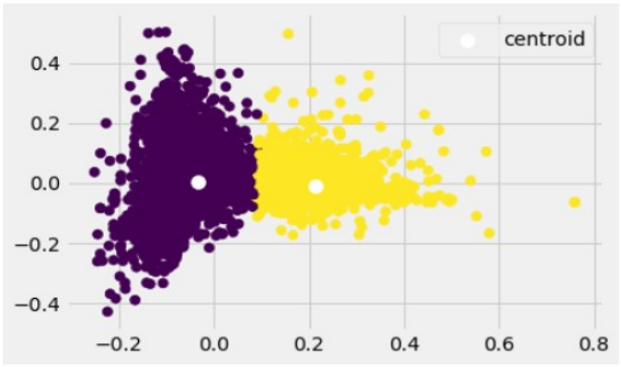

Fig. 17: Clustering for K=2

Observation: As we can see in Fig. 17 the resulting plot shows 2 distinct clusters for the two types of documents i.e ICSE and VLDB dataset. It shows the two centroid points in white color which are well separated. The clusters shows the property of ideal clustering as the data points within each cluster are close to each other and the two clusters are well separated.

After the two clusters are generated I have used feature selection technique to retrieve the most prominent features. I have utilized chi-square feature selection technique and visualized the top 15 features in the two clusters.

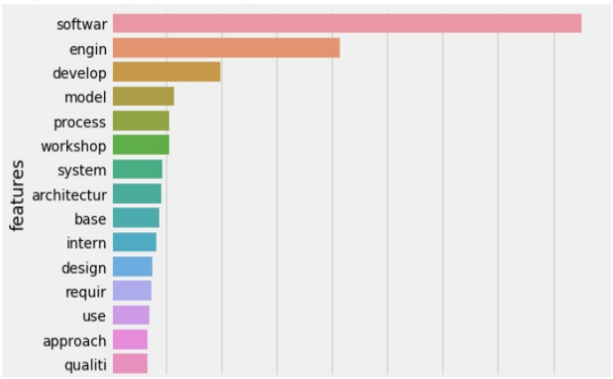

Fig. 18: Cluster-1

Fig-18 shows top 15 features of Cluster-1 and as we can see words like 'Software','Engineering','Development',etc. Which gives us an intuition that this cluster is related to

ICSE documents.

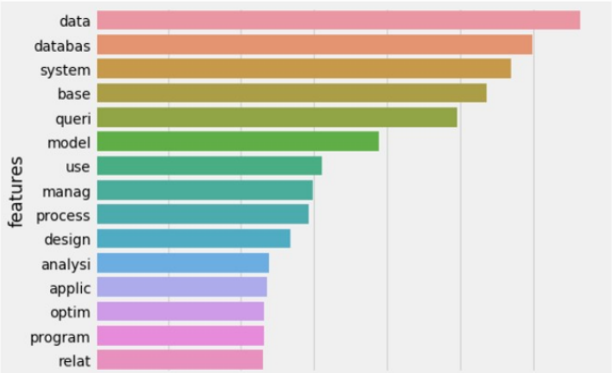

Fig. 19: Cluster-2

Fig-19 shows top 15 features of Cluster-2 and as we can see words like 'data','database','System',etc. Which gives us an intuition that this cluster is related to VLDB documents.

1) Finding Optimal Number of Clusters: In this case, dataset evidently showed 2 categories at the beginning of the clustering task i.e 'ICSE' and 'VLDB'. So, we chose number of clusters k = 2 for KMeans Algorithm. What if the number of categories (Clusters) are unknown? To determine the optimal number of cluster we use Elbow method. The method consists of plotting the explained variation as a function in this case Sum of Squared error distance of the number of clusters, and picking the elbow of the curve as the number of clusters to use.

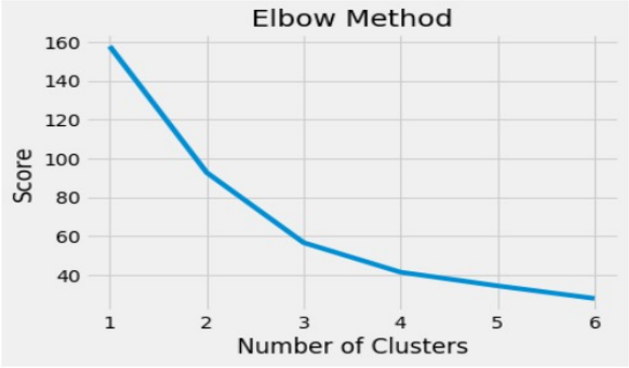

Fig. 20: Elbow Method

Observation: As we can see and Elbow curve or steep drop from cluster-1 to cluster-2, this shows that k=2 is the optimal choice for our dataset and is accurate.

2) Cluster Validation: After the clustering algorithm is performed on our dataset and the cluster have been generated it is essential to validate the quality of the clusters. The cluster validation is done using the following cluster evaluation techniques i.e RAND index, Purity and Silhouette analysis.

a) RAND Index: Rand Index can be seen as calculating the accuracy, but is applicable when class labels are not used. It is a measure of the similarity between two data clusterings. [5] Rand Index can value between 0 and 1, with 0 indicating that the two data clusterings do not agree on any pair of points and 1 indicating that the data clusterings are exactly the same. The Rand Index is calculated for the resulting clusters from KMeans algorithm on our dataset. We achieve a RAND index of 0.61. Which denotes that the clustering quality is good.

b) Purity: Purity is a measure of the extent to which clusters contain a single class. It is the percent of the total number of objects(data points) that were classified correctly. Which implies that each cluster Ci has identified a group of objects as the same class that the ground truth has indicated. The purity score is calculate for the resulting cluster and a score of 0.62 was achieved for clusters k=2. [6]

c) Silhouette analysis: Once the clusters are generated; quality of the clusters can be determined using Silhouette analysis. [7] It is used to determine the degree of separation between clusters by:

- Computing the average distance from all data points in the same cluster
- Computing the average distance from all data points in the closest cluster
- The coefficient can take values in the interval [-1, 1]
  0 : the sample is very close to the neighboring clusters.
  1 : the sample is far away from the neighboring clusters.
  -1 : the sample is assigned to the wrong clusters.

