**DWIT COLLEGE**

DEERWALK INSTITUTE OF TECHNOLOGY

# INFORMATION RETRIEVAL FROM CORPUS OF DOCUMENTS USING LATENT SEMANTIC INDEXING

A MINI PROJECT REPORT

Submitted to
Department of Computer Science and Information Technology
DWIT College

Submitted By

Sushil Awale
Class of 2019
December 13, 2018

# CERTIFICATION

This project titled ‘INFORMATION RETRIEVAL FROM CORPUS OF DOCUMENTS USING LATENT SEMANTIC INDEXING’ by Sushil Awale, under the supervision of Mr. Sarbin Sayami, Deerwalk Institute of Technology, is hereby submitted for the partial fulfillment of the requirements for the Sixth Semester Mini Project.

Approved By:

……………………………… ……………………..

Mr. Sarbin Sayami Date

Project Supervisor

# ACKNOWLEDGMENT

I would like to express my special thanks of gratitude to my supervisor, Mr. Sarbin Sayami, as well as DWIT who gave me the golden opportunity to do this wonderful project on the topic (Information Retrieval From Corpus of Documents Using Latent Semantic Indexing), which also helped me in doing research and I came to know about so many things, I am really thankful to them. Secondly, I would also like to thank my parents and friends who helped me a lot finalizing this project within limited time frame.

Regards

Sushil Awale

Class Of 2019

# ABSTRACT

Vocabulary mismatch problem is a common phenomenon in the usage of natural languages for information retrieval. Systems that rely on matching user's query and documents solely on the basis of vocabulary suffer from problems of synonymy and polysemy. Research shows that on average 80% of the time different people (experts in the same field) will name the same thing differently. In such scenario, information retrieval must move beyond keyword matching and find relevant documents based on the concept.

Latent Semantic Indexing is one such method that identifies patterns in the relationships between the terms and the concepts contained in an unstructured collection of text. This project implemented Latent Semantic Indexing with TFIDF to retrieve relevant documents for a search query on a corpus of machine learning research papers from ArXiv.

*Keywords: Latent Semantic Indexing, Information Retrieval, TFIDF*

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF ABBREVIATIONS

| | | |
|---|---|---|
| API | - | Application Programming Interface |
| CPM | - | Critical Path Method |
| LISA | - | Library and Information Science Abstracts |
| LSA | - | Latent Semantic Analysis |
| LSI | - | Latent Semantic Indexing |
| PDF | - | Portable Document Format |
| TF-IDF | - | Term Frequency - Inverse Document Frequency |
| WBS | - | Work Breakdown Structure |

# CHAPTER 1: INTRODUCTION

## 1.1 Background

### 1.1.1 Information Retrieval

Information Retrieval has broad meaning but as an academic discipline, it can be defined as follows. Information Retrieval (IR) is concerned with locating information (usually documents) of an unstructured nature (usually text) from within large collections (usually stored on computers) that is relevant to a user's information need. [1]

An information need is a topic about which the user desires to know more, and is differentiated from a query, which is what the user conveys to the computer in an attempt to communicate the information need. [1] An information retrieval system aims to provide documents from within the collection that is relevant to an arbitrary user information need, communicated to the system by means of a one-off, user-initiated query. [1]

Information Retrieval systems facilitate retrieval of unstructured data compared to data retrieval systems which facilitate retrieval of structured data. The term "unstructured data" refers to data which does not have clear, semantically overt, easy-for-a-computer structure. [1] It is the opposite of structured data, the canonical example of which is a relational database, of the sort companies usually use to maintain product inventories and personnel records. [1]

Information Retrieval used to be limited to only a few people, especially, record keepers, librarians, and professional searchers. But nowadays, millions of people engage in information retrieval every day when they use a web search engine or search their email. [1]

## 1.1.2 Information Retrieval Process

The information retrieval process can be divided into several sub-processes, as shown in figure 2.1.

Before initiating the retrieval process, the text database must be defined. This involves the following processes:

- The documents to be used,
- The operations to be performed on the text, and
- The text model

The text operations such as token identification, stemming, stop words removal, etc. are applied to transform original documents into logical view specified by the information retrieval system.

Next, an index of the text is built from the logical view of the documents. An index helps in optimizing speed and performance while retrieving relevant documents for a search query. A popular index structure is an inverted index. It consists of a dictionary of terms and each term is associated with a list of documents.

Now that the document database is indexed, the retrieval process can be initiated. The user's query is transformed by the same text operations as that of the documents and a list of retrieved documents is produced.

The retrieved documents are then ranked according to a relevance function.

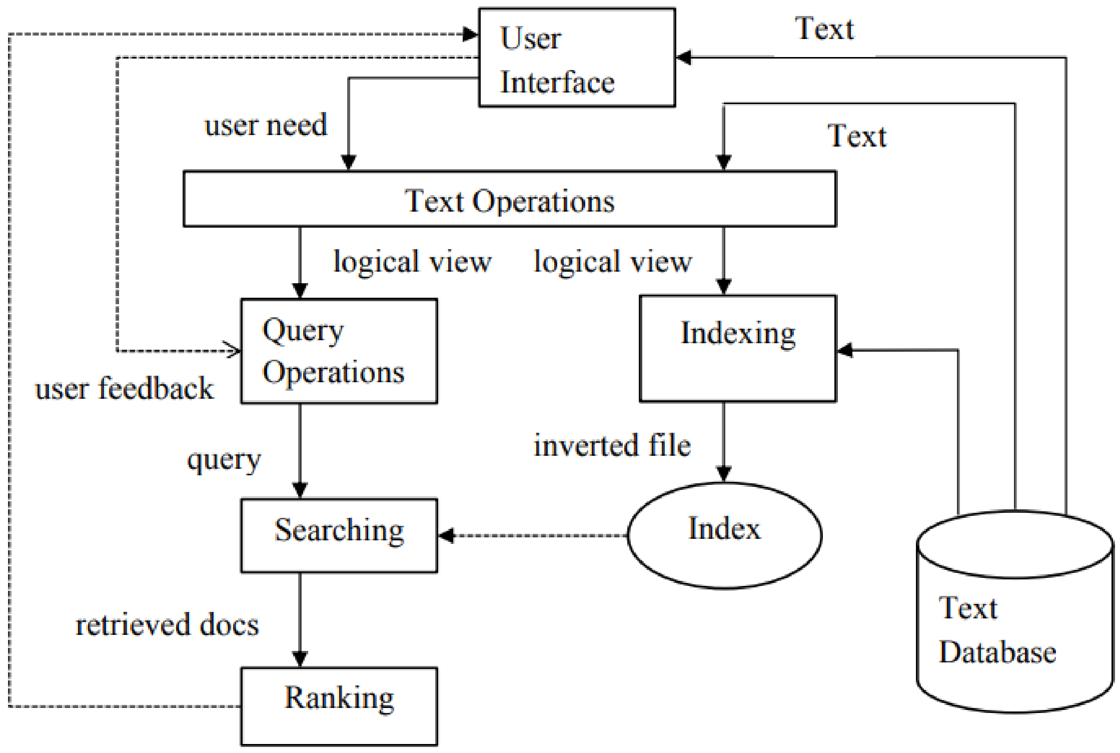


Figure 1.1 - Information Retrieval Process

## 1.2 Problem Statement

Vocabulary mismatch problem is a common phenomenon in the usage of natural languages. [2] Information retrieval systems that rely on matching user's query and documents solely on the basis of vocabulary suffer from problems of synonymy and polysemy. Synonymy refers to the use of synonyms or different words that have the same meaning, and polysemy refers to words that have different meanings when used in varying contexts. [3] Research shows that on average 80% of the time different people (experts in the same field) will name the same thing differently. [4]

## 1.3 Objectives

The objectives of this project are:

- To retrieve relevant documents from a corpus by matching the concept of a search query with that of each document
- To rank the retrieved documents based on relevance to the search query

## 1.4 Scope

The project uses Latent Semantic Indexing to retrieve documents relevant to the user-entered search query and ranks the documents in order of relevance using Cosine Distance. The corpus of documents consists of locally stored research papers on Machine Learning published on ArXiv downloaded using ArXiv's API. The model can be used to build concept-based information retrieval systems for any corpus of PDF documents.

