الجامعة العربية الأمريكية
ARAB AMERICAN UNIVERSITY

Arab American University

Faculty of Graduate Studies

Density-Based Approach for Information Retrieval Documents Ranking in Embedding Space

By

Loai Bitawi

Supervisor

Prof. Giovanni Stilo

This thesis was submitted in partial fulfillment of the requirements for the Master's degree in Data Science and Business Analytics.

12 / 2021

Thesis Approval

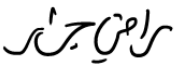
Density-Based Approach for Information Retrieval

Documents Ranking in Embedding Space

By

Loai Bitawi

This thesis was defended successfully on 12/2/2022 and approved by:

Committee members	Signature
1. Giovanni Stilo/Supervisor	
2. Mohammed Maree/Internal Examiner	
3. Radi Jarrar/External Examiner	

Declaration

I, Loai Mohammad Ali Bitawi, one of the students of the Faculty of Graduate Studies at the Arab American University hereby declare that this thesis entitled " Density-Based Approach for Information Retrieval Documents Ranking in Embedding Space ", is all by my own work and the resources that are used in this thesis (including the internet resources) have been referred to and properly acknowledged as required.

I declare that I have fully understood the concept of plagiarism and I acknowledge that my thesis will be immediately rejected in case of including any type of plagiarism.

Loai Mohammad Ali Bitawi

Signature: 

Date: 8/6/2022

Acknowledgements

Through the journey of this master thesis, I received a lot of support and assistance. I would like to thank my supervisor, Prof. Giovanni Stilo[1], who has been extremely supportive, motivating, and showed the best example in commitment and engagement. His comments and suggestion showed his experience and out-of-the-box thinking, which enlightened my path through the thesis.

I would like to thank my family, who supported me and encouraged with their love and support through my master's journey. A special thanks to my first teachers, mom and dad, who were and still my best support in my whole life. I would like to single out my fiancée, the one whom her love was the power for me to work harder and harder, and stayed beside me through all the times.

A special thanks to my best friends for their support, and my professors at the master's program. A special thanks to prof. Amjad Abu-Rmileh for his support and guidance.

Finally, I would like to thank my second family PALTEL for the opportunity to complete my master's program with partial fund for tuition and their technical facilitation for the research and development. A special thanks to the GM Mr. Maen Melhem for the opportunity and the fund, and IT director Mr. Nizar Shanaah for his great support.

[1] Prof. Stilo is partially supported by Territori Aperti a project funded by Fondo Territori Lavoro e Conoscenza CGIL CISL UIL.

Abstract

Search engines play an essential role in satisfying our daily life needs for information. Thus, optimizations on algorithms encapsulated in these engines are needed. Literature shows that neural networks and VSM are the best models to be used in IR systems. Moreover, the studies haven't mentioned the use of density-based algorithms for document matching. This research explores an untapped area where the density-based algorithm is used for document-query-matching. The proposed model represents document tokens using BERT embeddings. Thus, documents are represented as clouds of points. The matching part uses a hybrid solution that enhances the recall scores and reduces runtime. The model uses the BM25 ranking model to rank the whole dataset and use only the top *K* documents and pass them to the density-based algorithm. The density-based model is inspired by the anomaly detection algorithm

(LOF) that aims to find if a query point is an outlier to the document cloud. Since the LOF score measures the degree of outlier-ness, the points scores are used to find the score of the document to a certain query. Then, the document scores are used to rank the documents. Moreover, the density algorithm includes term weighting using TFIDF, which focuses on the most important words in the documents and reduces the effect of words that are irrelevant to the document topic. The proposed algorithm is tested against the state-of-the-art mean of embeddings model. The results show that the proposed algorithm outperforms the mean of embedding model on several datasets and testing scenarios. The algorithm shows high potential for future investigations and optimizations that would introduce a better document-query matching model for IR systems.

Contents

List of Tables

List of Figures

List of Equations

Chapter 1: Introduction

In our daily life, search engines play a crucial part in retrieving information about a certain topic. However, search engines might sometimes provide irrelevant results to the search context. Such results might be since the query is written in human's natural language, which is different from the way written in the documents. Another reason might be the query keywords match documents that have different topics. Still, the search engine finds the same similarity between the query and each of the documents.

These problems are induced by synonymy and polysemy issues. Many representation algorithms for both queries and documents tried to overcome these issues. One of the most widely used representations is Vector Space Models (VSM).

Vector space models represent both document and query as a single vector. In classic VSM each dimension represents a term of the vocabulary (extracted from all documents). This representation suffers from sparsity and high-dimensionality. The more documents in the database imply more terms in the lexicon, which requires higher dimensions for representation. Thus, finding similarity/distance between document and query is a well-known research problem. Many similarity measures such as Cosine, Jaccard, and Dice can estimate how close/far the document is from the query. However, these similarities differ in terms of performance from each other, and each measure differs in its performance depending on the domain it is used in. The differences in approaches and their difference in performance lead to the research question that this research is trying to find; Which is the most effective matching model to use when documents and queries are represented in the eVSM?

According to the literature, there is no one final answer to what is the best document – query matching algorithm. Each algorithm comprises its advantages and disadvantages.

For example, VSM suffers from sparsity and the curse of dimensionality. On the other hand, TF-IDF misses semantic meanings. Moreover, similarity measures vary in performance when applied to different representation models.