Higher the score and close to 1, better is the cluster quality, Silhouette coefficient is calculated for 2 and 3 numbers of clusters.

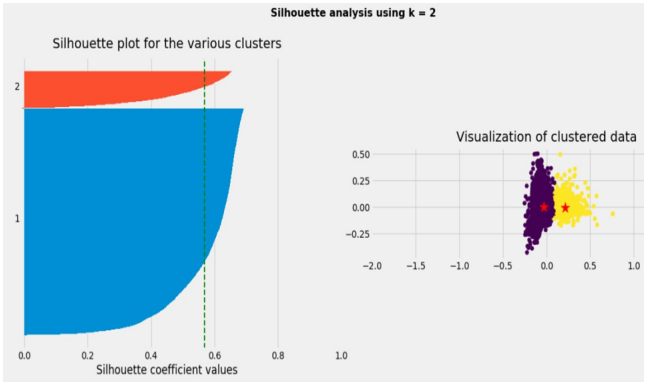

Fig. 21: Silhouette Score for K=2

Observation: Fig.21 shows the Silhouette score for K=2 clusters and achieves a score of 0.60 which denotes that the two cluster are well separated from each other and are the individual cluster are clustered tightly. Also, the blue region shows a cluster which has double the samples of the other cluster.

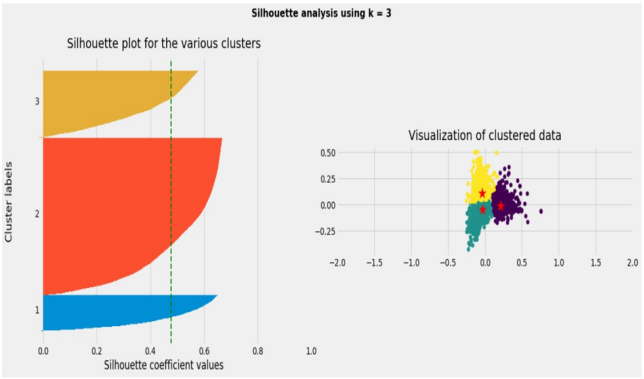

Fig. 22: Silhouette Score for K=3

Observation: Fig.22 shows the Silhouette score for K=3 clusters and achieves a score of 0.45 which denotes that it has significantly dropped than when we were using 2 clusters for our dataset.

V. Challenges Faced

This was an exciting project as I was working with new frameworks and programming languages to implement the features and enhancements. Also, this was the first experience working as a single member to develop an Academic project from scratch, as it involved brainstorming by myself and thinking hard sometime to solve an error or a bug. Some of the technical challenges that I came across are as follows:

- I was in a dilemma of choosing the framework for developing the Searching and Indexing part of the Search Engine. I had a choice to pick from Lucene or PyLucene(Python wrapper around Java Lucene). I have previously worked with python programming language and was comfortable using it. But I thought of taking this as an opportunity to work with new programming language, so I selected Lucene as the framework to work with. I developed some of the features and enhancements using python as well and integrating and making the two language communicate was a learning experience.
- The most challenging part of the project was the implementation of the word2vec and doc2vec models. Also, this part took the most of the time to accomplish it. As it involved generating word vectors using TF-IDF which required a lot of fine-tuning of parameters. It was important to get the correct numerical vectors of the text as it has direct influence on the performance of word2vec and doc2vec models.
- Lastly, hosting the Search Engine on an Amazon EC2 instance was something that I tried for the very first time. It gave a real experience of deploying a fully functional system in a production environment. Major challenge was to setup the environment and make the java environment run on the remote instance.

VI. Conclusion

To summarize, the Information Retrieval Search Engine is implemented using Lucene library which enabled to integrate features like Indexing and Searching, Search Statistics, Stop Word Removal, Highlighting, Wildcard Matching, Stemming , Spellcheck, Query Auto-Suggestion and Web hosting of the search engine on Amazon EC2 machine. Furthermore, advanced enhancements were made by implementing Word2Vec and Doc2Vec models to generate similar words for phrase expansion or query suggestion and to recommend similar documents to the user as per the given query. Additionally, a comparative study was done among five classification algorithms and text clustering was performed using KMeans algorithm on unlabelled dataset. Thus, by working on this project for the academic course of Information Retrieval System, I could grasp the fundamental concepts learned in class and implemented them practically using the programming languages and frameworks that I had never worked upon. This only helped me to learn more and enhance my ken on the subject of Information retrieval Systems.

References

[1] D. Lu, "Irs," *online:* http://jlu.myweb.cs.uwindsor.ca/538/.
[2] *online:* https://lucene.apache.org/solr/guide/6_6/introduction-to-solr-indexing.html.
[3] *online:* https://aws.amazon.com/ec2/instance-types/.
[4] *Journal:* [IIR]IntroductiontoInformationRetrieval, byC.Manning,P.Raghavan,andH.Schutze. CambridgeUniversityPress,2008.
[5] *Online:* https://en.wikipedia.org/wiki/Rand_index.
[6] *Online:* https://stats.stackexchange.com/questions/95731/how-to-calculate-purity.
[7] *Online:* https://medium.com/@jyotiyadav99111/selecting-optimal-number-of-clusters-in-kmeans-algorithm-si