## 1.5 Limitation

1. The retrieval process is limited to the corpus of documents used.

## 1.6 Outline of Document

The report is organized as follows:

**Preliminary Section:** This section consists of the title page, abstract, table of contents and list of figures and tables.

**Introduction Section:** In this section, the background of the project, problem statement, its objectives, scope, and limitation are discussed.

**Requirement and Feasibility Analysis Section:** Literature review, Requirement analysis, and Feasibility analysis make the bulk of this section.

**System Design Section:** This section consists of the methodology that was implemented in the project and the system design as well.

**Implementation and Testing Section:** In this section, the implementation, description of major methods and testing of the system are discussed.

**Conclusion and Recommendation:** This section consists of the final findings and the recommendations that can be worked on in order to improve the project.

# CHAPTER 2: REQUIREMENT AND FEASIBILITY ANALYSIS

## 2.1 Literature Review

### 2.1.1 Information Retrieval Strategies

The two major approaches to information retrieval are semantic and statistical approaches. Semantic approaches attempt to model some level of syntactic and semantic analysis; that is, to understand natural language text. [6] In statistical approaches, the degree of match, or similarity, between the query and the documents to retrieve, i.e., that are highly ranked, is determined by some statistical measure. [6]

Retrieval strategies assign a measure of similarity between a query and a document. The underlying algorithm takes a query and a set of documents $d_1$, $d_2$,..., $d_n$ and identifies the Similarity Coefficient SC(q, $d_i$) for each of the documents $1 \leq i \leq n$. [11] These strategies are based on the general assumption that the more often terms co-occur in both the document and the query, the more relevant the document is with respect to the query. [11]

The three classical retrieval models in information retrieval are Boolean, vector and probabilistic.

#### 2.1.1.1 Boolean Model

The Boolean retrieval model is one of the oldest information retrieval models. In this model, query and documents are exactly matched i.e. documents are retrieved if they exactly match the query description, otherwise, they are not.

Boolean retrieval is not generally described as a ranking algorithm; the model assumes that all documents retrieved are equivalent in terms of relevance [2], in addition to its retrieval strategy that is based on a binary decision criterion, i.e. relevance is binary. [12]

The Boolean model is considered to be the weakest classic method. Its inability to recognize partial matches frequently leads to poor performance. [12] Another disadvantage with the model includes the difficulty for the user to formulate an information need into a Boolean expression. [6]

#### 2.1.1.2 Vector Space Model

The vector space model [13] betters the Boolean model by facilitating a partial match. Instead of using binary values, the vector model assigns non-binary weights to the index terms in documents and queries which describe the importance of the terms in the documents.

The weighted terms are used to compare the degree of similarity between the documents and the query. The main resultant effect is that the ranked document answer set is a lot more precise (in the sense that it better matches the user information need) than the document answer set retrieved by the Boolean model. [12]

Main advantages of the vector model are that its term-weighting scheme improves retrieval performance, that its partial matching strategy allows retrieval of documents that approximate the query conditions and that its ranking formula(s) sort(s) the documents according to their degree of similarity to the query. [6]

The vector model provides a convenient computational framework, but provides little guidance on the details of how weighting and ranking algorithms are related to relevance. [2]

**Weighting**

Term weights define the importance of a term in a document and in the overall collection. Term discrimination consideration suggests that the best terms for document content identification are those able to distinguish certain individual documents from the remainder of the collection. [15]

This implies that the best terms should have high term frequencies in the document but low overall collection frequencies. [6]

An issue with the term weighting formula is that it will favor long documents over short documents since the former tend to repeat terms more often. To avoid favoring longer documents a normalization factor needs to be included in the weighting formula. [3]

The weighting scheme contains a local factor that indicates how important the term is in the document, a global factor that indicates how important the term is in the collection and a normalization factor to avoid favoring long documents over shorter ones. [15] [9]

**Local Term-Weighting**

**Binary**

Definition:

$$\mathrm{l}_{i,j} = 1, \text{if the term i exists in document j, otherwise 0}$$

The above formula ignores the actual frequency the term has in the document. Instead, it assigns every word that appears in a document equal relevance based on if the word is present or not. The use of binary weights is often too limiting since it does not provide consideration for partial matches.

**Term Frequency**

Definition:

$$l_{i,j} = f_{i,j}, f_{i,j} \text{ indicates the number of occurrences of term i in document j}$$

The above formula takes into account the frequency that the term has in the document. A term that occurs more often within a document is considered more important than a term that occurs less often.

**Logarithmic Term Frequency**

Definition:

$l_{i,j} = \log(f_{i,j} + 1), f_{i,j}$ indicates the number of occurrences of term i in document j

The above formula, like term frequency, counts how many times the term occurs in the document, but it de-emphasizes the effect by using the log function. A term that appears ten times in a document is not necessarily ten times more important than a term that appears only once.

**Global Term-Weighting**

**No Changes**

$$g_i = \mathbf{1}$$

The above formula removes the effect of the global weight factor.

**Inverse Document Frequency**

Definition:

$$g_i = log_2 \frac{n}{1 + df_i} ,$$

where n is the number of documents, and dfi is the number ofdocuments that contain term i

The above formula is used to increase the importance of terms that are rare in the document collection and to decrease the importance of terms that are very common in the document collection. [15]

### 2.1.1.3 Conceptual Indexing

In the vector space model, representation solely based on term frequencies does not accurately model the semantic content or relations of information (in a collection). [6] Improvements on the vector space model can be made by following one of two approaches. One approach is based on term weights and another on the computation of low-rank approximations to the original term-by-document matrix. [6] Vector space models such as Latent Semantic Indexing rely on such approximations to encode both terms and documents for conceptual-based query matching. [7]

LSI attempts to exploit and model global usage patterns of terms so that related documents that may not share common (literal) terms are still represented by nearby vectors in a k-dimensional subspace. [3]

### 2.1.2 Vocabulary Mismatch Problem

Vocabulary mismatch problem is a common phenomenon in the usage of natural languages. [2] Information retrieval systems that rely on matching user's query and documents solely on the basis of vocabulary suffer from problems of synonymy and polysemy. Synonymy refers to the use of synonyms or different words that have the same meaning, and polysemy refers to words that have different meanings when used in varying contexts. [3] Research shows that on average 80% of the time different people (experts in the same field) will name the same thing differently. [4]

There are a number of existing approaches to solving the word mismatch problem:

#### 2.1.2.1 Stemming

Stemming describes the process of reducing words to its word stem, base or root form, and is a form of query expansion which adds morphologic variations to the query words. [5] Morphological variation is the most obvious and the narrowest form of word mismatch. [5] A stem is the portion of a word after the removal of its affixes (prefixes and suffixes). An example of a stem is the word ***play***, which is the stem for the variants ***playing, played, playful.***

Stems are useful for improving retrieval performance as they reduce variations of the same root form to a common concept. [6]

#### 2.1.2.2 Manual Thesauri

Manual Thesauri are lists of words and their synonyms compiled by domain experts and linguistics. Thesauri are used in query processing where synonyms are added to the original query. [5]

#### 2.1.2.3 Dimensionality Reduction

Dimensionality reduction techniques facilitate the ability to find relevant information without requiring literal word matches. [3] Words and documents are decomposed into vectors in a low dimensional space that represents semantic primitives who can be linearly combined to form the concept, or meaning, of any word, document or query. [5] The word mismatch problem is addressed because any word can be matched with another word to some degree in the low dimensional concept space. [6] Latent Semantic Indexing (LSI) is an example of these techniques. [7]

#### 2.1.2.4 Relevance Feedback

Relevance feedback is a technique that reformulates the original query based on the user's relevance judgment. [8] In a relevance feedback cycle, the most important terms, or expressions, attached to the documents that have been identified as relevant (to the user), enhance the importance of these terms in a modified query formulation. [6] Weights of the query terms are adjusted according to their frequencies in the relevant and non-relevant documents. [5]

### 2.1.3 Query Matching

In an information retrieval system, the user's query representation must be matched with the documents in the collection in order to retrieve the relevant ones. Three different methods can be used to measure the similarity between the queries and the documents.