To answer this question, the proposed model in this thesis tries to answer the question by overcoming the following problems:

1. The sparsity and dimensionality, by using word embeddings (taken from embedding algorithms like Word2Vec, BERT Embeddings... etc.) to represent the document as a set of points (vectors) in the space model. The same is for the query. This method reduces the dimensionality and preserves the semantic meaning of the terms.

2. Studying the performances of the proposed density-based approaches by comparing it with other classical similarity measures.

The research introduces a new query/document representation model called Extended Vector Space Model (eVSM). Also, a new density-based matching algorithm is introduced to rank the documents among queries. The contribution to the body of knowledge is represented by the novel algorithm that significantly improves the information retrieval field of study. Moreover, it explores a specific area in information retrieval where density measures are not previously used to find the similarity/distance between the documents and the query. We like to note that this new approach is permitted by the paradigm shift introduced by the eVSM.

Chapter 2: Related Work

2.1 Information Retrieval

Information retrieval (IR) models are widely used in daily life since they are needed by many natural language-related tasks; a classic example is the search engines. An academic definition of information retrieval is ”finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)” (Manning et al. 2010). From the definition, the textual IR model consists of three main components: Documents, Query, and the matching model.

At first, a document is an unstructured text describing one or more topics. The document can be divided into terms (or words), which are recognizable elements that form a language. A large set of documents, which are the elements that the IR system is built on, and can be denoted as $D = \{D_1, D_2, D_3, ..., D_n\}$, where n is the size of the set is referred to as a corpus or a collection (Manning et al. 2010).

Secondly, a query, which is a short text (compared to the document) that contains what a user passes to the system to match one or more of the documents, and can be denoted as Q. However, the query is different than the information needed; The user sometimes enters a query in a way that returns undesired documents. It doesn’t mean that the IR system performance is low, but the user’s need is different from his input to the system. A relevant document is a document that contains the desired information requested by the user in the query.

Finally, a matching model, which is the goal of the IR model, is the way (algorithm) that aims to match the query Q with document set D and retrieve the most relevant K documents.

Since both document and query are in an unstructured manner, there will be a need for a numerical representation for them to find a match. In this context, a dictionary of terms that contain all mentioned words in all document sets. This dictionary is called a vocabulary or a lexicon (Manning et al. 2010), and is denoted as L. For example, suppose:

D_1:" The boy is playing in the park"

D_2:" A taxi is waiting for the passenger"

Then, the lexicon L = {*a, boy, for, in, is, park, passenger, playing, taxi, the, waiting*}. IR models differ in terms of performance, complexity, and energy cost. IR model's performances are mainly affected by the chosen representation model. The representation model allows representing both documents and queries unstructured by their nature in a more feasible way.

IR performance is also affected by the similarity measure (a matching mechanism), which represents how close/far is the document from the query. Many researchers discussed both representation models and similarity measures and compared various techniques to find an IR model that comprises the best performance.

There are many representation models for implementing information retrieval systems, but the main models are Boolean, vector, and inference network models (Roshdi & Roohparvar 2015). However, in This research, we're interested in Vector models, which are described in the next section. The performance of the IR model is mainly measured by two main metrics: Precision and recall.

Precision measures the amount of relevant retrieved documents from all retrieved documents, while Recall measures the amount of relevant retrieved documents from all relevant documents that should be returned.

To perform the mentioned evaluation metrics, the data should be associated with a set of queries and a relevance assessment should be performed, which lists all relevant documents for each query. In this way, both precision and recall can be measured.

Moreover, a combination of the mentioned metrics would express the overall performance of the IR model. This combination is referred to as F-score. Figure 2.1 illustrate the precision and recall.

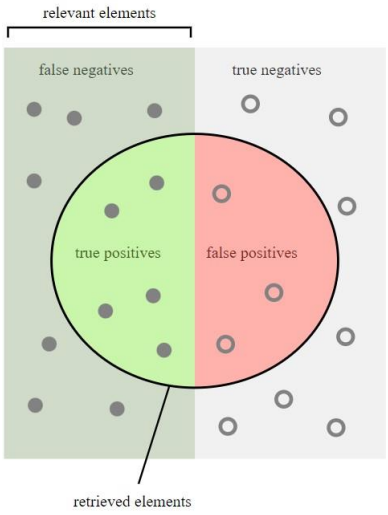


Figure 2.1 : Precision and Recall Illustration

There is a trade-off between precision and recall since as precision increases the recall decreases. This happens because as precision increases, the number of retrieved elements are less, with larger number of true positives and less false positives, resulting in increasing number of true negatives, and this reduces the recall values. So, F-Score tries to calculate a harmonic average between both metrics to find one measure to represent the performance of the model. Also, there is a trade-off point between precision and recal, which can be called an equilibrium point, where precision is the highest point while maintaining good recall or vice-versa. This point is achieved by trying several thresholds for the model and calculate both precision and recall, then draw both lines, and find the intersection between them.

Another metric used in evaluating IR models is the accuracy, which measures the amount of correct classification of documents.