#### 2.1.3.1 Cosine Coefficient

In the cosine coefficient, the documents and queries are considered as vectors in a term space. A document is considered similar if the angle between the document and the query is small in term space. [3]

Definition:

$$Sim\ (Q, D)\ = \frac{Q.D}{|\mathrm{Q}|.|\mathrm{D}|}$$

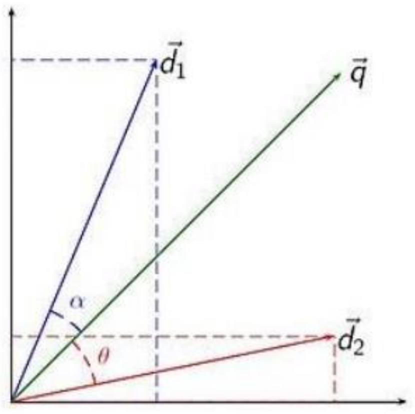


*Figure 2.1 Similarity between documents and query*

Figure 2.1 shows the query $q$ and the document $d_1$ and $d_2$ in the same term space. To simplify, the term space in this figure only consists of two dimensions. The Cosine coefficient calculates the angle $\alpha$ between query vector $q$ and document $d_1$ and the angle $\theta$ between query vector $q$ and document $d_2$. The result shows that, since the angle $\alpha$ is smaller than angle $\theta$, document $d_1$ is a better match to the query $q$ than $d_2$.

#### 2.1.3.2 Jaccard Coefficient

The Jaccard coefficient is a measure that calculates the overlap between two sets. Consider the terms in the document as a set D, and the terms in the query as a set Q. Jaccard then takes the number of terms in the intersection of D and Q and divide it by the number of terms in the union of D and Q.

Definition:

$$Sim\ (\mathrm{Q}, \mathrm{D}) = \frac{|\mathrm{Q} \cap \mathrm{D}|}{|Q \cup D|}$$

Since $|Q \cap D| \leq |Q \cup D|$ the value of Jaccard coefficient ranges from zero to one. If the query and the document contain the same terms the value will be one. If the query and the document have no terms in common the value will be zero. In all other cases, the value will be a number between zero and one.

#### 2.1.3.3 Dice Coefficient

The Dice coefficient, also called Sorensen-Dice coefficient, is very similar to the Jaccard coefficient. Like the Jaccard coefficient, it also takes the overlap between two sets. A difference is that it doubles the value of the intersection and then, rather than dividing it with the union of the sets, it divides with the sum of all elements in both sets. This means that all elements in both sets will be counted twice.

Definition:

$$Sim\ (Q, D) = \frac{2\ |Q \cap D|}{|Q| + |D|}$$

## 2.2 Requirement Analysis

### 2.2.1 Functional Requirements

The functional requirements of this project are as follows:

a. The project must allow the users to enter search query using a web browser.

b. Relevant documents must be retrieved from the corpus upon user's request.

c. The retrieved documents must be ranked according to the relevance to the query.

d. The user must be able to view the retrieved documents.

### 2.2.2 Non-Functional Requirements

The non-functional requirements of this project are as follows:

a. The web application should have a good UI/UX.

b. The response time of the system should be less than 0.01 seconds.

## 2.3 Feasibility Analysis

After gathering of the required resource, whether the completion of the project with the gathered resource is feasible is checked using the following feasibility analysis.

### 2.3.1 Technical Feasibility

The project is implemented as a web application that uses the Flask framework, which is a Python web development framework. It uses Javascript, JQuery, CSS, HTML pages for frontend and Python for the backend. It requires a server, client and Internet connection to function properly. It supports Mac, Windows, and Linux platform for its operation.

The tools, modules, and libraries needed to build the system are open source, freely available and are easy to use. Hence, the project was determined technically feasible.

### 2.3.2 Schedule Feasibility

The schedule feasibility analysis is carried out using the CPM method. With CPM, critical tasks were identified and the interrelationship between tasks was identified which helped in planning that defines critical and non-critical tasks with the goal of preventing time-frame problems and process bottlenecks. The CPM analysis was carried out as follows:

First, the activity specification table with WBS is constructed as shown in the Table:

Table 2.1 - Activity specification with WBS

| **Activity** | **Time(Days)** | **Predecessor** |
|---|---|---|
| Data Collection and Preprocessing (A) | 10 | - |
| Research on previous work and algorithms to be implemented (B) | 14 | - |
| Building TF-IDF vector and LSA model (C) | 7 | A, B |
| User Interface Design (D) | 7 | C |
| System Testing (E) | 10 | D |
| Documentation (F) | 20 | E |

Then, identification of critical path and estimates for each activity are analyzed with the help of a network diagram, which is based on Table 2.1. The network diagram is shown in the figure.

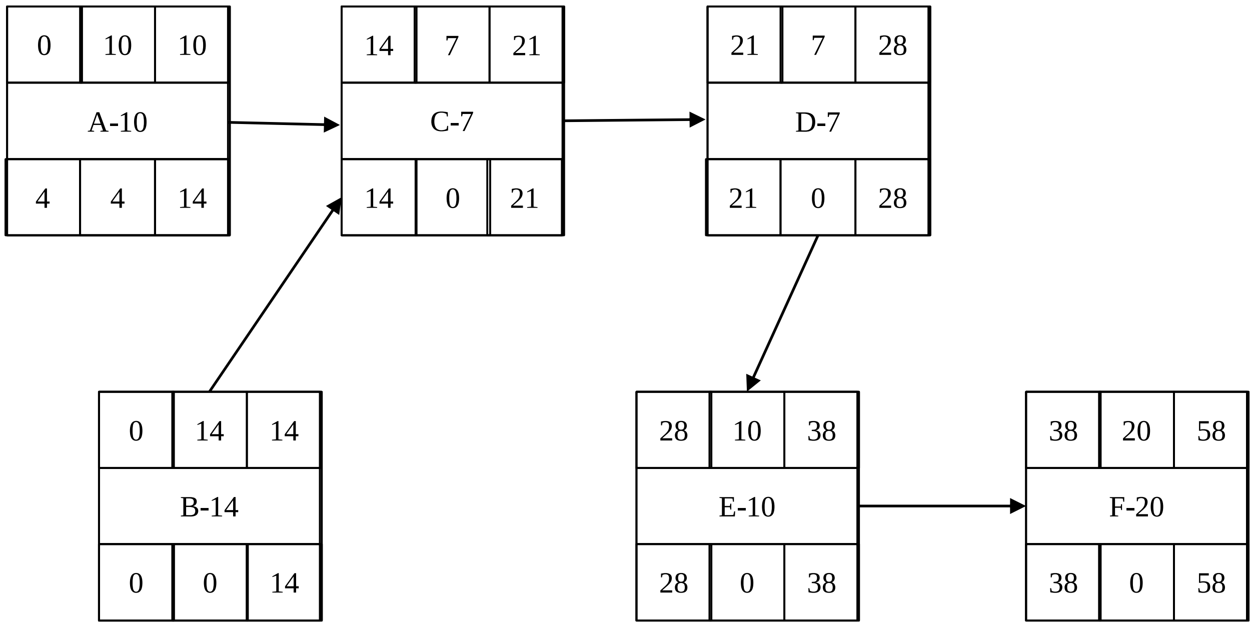


*Figure 2.2 - Activity Network diagram to identify the critical path of the project*

Figure 2.2 shows the Activity Network Diagram of the project. As shown in the above figure, it is observed that the critical tasks are (B) research on previous works and algorithms to be implemented, (C) building TF-IDF vectors and LSA model, (D) user-interface design, (E) system testing and (F) documentation. The total duration of the critical path (B-C-D-E-F) is 58 days, which is in the project deadline range. Hence, this project is feasible in terms of the schedule since the project is completed in time if the critical tasks are carried out within the specified task's duration in Table 2.1.

# CHAPTER 3: SYSTEM DESIGN

## 3.1 Methodology

### 3.1.1 Vector Space Model

#### 3.1.1.1 Representation

The vector space model is an algebraic model for representing both text documents and queries as ordered vectors of terms. [16] With this model a term by document matrix can be represented where each column represents a document and each row represents a term (Figure 3.1). Terms could be words, phrases or concepts. The number of dimensions of the document vectors equals the number of terms in the document collection modeled by the term by document matrix.

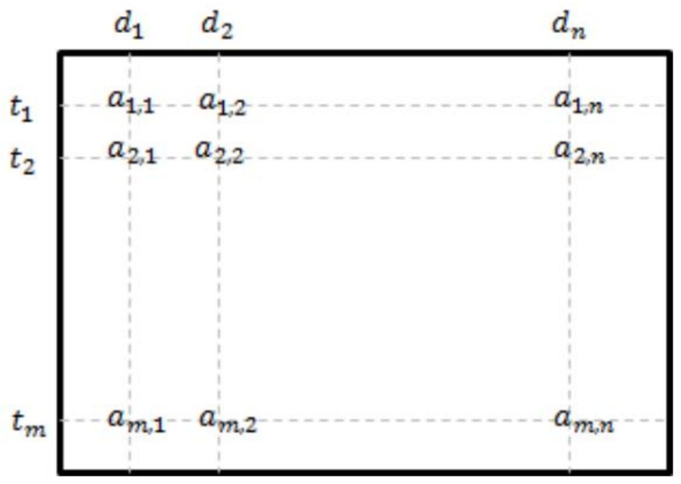


*Figure 3.1 A term by document matrix*

In the term by document matrix, the element $a_{i,j}$ is set to the number of occurrences of term *i* in document *j*. This means that if term *i* is not included in document *j* then $a_{i,j}$ will have the value 0. After the term by document matrix has been constructed, the next step is to assign weights to the terms according to their importance in the document and the overall collection. This process is called weighting and can have a major impact on the performance of the Vector Space Model. [15]

$\mathrm{w}_{i,j} = \mathrm{F}(a_{i,j})$, F is a weighting function

When all terms $a$ have been weighed, the term by document matrix will look like (Figure 3.2), where $w_{i,j}$is the weighted term $a_{i,j}$.

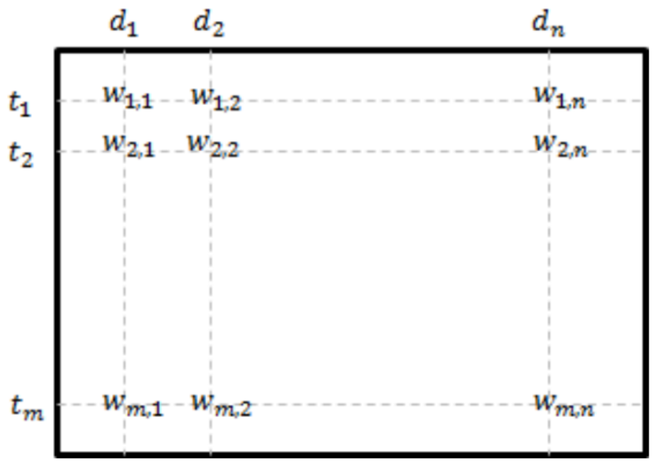


*Figure 3.2 A weighted term by document matrix*

Queries are represented in the same way as documents in the Vector Space Model. A query is represented by a vector that contains the weighted terms of the query.

$$d_i = (W_{1,i}, W_{2,i}, \dots, W_{m,i}) \qquad (1)$$

$$q = (W_{1,q}, W_{2,q}, \dots, W_{m,q}) \qquad (2)$$

### 3.1.2 TFIDF Weighting

TFIDF (Term Frequency–Inverse Document Frequency) is a statistical numeral that shows how important a word is to a document in a corpus. [17] It is often used as a weighting factor in searches of information retrieval, and text mining. The TFIDF value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general.

TFIDF is one of the most popular term-weighting schemes today; 83% of text-based recommender systems in digital libraries use TFIDF. [18]

### 3.1.3 Latent Semantic Indexing

Latent Semantic Indexing (LSI) or Latent Semantic Analysis (LSA) is an indexing and retrieval method that is based on the principle that words used in the same context tend to have the same meaning. [6] It uses a mathematical technique called Singular Value Decomposition (SVD) to identify patterns in the relationships between the terms and the concepts contained in an unstructured collection of text. [6]

LSI solves the problems of synonymy and polysemy. These issues lead to mismatches in the vocabulary used between the documents in the collection and in the queries being performed, resulting in both low precision and recall for these queries. [7]

LSI is a pure mathematical method and it does not rely on any knowledge about the text. Hence, it works well with any language and it can even be used to find documents across languages, especially in scientific collections where a lot of the terminology is the same between languages. [6] LSI is also very tolerant to noise, like misspelled words, and it adapts well to changes in the terminology used in the data collection. [7]

#### 3.1.3.1 Algorithm

LSI uses common linear algebra techniques to learn the conceptual correlations in a collection of text. In general, the process involves constructing a weighted term by document matrix and then use rank reduced singular value decomposition to construct a low-rank approximation of this matrix. The idea is that this truncated matrix contains information about the concepts contained in the text.

**Rank-reduced Singular Value Decomposition**

The rank-reduced singular value decomposition is performed on the weighted term by document matrix to determine patterns in the relationships between the terms and concepts contained in the text. The SVD forms the foundation for LSI. It calculates the term and document vector spaces by approximating the weighted term by document matrix, *A*, into three other matrices:

I. *m* by *r* term-concept vector matrix *T*,

II. *r* by *r* singular value $\Sigma$, and

III. *r* by *n* concept-document vector matrix, *D*

which satisfy the following constraints:

$A = T\sum D^T$ where,

$T^T T = I_r$ , $D^T D = I_r$ , $\Sigma_{1,1} \geq \Sigma_{2,2} \geq \ldots \geq \Sigma_{r,r} > 0, \Sigma_{i,j} = 0$ *where* $i \neq j$

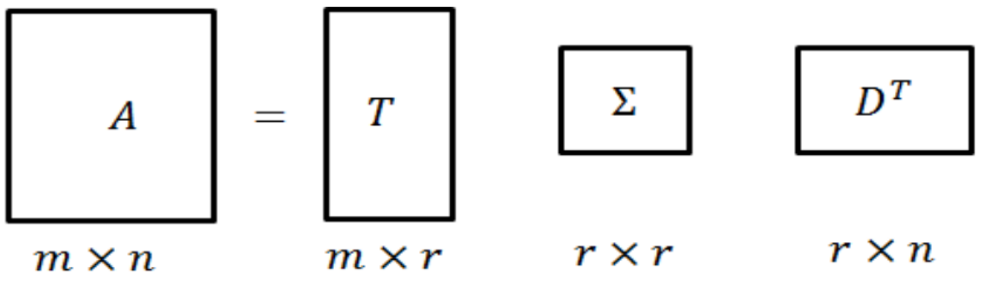


*Figure 3.3 Mathematical representation of matrix A*

Figure 3.3 shows *A* being the supplied *m* by *n* weighted term by document matrix derived from a collection of text where *m* is the number of unique terms in the collection, and *n* is the number of documents in the collection. *T* is a computed *m* by *r* matrix of term vectors where *r* is the rank of *A*. $\Sigma$ is a computed *r* by *r* diagonal matrix of decreasing singular values, and *D* is a computed *n* by *r* matrix of document vectors.

The LSI modification to a standard SVD is to reduce the rank or truncate the singular value matrix $\Sigma$ to size $k << r$, typically the order of $k$ is in the range of 100 to 300 dimensions, effectively reducing the term and document vector matrix sizes to *m* by *k* and *k* by *n* respectively. The SVD operation, along with this reduction, has the effect of preserving the most important semantic information in the text. This reduced set of matrices is often denoted with a modified formula such as:

$$A \approx A_k = T_k \Sigma_k D_k^T$$

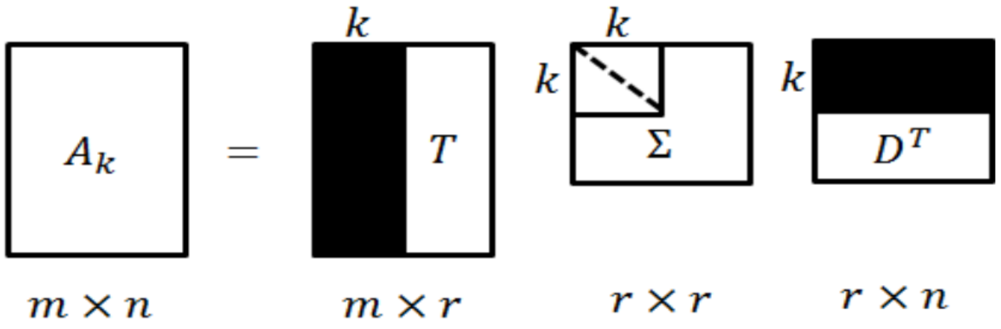


*Figure 3.4 Mathematical representation of matrix* $A_k$

Figure 3.4 shows the mathematical representation of the matrix $A_k$. $T$ and $D$ are the term and document vectors and $\Sigma$ represents the singular values. The shaded regions represent $A_k$from above equation.