According to (Manning et al. 2010), a mathematical formulation of the scores can be:

$$Precision(P) = \frac{tp}{tp+fp} \tag{2.1}$$

$$Recall(R) = \frac{tp}{tp+fn} \tag{2.2}$$

$$F - Score(F) = \frac{2*P*R}{P+R} \tag{2.3}$$

$$Accuracy(AC) = \frac{tp+tn}{tp+tn+fp+fn} \tag{2.4}$$

Where *tp* is the number of true-positive documents (documents retrieved and should be retrieved), *fp* is the number of false-positive documents (documents retrieved and should not be retrieved), *fn* is the number of false-negative documents (documents not retrieved and should be retrieved), and *tn* is the number of true-negative documents (documents not retrieved and should not be retrieved).

However, in IR models, precision, recall, and F-score are better measures than the accuracy, and the reason is that $AC \propto tn$, which means that the larger the corpus, the more the number of true-negative documents for a certain query, which increases the *AC* score, and since *tn* is much larger than *tp*, the accuracy measure will approach 1 and be the same regardless the model has retrieved *tp* or not.

2.2 Vector Space Models

One of the most used document representation models is Vector Space Models (VSM). VSM uses vectors in n-dimensional space to represent the documents, where n is the number of unique terms extracted from documents and stored as a vocabulary.

The basic simple representation of a document in VSM is:

$D_i = (d_{i1}, d_{i2}, d_{i3}, ..., dij)$

Where i is the document and j is the term.

The aim of the VSM model is to minimize the function which calculates the sum of distances between document vectors (Salton et al. 1975). The function is given by:

$$F = \sum_{i=1}^{n} \sum_{j=1}^{n} s(D_i, D_j) \quad (2.5)$$

Where s (D_i, Dj) is the similarity between two vectors.

Moreover, terms might differ in their weights depending on the schemes used. One of the most used schemes for weighting the terms is Term Frequency- Inverse Document Frequency (TF-IDF), which weights the terms depending on their frequency, and frequency of documents it's mentioned in. However, this model suffers from a lack of semantic representation, in which each term is considered independently from other terms (Singh et al. 2017).

VSM can be extended into several representations such as Generalized Vector Space Model, Latent Semantic Analysis (Latent Semantic Indexing), and Random Indexing (Karnalim 2015). Generalized VSM solves the pairwise orthogonality problem assumed in VSM, which implies that vectors corresponding to terms are pairwise orthogonal. GVSM uses term-to-term correlations, which solves the orthogonality assumption. It represents each term vector as a combination of vectors in a 2^n dimensional space (Wong et al. 1985). Thus, the similarity of the new vectors can be calculated using:

$$sim(d_k, q) = \frac{\sum_{j=1}^{n} \sum_{i=1}^{n} w_{i,k} \times w_{j,k} \times t_i \times t_j}{\sqrt{\sum_{i=1}^{n} w_{i,k}^2} \times \sqrt{\sum_{i=1}^{n} w_{i,q}^2}} \quad (2.6)$$

Where t_i and t_j are the vectors for the terms i and j in the new 2^n dimension, $w_{i,k}$,$w_{j,k}$ are the weights of the new vectors t_i and t_j respectively, and n is the number of dimensions of the new space (Tsatsaronis & Panagiotopoulou 2009)

Latent Semantic Analysis is the first technique that simulates human reasoning of word meanings within a context (Evangelopoulos 2013). It comprises a large data corpus, high computational power, and sophisticated algorithms. LSA divides the document into passages, then assumes that the meaning of the passage is the linear sum of word meanings. The basic computation of LSA can be expressed as:

$$m(passage_i) \approx m(word_{i1}), m(word_{i2}), m(word_{i3}) \ldots, m(word_{in}) \qquad (2.7)$$

The steps of LSA are (1) collecting a large corpus from different areas of study, (2) representing the corpus as term co-occurrence matrix, (3) frequency transformation of the matrix (TF-IDF and normalize the TF-IDF frequencies), (4) Decompose the matrix to n dimensions, (5) compute the cosine similarity, and (6) calculate the semantic relatedness scores. The decomposition phase needs to take into account the semantics of each word, which implies that LSA should not only rely on comparing the two vectors based on the word's existence literally. Thus, LSA uses SVD (Singular Value decomposition) to achieve semantic reasoning. After the SVD step, each word or passage is represented as a vector in the semantic space. Then, cosine similarity can be calculated between the vectors.

Random indexing is a dimension reduction technique that solves the high dimensionality problem in VSM. Random Indexing consists of two phases (1) each word or document is assigned to a unique randomly generated index vector. These indexing vectors suffer from sparsity, high-dimensionality, and ternary (have values of -1,0,1), and (2) creating the context vectors by scanning the document and adding the indexing vector

to the context vector each time a word occurs. In other words, each word is represented by a set of index vectors that are the sum of the word's contexts

(Sahlgren 2005)

2.3 Word Embedding

Other models that include semantic meanings in their representation are Parallelogram models. These models assume that words are represented as points in Euclidean space, and there is a relationship between words given by the difference vector. It assumes that if the difference between two pairs of words (w_1, w_2) is equal to the difference between another two pairs of words (w_3, w_4), then the two pairs are relationally similar (Rumelhart & Abrahamson 1973).

Parallelogram models sat the ground for more sophisticated models for text reasoning. In the past 10 years, these models were embodied in machine learning algorithms producing the two popular algorithms for word embeddings Word2Vec and GloVe

(Peterson et al. 2020), and the context-based word embedding algorithm BERT (Devlin et al. 2019).

2.3.1 Word2Vec

According to (Reshma et al. 2020), Word2Vec representation for document and query in Vector Space Model (VSM) performed better when compared to Doc2Vec and TF-IDF representations. Moreover, the Cosine similarity measure performed better than the Jaccard similarity in the mentioned domain.