The computed $T_k$ and $D_k$ matrices define the term and document vector spaces, which with the computed singular values, $\Sigma_k$ embody the conceptual information derived from the document collection. The similarity of terms or documents within these spaces is a factor of how close they are to each other in these conceptual spaces, usually this is computed as a function of the angle between the corresponding vectors.

To transform the original query $q$ to a query $\hat{q}$ for the k-dimensional space, the following formula is used: $\hat{q} = q^T T_k \Sigma_k^{-1}$

### 3.1.4 Cosine Distance

The cosine similarity between a query vector $Q$ and document vector $D$ is given by

$$Sim\ (Q, D)\ =\ \frac{Q.D}{|\mathrm{Q}|.|\mathrm{D}|}$$

And the cosine distance is given by

$$\text{Cosine Distance} = 1 - Sim(Q, D)$$

The cosine distance has been the most popular similarity measure to use for the Vector Space Model. A reason for this, besides its good matching performance, is that it can be easily calculated with vector operations. [6]

### 3.1.5 Performance Evaluation

Efficient retrieval models are expected to produce results that correlate well with human decisions on relevance. In other words, ranking algorithms built on good retrieval models will retrieve relevant documents and have high effectiveness. [2]

In one aspect, the ultimate evaluation of any information retrieval system is determined by whether the user is satisfied with the results of the information need requested. [3] However, it is not easy for a user to explain why one document is more relevant than the other, i.e. relevance judgments for a given query often differ when defined by different users. [2]

Although there has been some debate over the years, the two desired properties that have been accepted and most commonly used by the research community for measurement of search effectiveness are recall and precision. [9]

#### 3.1.5.1 Recall and Precision

The two most common measures, precision and recall, initially introduced in the Cranfield studies [2], are both based on the concept of relevance to a given query or information need. [10] An effective retrieval system should retrieve as many relevant documents as possible, i.e. have a high recall, while it should retrieve as few non-relevant documents, i.e. have high precision. [9]

**Precision**

The precision or precision ratio $P$ of a search method is defined as $P = \frac{D_r}{D_t}$

where $D_r$ is the number of relevant documents retrieved and $D_t$ is the total number of documents retrieved.

Although precision is a formal evaluation measure it does not say anything about the retrieval algorithm's way of ranking documents nor does it state anything about the user's interpretation of that ordering. [6] Relevance is still a judgment call, that is, relevance is subjective and dependent on who is keeping score. [3]

**Recall**

Recall or recall ratio R of a search method is defined as $R = \frac{D_r}{N_r}$

where $D_r$ is the number of relevant documents retrieved and $N_r$is the total number of relevant documents in the collection.

Recall ratios are somewhat difficult to obtain, as the total number of relevant documents is always an unknown. [6]

In an information retrieval system, a user is not usually presented with all the documents in the answer set at once. Analyzing the documents one by one starting from the top according to the degree of relevance, recall and precision measures vary as the user proceeds with his examination of the retrieved answer set. [6]

For large scale systems retrieval algorithms are evaluated by running batches of distinct queries. Average precision is a single-valued measure most commonly used to evaluate ranked retrieval performance. It is computed by measuring precision at different recall levels (10%, 20%, and so on) and averaging. [16]

Definition

Let $r_i$ denote the number of relevant documents up to and including position $i$ in the (ordered) returned list of documents. The recall at the $i^{th}$ document in the list is the proportion of relevant documents seen thus far, i.e. $R_i = \frac{r_i}{r_n}$. The precision at the $i^{th}$ document, $P_i$ is defined to be the proportion of documents up to including position *i* that are relevant to the given query. Hence, $P = \frac{r_i}{i}$

### 3.1.5.2 Focusing on Top Documents

In many real-world search systems, users tend to look at only the top-ranked documents of the retrieved result set. Under these circumstances, recall is just not an appropriate measure. Instead, the focus should be on how effective the system is on retrieving relevant documents at very high ranks, i.e. near the top of the ranking.

One measure which captures this concept is precision at rank *p*. [2] The idea here is to calculate recall-precision values at a small number of predefined rank positions. To compare retrieval rankings for a distinct query, only the precision at the predefined rank positions needs to be computed. A higher precision value at rank position *p* implies a higher recall value as well. The value of *p* is arbitrary, but since this measure is typically used to evaluate output at high rankings most common values are 10 and 20.

## 3.2 System Design

### 3.2.1 Use Case Diagram

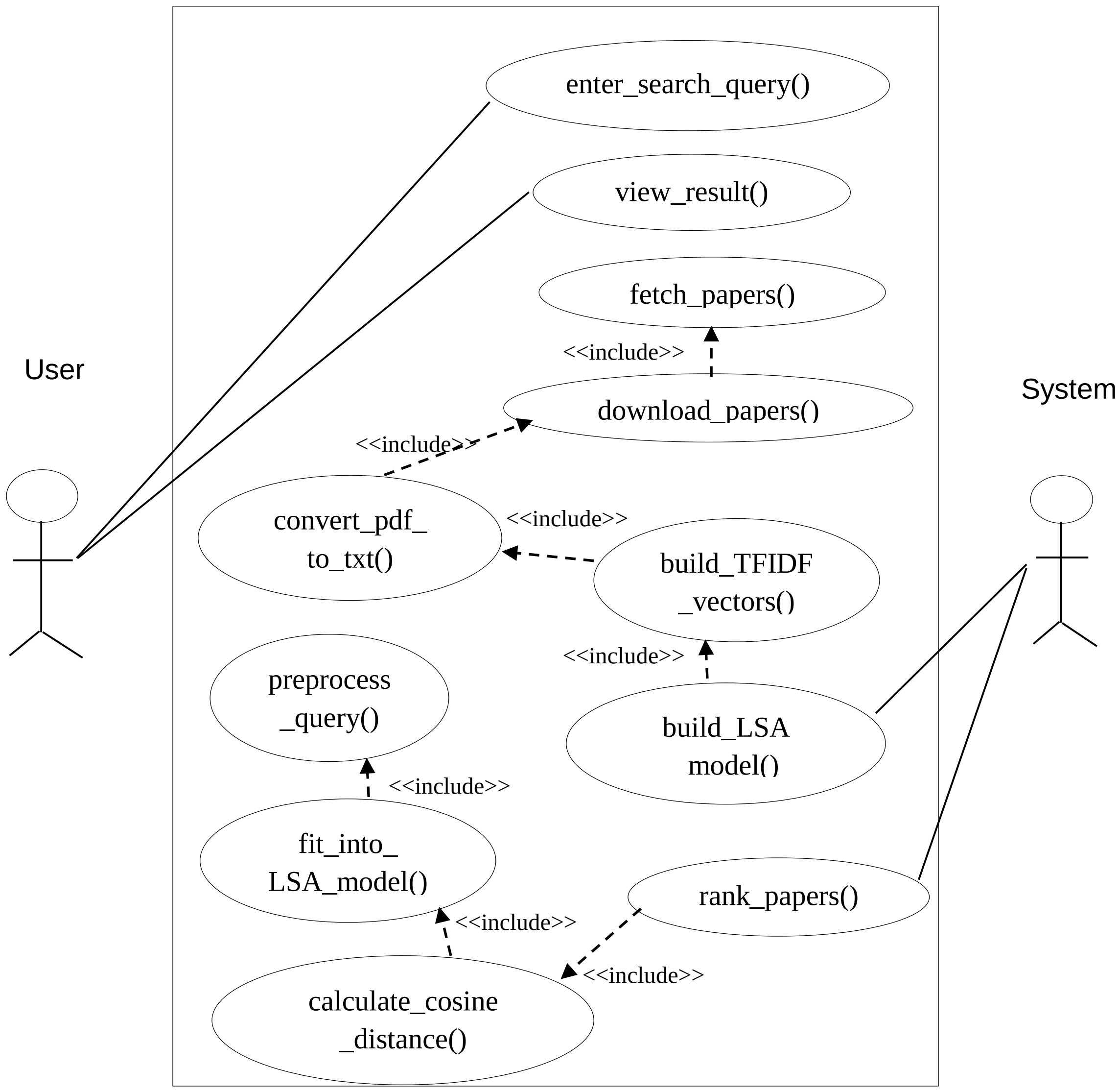
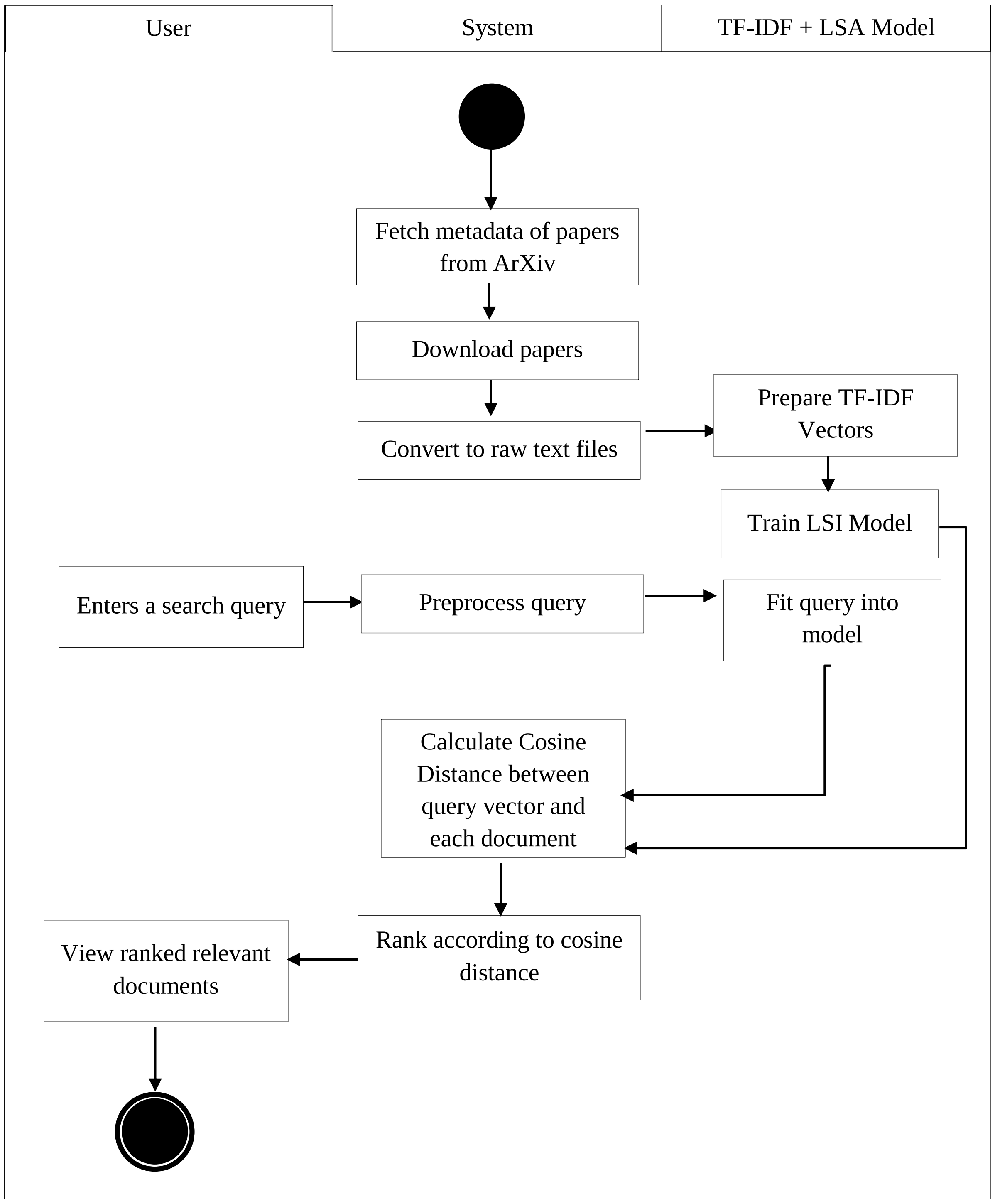


*Figure 3.5 Use Case Diagram of Information Retrieval System*

Figure 3.5 explains the use case diagram of the system. Two use cases namely, enter_search_query() and view_result() are triggered by the user. All other functionalities are performed by the system.

## 3.2.2 Activity Diagram



User
System
TF-IDF + LSA Model
Fetch metadata of papers from ArXiv
Download papers
Convert to raw text files
Prepare TF-IDF Vectors
Train LSI Model
Enters a search query
Preprocess query
Fit query into model
Calculate Cosine Distance between query vector and each document
View ranked relevant documents
Rank according to cosine distance

*Figure 3.6 Activity Diagram of the system*

Figure 3.6 explains the activity of the system and showcases the flow the system. The documents are fetched, downloaded, and preprocessed by the system and LSA model is built. The user enters a search query which pre-processed and fitted into the built model before calculating the cosine distance for ranking.

# CHAPTER 4: IMPLEMENTATION AND TESTING

## 4.1 Implementation

The information retrieval system is implemented as a web application using Flask Framework in Python 3.6. The corpus of documents used in the system constitutes of 6900 PDF version of machine learning research papers published on ArXiv that were downloaded using ArXiv's API. The PDF documents are first converted to raw files and then a TF-IDF vector is built using the popular machine learning Python module scikit-learn. Next, the TF-IDF vector is decomposed to a 100-rank matrix using Singular Value Decomposition method.

The user can enter a search query using the user interface of the web application. The user's query also goes through the same pipeline as that of the documents. Finally, cosine distance is calculated between the query vector and each document vector to produce a rank the documents in order of relevance (cosine distance) to that of the query vector.

### 4.1.1 Tools Used

**CASE tools:**

a) Draw.io

**Client Side:**

1. HTML is used to display content in the browser.
2. CSS is used to adjust the layout, look and design of the HTML content.
3. Javascript and jQuery are used to enhance the user experience and implement lazy loading of the retrieved documents.

**Server Side:**

1. Python programming language is used to implement the core program logic.

2. Flask web framework is used for dynamic web page generation and to display the ranked documents in the browser as well as to handle search queries.

3. pdftotext is used to convert PDF files into raw text files.

4. Scikit-learn is used for the major functionalities of the system such as text-preprocessing, building TF-IDF and LSA model, and calculating the cosine distance.

## 4.2 Description of Major Files and Methods

**File: fetch_papers.py**

Fetch metadata of all the research papers in machine learning category published on ArXiv and arrange the data using Python in-built data structure, *dictionary*, and store it locally as a pickle file '*db.p'*.

**File: download_papers.py**

Download all the research papers listed in the pickle file *'db.p'*

**File: pdf_to_txt.py**

Convert all the research papers in PDF file format to a raw text file using **pdftotext** module. If the PDF file is not convertible, skip it.

**File: analyze.py**

In this file, the TF-IDF and LSI model is built. The major functions used are:

I. TfidfVectorizer()

TfidfVectorizer is a method provided by the scikit-learn machine learning Python library. It used here to compute the TF-IDF vectors. The method is tuned to use built-in stopwords dictionary, convert all text to lowercase, and smoothen the idf value. Smoothening the idf weights by adding one to document frequencies, as if an extra document was seen containing every term in the collection exactly once, prevents zero divisions.

II. TruncatedSVD()

TruncatedSVD is another method from the scikit-learn machine learning Python library. This transformer performs linear dimensionality reduction by means of truncated singular value decomposition (SVD).

The built models are saved as pickle files.

**File: views.py**

This file handles all the requests from the client in the Flask app environment. It pre-loads the LSI model built by *analyze.py*. The major function in this file is papers_search(). This function is called when user enters a search query from the client side. The function performs pre-processing on the search query and builds the TF-IDF vector and fits it into the LSI model. Now the cosine distance is calculated between the decomposed query vector and each decomposed document vector.

Finally, the retrieved documents is sorted according to the cosine distance value and result is displayed in the client side.

## 4.3 Testing

The system was tested by building the LSA model with 100-rank decomposition. The model achieved to get Mean Average Precision score of 64% on the LISA dataset.

### 4.3.1 Testing parameters

In this section, the chosen values for the hyperparameters are listed. Due to limitations, such as time constraints, not all parameter values could be chosen based on experiments. Furthermore, no experiments were carried out on parameters that were expected to have no or very little impact on the performance of the algorithm.