Word2Vec is a word-to-word deep learning embedding model presented in 2013 by Google. It converts terms into a distribution that is converted to a vector representation

of n-dimensions. According to (Wu & Wang 2017), this model reduces the dimensions used for representations and comprises semantic meaning.

Word2Vec model learns the embeddings in one of two methods (Peterson et al. 2020):

1. CBOW (Continuous Bag of Words): Taking a window of terms for each term (before and after), learning the embeddings for each term, then predicting the targeted term. CBOW aims to maximize the average log probability given by:

$$\frac{1}{T}\sum_{t=1}^{T} \log p(w_t|w_{t-c} \dots w_{t-1}, w_{t+1}, w_{t+c}) \quad (2.8)$$

Where c is the size of the window. CBOW uses SoftMax function as a single hidden layer to map input vectors to output.

2. Skip-Gram Method: Taking the term and a corresponding term (either before or after), then predicting other surrounding terms. This method aims to maximize the probability of surrounding term:

$$\frac{1}{T}\sum_{t=1}^{T} \sum_{-c\leq j\leq cj\neq 0} \log p\left(w_{t+j}|w_t\right) \quad (2.9)$$

Like CBOW, Skip-Gram also uses a single hidden layer for input-output mapping, but instead of SoftMax which is used for multiple categories, it uses the Sigmoid function which maps two output categories.

Although word2vec is a useful method, it is not sufficient for long document embedding.

2.3.2 GLOVE

The other popular model is GloVe, which stands for Global Vector, is an unsupervised model that was developed by a group of researchers from Stanford University in 2014. Unlike W2V which considers only the local property of the context, GloVe considers the global property of the whole dataset in addition to the local

properties of the words in the dataset (Pennington et al. 2014). The GloVe uses word-word co-occurrence matrix, which calculates the occurrence of each term in the context of another term. Simple taking the conditional probability for each term given the other term:

$$P(j) = \frac{X_{ij}}{X_i} \tag{2.10}$$

where X_{ij} is the number of times term j occurred in the context of term i, and $X_i = \sum_k x_{ik}$ is the sum of occurrence of any term in the context of term i.

However, GloVe aims to find an embedding vector that minimizes the cost function:

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2 \tag{2.11}$$

Where w_i^T is the transposed vector of term i, $\tilde{w}_j$ is the vector of the context term j, b_i and b_j are bias terms, and $f\left(X_{ij}\right)$ is an arbitrary function used for optimizing the loss function.

GloVe provides better performance than W2V since it considers both global and local properties of terms and documents.

2.3.3 BERT

In 2019, a group of researchers at Google implemented a state-of-the-art unsupervised language representation model named BERT, which refers to Bidirectional Encoder Representations from Transformation. This model is a pre-trained deep bidirectional encoder representation on unlabeled text.

The word bidirectional refers to the fact that the model joins left to right and right to the left context at all layers. Other encoders like LSTM Network, take words sequentially and generate their embeddings sequentially, which requires time and computational resources to train the network. It also uses only one direction (left context).

This means the model only looks at the words before the input word as a context and ignores the context after it.

On the other hand, Bidirectional LSTM Networks takes both directions, but it treats each direction separately, and just concatenates them, which result in losses in the true context of the word.

However, BERT has overcome these issues by using the transformer architecture, which can process multiple words simultaneously, which makes it faster than its predecessors.

Moreover, it has a large deep neural network that can manage both context directions at the same time, resulting in merging the full context of the word in the sentence. The output of the transformer is embeddings that represent the meaning in a numerical format (vector format).

The advantages of BERT are that, unlike other encoders, it understands the language, its grammar, and the context (Devlin et al. 2019).

Using neural embeddings in information retrieval tasks has been studied over the years, and they have shown significant improvement in performance among other representations.

However, BERT showed better performance among traditional representations, and other neural embeddings (Dai et al. 2019, Devlin et al. 2019, Cámara & Hauff 2020).

2.3.4 Other Models

(Peterson et al. 2020) merged the mentioned embedding algorithms (Word2Vec and GLOVE) with parallelogram models, which concluded in their confirmation of the performance of such algorithms to predict human analogy completion, in addition to relational similarity predictions. However, their proposed model is limited to the number

of relations it can predict. Moreover, this algorithm might differ in its performance per domain, so this might limit the algorithm for specific domains.

(Shahmirzadi et al. 2019) Compared several document representations models (TFIDF, D2V, LSI) based on Cosine similarity. The research concluded that TFIDF is more sensible to be used for non-condensed documents. On the other hand, LSI and D2V provide better representations but have a higher cost. The authors recommend testing on other similarity measures.

(Paskaleva & Bochev 2012) compared the performance of different similarity functions (Jaccard, Dice, Normalized Weighted Intersection) to cosine similarity measures under the TF-IDF representation model. Other similarity measures show slightly better performance than Cosine measures. The authors recommend testing other similarity measures which might open the potential for better performance. (Eminagaoglu 2020) proposed a new similarity measure that focuses on the dissimilarity between attributes that represents documents and queries in a vector space representation model. The measure calculates the dissimilarity when corresponding attributes are equal to 1, or when they’re different, and returns zero if the attributes are zero. This novel measure comprises better performance when compared to other well-known similarity measures. it also performed better for classification algorithms like KNN and Rocchio.