Table 4.1 List of chosen parameters

| | |
|---|---|
| Number of Queries | 5 |
| Total rank positions considered | 10 |
| Query Type | Long |
| Total Documents | 6000 |

### 4.3.2 Results

Table 4.2 Precision of each query at different ranks, k

| Rank\Query | I | II | III | IV | V |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 0.5 | 0.5 | 0.5 | 1 |
| 3 | 0.67 | 0.33 | 0.33 | 0.67 | 1 |
| 4 | 0.5 | 0.5 | 0.25 | 0.75 | 1 |
| 5 | 0.4 | 0.4 | 0.4 | 0.8 | 1 |
| 6 | 0.5 | 0.33 | 0.33 | 0.83 | 0.83 |
| 7 | 0.43 | 0.43 | 0.43 | 0.86 | 0.86 |
| 8 | 0.38 | 0.38 | 0.5 | 0.88 | 0.88 |
| 9 | 0.33 | 0.33 | 0.56 | 0.78 | 0.89 |
| 10 | 0.3 | 0.3 | 0.5 | 0.8 | 0.9 |

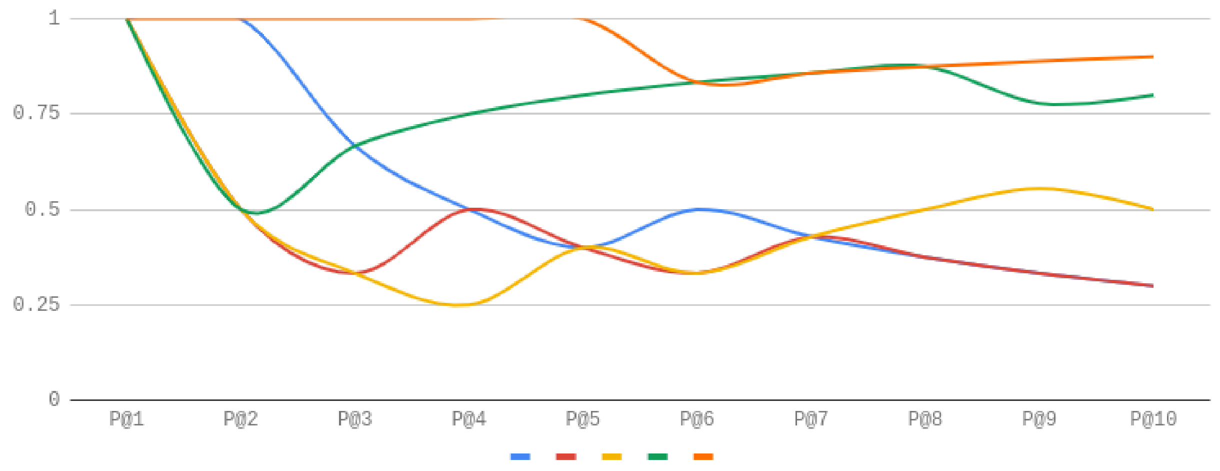


Figure 4.1 Line graph representing precision at different ranks of different queries

Figure 4.1 shows the progression of precision score of each query as the rank starts increasing from 1 to 10.

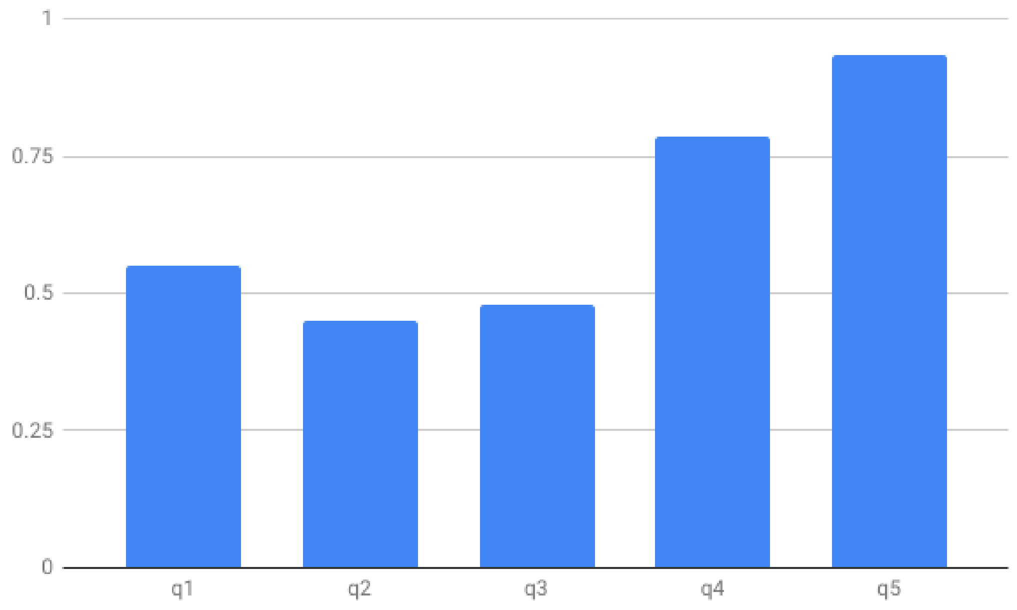


Figure 4.2 Average Precision score of each query upto rank, k = 10

Figure 4.2 shows the average precision score of each query. The average precision score is calculated according to the data shown in Table 4.2.

# CHAPTER 5: CONCLUSION AND RECOMMENDATION

## 5.1 Conclusion

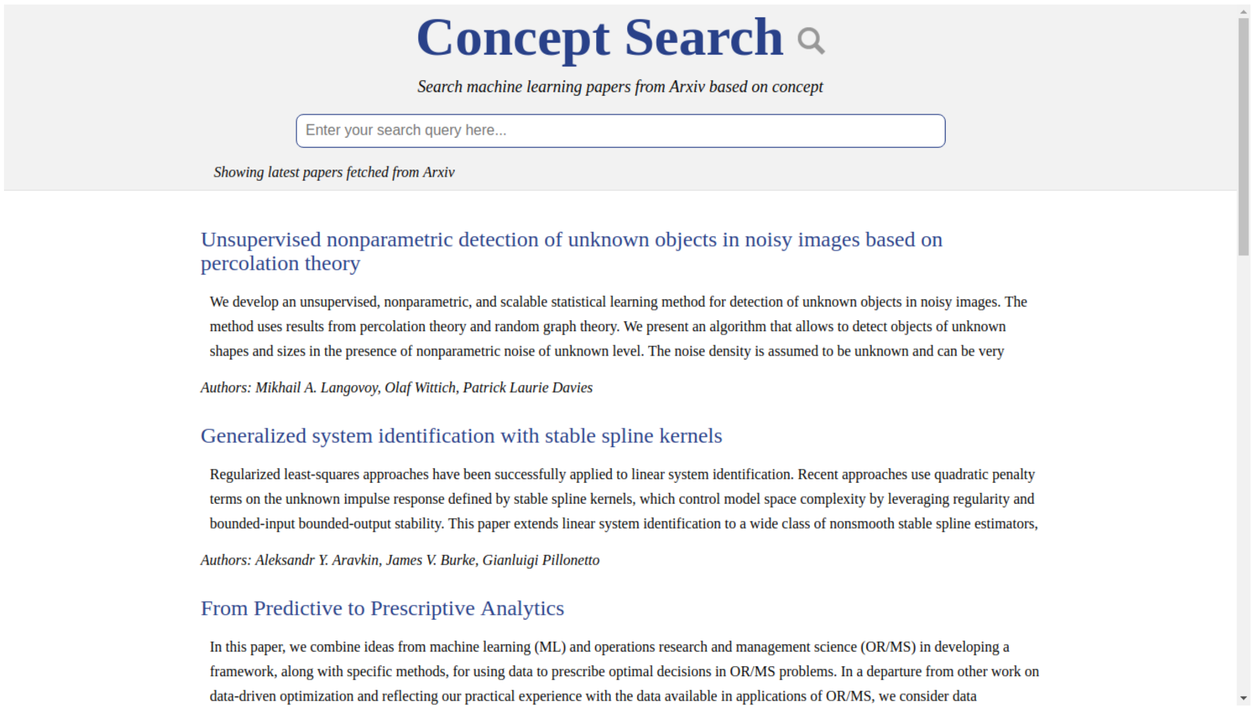
A web-based information retrieval system was successfully built using latent semantic indexing. The system used a corpus containing more than 6500 documents. The LSI model successfully handled the problems of vocabulary-mismatch i.e. problems of synonymy and polysemy. However, statistical data regarding its performance with vocabulary-mismatch could not be obtained due to lack of labelled dataset.

In regards to the performance of the system, the model was able to achieve a Mean Average Precision score of 64%. This result was obtained by performing search query with only five long search queries up to rank 10.

## 5.2 Recommendation

The performance of the system can be improved by using more advanced weighting schemes such as BM-25 or other variants of the TFIDF weighting scheme. Another key area to improve is spelling errors in search query. A concept such as Fuzzy search can be implemented to correct the spelling errors.

# APPENDIX


Concept Search

*Search machine learning papers from Arxiv based on concept*

Enter your search query here...

*Showing latest papers fetched from Arxiv*

Unsupervised nonparametric detection of unknown objects in noisy images based on percolation theory

We develop an unsupervised, nonparametric, and scalable statistical learning method for detection of unknown objects in noisy images. The method uses results from percolation theory and random graph theory. We present an algorithm that allows to detect objects of unknown shapes and sizes in the presence of nonparametric noise of unknown level. The noise density is assumed to be unknown and can be very

*Authors: Mikhail A. Langovoy, Olaf Wittich, Patrick Laurie Davies*

Generalized system identification with stable spline kernels

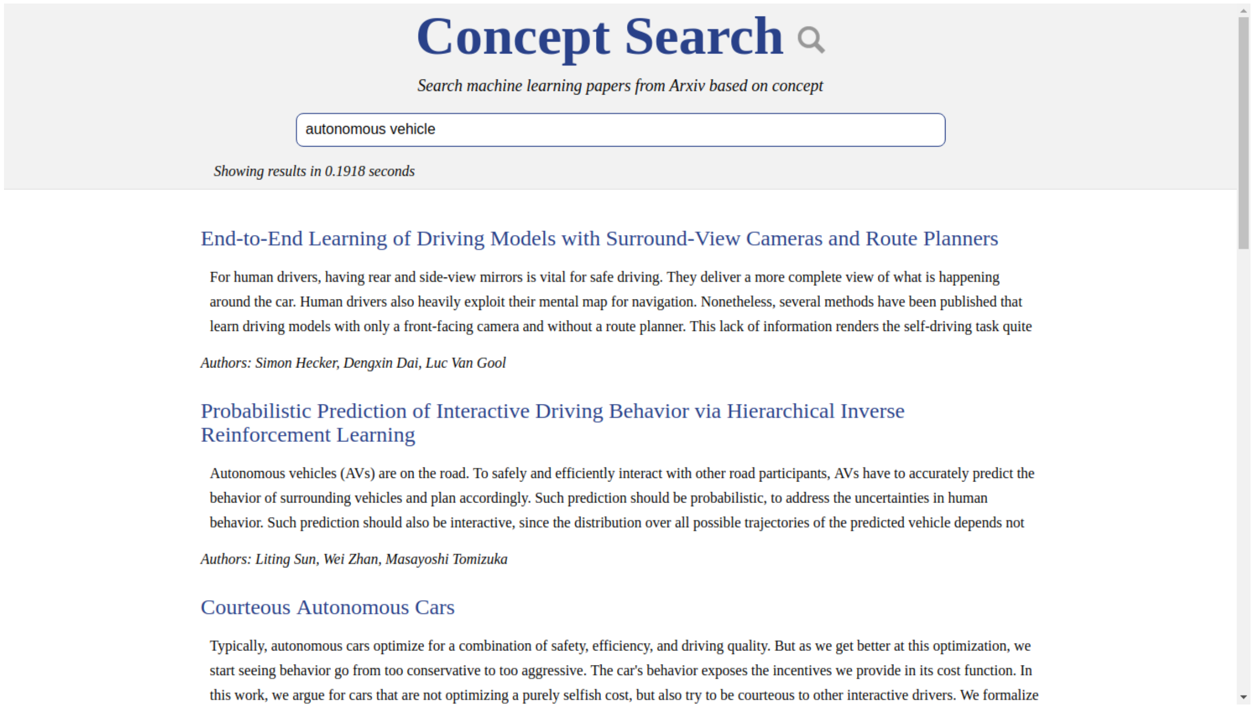
Regularized least-squares approaches have been successfully applied to linear system identification. Recent approaches use quadratic penalty terms on the unknown impulse response defined by stable spline kernels, which control model space complexity by leveraging regularity and bounded-input bounded-output stability. This paper extends linear system identification to a wide class of nonsmooth stable spline estimators,

*Authors: Aleksandr Y. Aravkin, James V. Burke, Gianluigi Pillonetto*

From Predictive to Prescriptive Analytics

In this paper, we combine ideas from machine learning (ML) and operations research and management science (OR/MS) in developing a framework, along with specific methods, for using data to prescribe optimal decisions in OR/MS problems. In a departure from other work on data-driven optimization and reflecting our practical experience with the data available in applications of OR/MS, we consider data

Figure A.1 Landing Page of the Web Application


Concept Search

*Search machine learning papers from Arxiv based on concept*

autonomous vehicle

*Showing results in 0.1918 seconds*

End-to-End Learning of Driving Models with Surround-View Cameras and Route Planners

For human drivers, having rear and side-view mirrors is vital for safe driving. They deliver a more complete view of what is happening around the car. Human drivers also heavily exploit their mental map for navigation. Nonetheless, several methods have been published that learn driving models with only a front-facing camera and without a route planner. This lack of information renders the self-driving task quite

*Authors: Simon Hecker, Dengxin Dai, Luc Van Gool*

Probabilistic Prediction of Interactive Driving Behavior via Hierarchical Inverse Reinforcement Learning

Autonomous vehicles (AVs) are on the road. To safely and efficiently interact with other road participants, AVs have to accurately predict the behavior of surrounding vehicles and plan accordingly. Such prediction should be probabilistic, to address the uncertainties in human behavior. Such prediction should also be interactive, since the distribution over all possible trajectories of the predicted vehicle depends not

*Authors: Liting Sun, Wei Zhan, Masayoshi Tomizuka*

Courteous Autonomous Cars

Typically, autonomous cars optimize for a combination of safety, efficiency, and driving quality. But as we get better at this optimization, we start seeing behavior go from too conservative to too aggressive. The car's behavior exposes the incentives we provide in its cost function. In this work, we argue for cars that are not optimizing a purely selfish cost, but also try to be courteous to other interactive drivers. We formalize

Figure A.2 – Showing top 3 relevant documents for query ‘autonomous vehicle’

# REFERENCES

[1] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. An Introduction to Information Retrieval, 2009

[2] W. Bruce Croft, Donald Metzler, and Trevor Strohman. Search Engines: Information Retrieval in Practice, 2010

[3] Michael W. Berry and Murray Browne. Understanding Search Engines: Mathematical Modeling and Text Retrieval, 1999

[4] G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais. The Vocabulary Problem in Human-System Communication, 1987

[5] Jinxi Xu. *Solving The Word Mismatch Problem Through Automatic Text Analysis*, 1997

[6] Magnus La Fleur, Fredrik Renström. Conceptual Indexing using Latent Semantic Indexing, 2015

[7] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by Latent Semantic Analysis, 1990

[8] Gerard Salton and Chris Buckley. Improving Retrieval Performance by Relevance Feedback, 1990

[9] Amit Singhal. Modern Information Retrieval: A Brief Overview, 2001

[10] Ed Greengrass. Information Retrieval: A Survey, 2000

[11] Frieder Grossman. Information Retrieval, 2004

[12] Ricardo Baeza-Yates and Berthier Ribiero-Neto. Modern Information Retrieval, 1999

[13] Gerard Salton, A. Wong, and C. S. Yang. A vector space model for information retrieval, 1975

[14] Stephen. E. Robertson and Karen Sparck Jones. Relevance weighting of search terms, 1976

[15] Gerald Salton and Christopher Buckley. Term-Weighting Approaches in Automatic Text Retrieval, 1988

[16] Gerard Salton and M. J. McGill. Introduction to Modern Information Retrieval, 1983

[17] A. Rajaraman and J.D. Ullman, *"Data Mining – Mining of massive datasets",* 2011

[18] Corinna Breitinger, Bela Gipp, and Stefan Langer, *"Research-paper recommender systems: a literature survey".* International Journal on Digital Libraries, 2015