2.4 Local Outlier Factor (LOF)

Finding anomalies is widely used in many domains such as money laundering, detecting fraud, finding abnormal activities, and much more. LOF is a density-based anomaly detection algorithm that calculates the degree of outlier-ness of a certain data point concerning its neighboring points (Breuniq et al. 2000).

The algorithm defines the local density using k neighbors. the number of neighbors defines the area of the local density, which is the maximum reachable distance by the point's neighbors, then each point is compared to this density to determine whether a point is an outlier or not.

To illustrate how LOF works, let K be the number of neighbors of a certain point, and let RD be the reachability distance, and it can be calculated by:

$$RD(A,B) = MaxK - distance(B), d(A,B) \tag{2.12}$$

where K-distance of a point B is the distance between point B and its k nearest neighbor, and d (A, B) is the distance between points A and B. Figure 2.2 shows how RD is calculated.

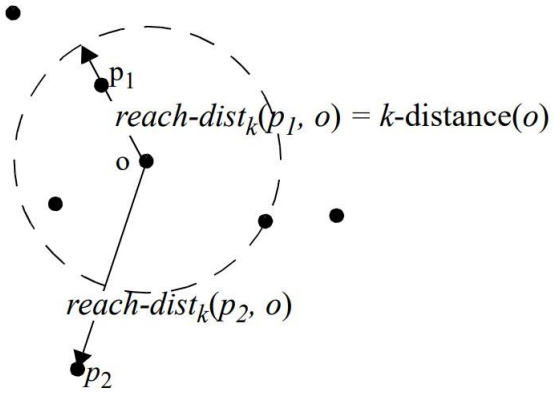


Figure 2.2: Reachability distance for K=4

From calculating RD's for all data points, the local reachability density (LRD) is calculated by taking the inverse average of RD's of a point's neighbors. LRD can be defined as:

$$LRD(A) = \frac{1}{\sum_{B \in N_K(A)} RD(A,B)/N_K(A)} \tag{2.13}$$

Where $N_K(A)$ is the number of K neighbors for point A.

Finally, LO of point A can be calculated by taking the average of LRD of the neighbors of A divided by the LRD of A. The mathematical expression can be

defined as:

$$LOF(A) = \frac{\sum_{B \in N_K(A)} LRD(B)}{N_K(A) \times LRD(A)} \quad (2.14)$$

The LOF score can be interpreted as:

- LOF ~ 1: Within density of neighbors

- LOF > 1: Outlier

- LOF < 1: Inlier

LOF can be applied on various applications such as intrusion detection systems (Lazarevic et al. 2003), and for outlier detection of different data types (spatial, video streams, etc.). (Schubert et al. 2014) Although the LOF approach provides a degree of outlier-ness, and can better detect local outliers, which might not be considered outliers in the global approaches Breuniq et al. (2000), the score is a ratio, which is hard to interpret, and there is no threshold for the values to be considered outliers. Moreover, the scores are relative, which means that a score of 1.1 might not be considered an outlier for some cases, but is an outlier in others. In other words, the threshold differs with data and problem type, but for granted, the higher the score, the more the outlier-ness of a point.

Through the literature, density-based algorithms were used mainly in anomaly detection but never used in information retrieval systems as a similarity measure. Some researchers used density algorithms such as DBSCAN for clustering similar images which are represented as vectors and using Euclidean distance as similarity index, which is then used for content-based image retrieval system (Khalid et al.

2020).

Chapter 3: Formalization Of The Embedding Based Vector Space Model

According to the literature described in the previous section, the vector space model is one of the best representation models for IR tasks. On the other hand, the neural embedding of documents and terms started to take huge roles in IR tasks. in the last ten years, researchers worked heavily on using neural networks for the neural embedding of document terms. It simply represents the document using the distribution of the information contained (Mitra & Craswell 2018).

Neural embedding can be treated as vectors, where each value in the vector can be represented as a dimension in the latent space model. In other words, we can simply match the query with the documents using neural embedding representation and find how closely these vectors are either using Euclidean, Cosine, ...etc. (Zhang et al. 2016).

The conventional vector space model represents the documents as a vector in space, where several representations can be used; The document can be represented by constructing a bag of words (BOW): $BOW = t_1, t_2, t_3, \ldots, tn$ where $t_1, t_2, t_3, \ldots, t_n$ are the tokens extracted from all documents in the dataset, then each document can be represented by the existence of token t_i in the document, resulting in a vector of 1's and 0's. Suppose there are two documents:

D_1: "The sun rises everyday"

D_2: "I see this dog everyday"

Then, $\mathcal{L} = \{dog, everyday, i, rises, see, sun, the, this\}$

and the vectors of D_1 and D_2 are:

$D1 = \{0,1,0,1,0,1,1,0\}$

$D2 = \{1,1,1,0,1,0,0,1\}$

Also, instead of representing the tokens by their existence, they can be represented in their importance to the document using TFIDF, replacing the ones with fractional values representing the importance of a token to the document and across all documents. Thus, the vector space model can be constructed by setting a token to each dimension. Figure 3.1 shows how documents can be represented in space model.

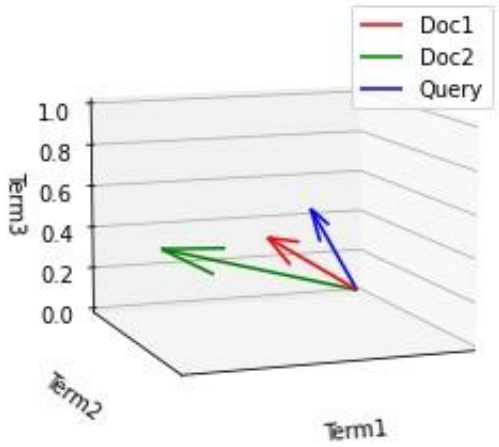


Figure 3.1: Illustration of conventional VSM

However, this method produces two main problems; Sparsity, and high-dimensionality. Although this model represents each term as a separate dimension, which reserves the local information of each term, the larger the number of documents the larger might be the constructed lexicon L, thus higher will be the dimensions needed to represent the document. Moreover, the larger the BOW, the more probable the document representation to have a lot of zero values in the vector due to the absence of the corresponding terms, resulting in a vector full of zero values with few ones. This phenomenon is called sparsity, and it increases the complexity of the space model and reduces the efficiency of the Euclidean distance since all distances would almost be in the same range of values, which makes it hard to distinguish between vectors. Another problem in this representation is that it misses the context where terms were mentioned.

As mentioned previously, a term could have multiple meanings according to its context, and this representation misses the context and the order of the terms in the documents.

For the aforementioned reasons, in this thesis, we present the Embedding-Based Vector Space Model (EVSM) representation which takes advantage of a dense vectorial representation of the terms.

Instead of statistically representing directly the documents as vectors. The document passes through a neural network that encapsulates as much information as possible and represents the document in a relatively low-dimensional space. The output of the NN can't be interpreted as the conventional VSM. The values of the resulting vector do not correspond to terms, it is an embedding or encapsulation of different parameters that represents the document. It might encapsulate the term's locations, frequencies, context, ...etc.

One of the neural embedding-based representations is BERT. It has pre-trained deep neural networks that are used for language modeling on a combination of masked language modeling (MLM) and next sentence prediction (NSP) using a large corpus from the Toronto Book Corpus and Wikipedia (Devlin et al. 2019).

The application of the embedding used the predefined transformers made by the Hugging Face Team. The team has built multiple DNN's that differ in corpus size, number of hidden layers, and number of attention heads. Moreover, it has its predefined tokenizer that is compatible with the BERT transformer.

BERT embedding has outperformed other embedding models in encapsulating the context of documents, and therefore better document representation (Devlin et al. 2019).

To formalize the concept, suppose that $T = \{t_1, t_2, t_3, \ldots, t_i\}$ are the terms in document D_1, and $W = \{w_1, w_2, w_3, \ldots, w_j\}$ are the terms in the BOW where $T \in W$. Therefore, the representation of D_1 in the conventional VSM:

$D_1 = 1$ if w_j in T,0 otherwise. On the other hand, D_1 is represented in EVSM:

$$D_1 = \{e_1, e_2, e_3, \ldots e_n\}$$

Where $\{e_1, e_2, e_3, \ldots e_n\}$ are the output values from the NN.

This representation is a global representation of the document in lower-dimensional space. It encapsulates the general context of the document. However, this representation suffers from the absence of local information. In other words, the local context of each term, and the context of each sentence are not considered in this representation, which might provide misleading results.

To encapsulate the local information, neural embedding can be applied to each term, where each term has its representation that preserves the local context of a term by taking the neighboring terms in the embedding process. This method increases the information represented by the embedding without increasing the dimensions of the space model and leads to better language modeling, which would positively affect the retrieved results. However, it has a complex drawback; Instead of representing the document in one vector, it is now represented by *n* vectors, where *n* is the number of terms in the document.

To illustrate how a document is represented as a cloud of points (vectors), suppose the following table represents the vector dimensions for terms in documents and query in 2D:

Table 3.1: Sample of points to represent cloud of vectors in EVSM

(a) D1

Point	X	Y
$D1_1$	15	19
$D1_2$	16	19
$D1_3$	15	17
$D1_4$	11	14
$D1_5$	17	10
$D1_6$	16	14
$D1_7$	19	11

(b) $D2$

Point	X	Y
$D2_1$	3	7
$D2_2$	4	16
$D2_3$	1	17
$D2_4$	1	20
$D2_5$	3	11
$D2_6$	18	8
$D2_7$	18	7
$D2_8$	20	9
$D2_9$	20	4
$D2_10$	15	3
$D2_11$	18	7
$D2_12$	15	12

(c) Q

Point	X	Y
$Q1_1$	1	16
$Q2_2$	1	11
$Q3_3$	8	16
$Q4_4$	6	9
$Q5_5$	1	10
$Q6_6$	2	17

Table 3.1 is illustrated in Figure 3.2. It can be seen how the terms of D1 are close to each other and far from query points, while D2 has terms that are close to the query. In other words, D2 is much related to Q1 since its density is close.

To get a more concrete example, 2 sample documents and a query from a sample

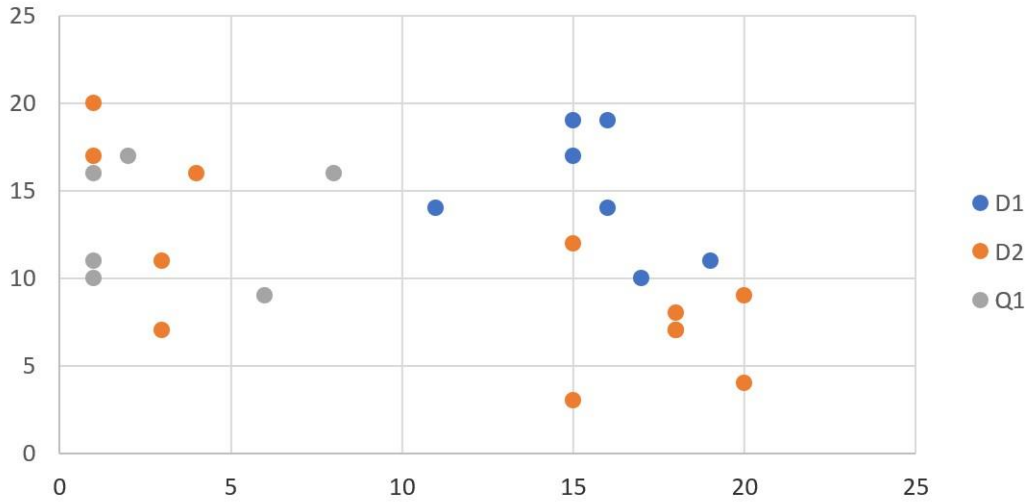


Figure 3.2: Illustration of cloud of points in 2D

dataset is embedded using BERT model. to illustrate the example, the embeddings should be in 2D to be able to visualize it since BERT embedding takes hundreds of dimensions (512, 768, 1024 ...etc.). Thus, T-SNE algorithm is used, which is used to embed high

dimensional data in lower dimension according to a given number of components (van der Maaten & Hinton 2008).

T-SNE is a T-distribution Stochastic Neighbor Embedding that converts high dimensional Euclidean distances into conditional probabilities and aims to aims to find a low-dimensional data representation that minimizes the cost function, and the cost function is reduced using stochastic gradient descent. This algorithm has a simple cost function that can be optimized easily, but its computational and memory complexity is quadratic to the number of data points.

Figure 3.3 represents embedding 2 documents (one is relevant and the other is irrelevant), and a query that is related to the document.

From the illustration, it can be seen how query points are surrounded by the relevant document points, and the latter is surrounded by the irrelevant documents. In other words, the relevant document points are closer to the query points.

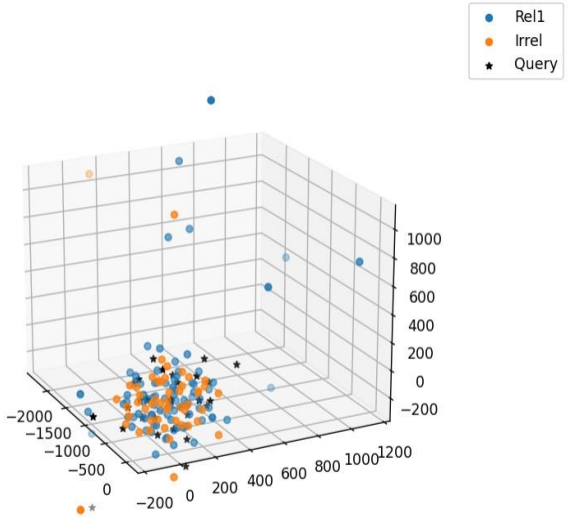
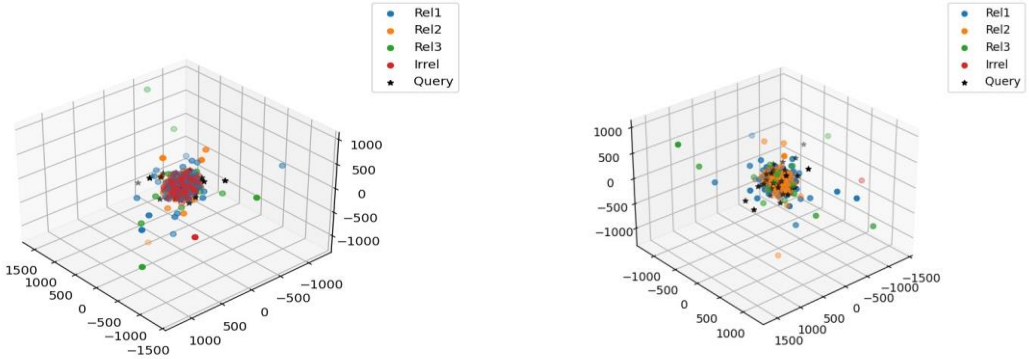


Figure 3.3: Example1: 3D illustration of an embedded cloud of points in EVSM

Figure 3.4 shows another example, where more documents are illustrated.



(a) View1 (b) View2

Figure 3.4: Example2: 3D illustration of embedded cloud of points in EVSM

From the Figures 3.4a and 3.4b, it can be seen how query points are closer to the relevant documents, while it is far from irrelevant document. This method proofs that documents are represented with both global and local information, where the points related to the main context of the document are close together, and some points that are far from the context are away from the denser area.

This leads to the next chapter, where the density concept is developed for finding the similarity between documents and queries.

Chapter 4: Density-Based Approach To Document Matching Problem

As illustrated in the previous section, the documents and query are represented as a cloud of points, where there are areas in the EVSM that are denser than others, which represents the context of the document, and other scattered areas, where these points are far from the context.

To find the similarity between a document and a query, the context should be the same. In EVSM, the dense area of both document and query should be overlapped. To measure how much this overlap is, we need to represent this overlap in a numerical measure. The concept of defining a dense area is simple; the area can be defined by a circle of radius k, and the point p is considered within this area if $d(p, o) <= k$, where $d(p, o)$ is the distance between point p and the origin of the context. Thus, we need to find the origin of the context. It can be calculated as:

$$o_j = \frac{\sum_{i=0}^{n} p_{n,j}}{n} \tag{4.1}$$

where o_j and $p_{n,j}$ are the the value in the dimension j for the center of the area and point p_n respectively, and n is the number of points in the EVSM.Thus, o_j is the arithmetic mean of points in EVSM in the j^{th} dimension.

Although this method is easy to calculate, it is highly affected by extreme points (points outside the dense area) which causes shifts in the center of the area.

A solution would be dealing with these extreme points as outliers (anomalies). If this point is far from its neighbors, then it can be considered as an outlier, and then the center of the inlier points can be an accurate measure of the density.

Another way is to use density-based anomaly detection algorithms like LOF. This algorithm generates a score of outlier-ness degree for each point, which can be interpreted in a better way than considering only the center of the document. In other words, the

outlier-ness degree can be interpreted as how much document and query are overlapped. Moreover, it can provide a rank to the document that can be compared with other document scores.

LOF defines k neighbors, and the local density of a point is defined by the distance between this point and its k nearest neighbor. Then, from the local densities, LOF score can be calculated for each point, which defines the outlier-ness of each point from the density of the points. The more the score, the more the outlier-ness of point.

To embrace this concept in document-query matching, we can check how much the outlier-ness of query points are to the document density, which can be then interpreted as the similarity/difference between the document and the query.

To formalize it, let's assume that:

$Q = \{q_1, q_2, q_3, \dots q_i\}$ are the query points

$D = \{d_1, d_2, d_3, \dots d_j\}$ are the document points

Then, there are i EVSM's:

$E_D = \{e_{1,D}, e_{2,D}, e_{3,D}, \dots e_{i,D}\}$ where

$e_{i,D} = \{q_i, d_1, d_2, d_3, \dots d_j\}$

$e_{i,D}$ is an EVSM that contains all the document D points and one query point. The objective is to find the LOF score of q_i in its corresponding e_i. The LOF calculations were described in section 2.4.

After calculating LOF for each query point for document D, the total outlier- ness of the query to the document can be defined by taking the mean of LOF scores. Thus, the rank R of query Q for a document D:

$$R_{Q,D} = \frac{\sum_{n=0}^{i} LOF_{q_n, e_{n,D}}}{i} \tag{4.2}$$

Where i is the number of query points.

However, the LOF has a hyperparameter that should be predefined, which is the number of nearest neighbors k to a point. This rank can be used to re-order the documents based on their outlier-ness with the query, and the first documents in the order are the most relevant documents to the query.

This is the concept to be explored in this research. The model would represent the documents and queries as a cloud of points in the EVSM model. Then, a Density-based algorithm inspired by the LOF score is proposed for query matching and document ranking.

The model would be tested on several datasets and compared with the other benchmark model. The next section will describe in detail the experimental environment, the model development, optimizations, and final testing.

4.1 The Proposed Model

This research will explore a novel algorithm that is inspired by density-based algorithm for matching documents represented in the EVSM. The novel algorithm represents documents using neural embedding techniques and matches the embedding using a new matching method inspired by LOF.

For the representation, one of the proposed models by [4] "Bert-Base-Uncased" is used, which is a pre-trained large deep neural network that contains 12-layer, with 768-hidden, 12 attention heads, 110M parameters, and trained on lower-cased English text. The result is a vector for each token with 768 values (dimensions).

Note that this model is one of a large list of pre-trained models developed by the Hugging Face Team and other contributors, which covers almost all languages with different accents for each language, in addition to the availability to train a new network.

However, this research will not shed the light on embedding techniques and will use the standard predefined model.

The use of these pre-trained networks allows the proposed model to be applied to different languages without the need to train the model. Moreover, it allows fine-tuning the networks for improving the results on a specific subject or context.

The transformer generates embedding (vector) for each token extracted by the BERT tokenizer. Each vector embeds the token, Left-to-right (LTR) context, and right-to-left (RTL) context, resulting in a full representation of the context of each token. In other words, the same word might have different representations depending on its context. This helps the model to better understand the language and recognize semantic meanings of the same word. For example, the word barks in “paper are made from tree barks” refers to a part of the tree, while “The dog barks so loud” refers to the noise made by the dog.

However, there are two ways to handle the embedding; either by taking the mean of these embeddings and representing the document as one vector, or dealing with each token as a vector in the space, resulting in a cloud of vectors that represent the document.

Although representing the document by a cloud of vectors is much complex than the mean of embedding method, it encapsulates more information about the document and its context, resulting in more accurate context-based matching. In this research, both document and query will be represented as a cloud of vectors in the vector space model.

However, such representation doesn’t allow to use of traditional matching techniques such as Euclidean distance or cosine distance between all tokens because it will need high computational power, in which the distance would be calculated for each point with all other points in the space. Moreover, the approach will suffer from the curse of dimensionality; in vector models, especially in Euclidean distance, when distances tend

to be the same as dimensions grow. This results in much less representation of the distances between points in the space. Although the embedding technique reduces the dimensions, according to the embedding discussed previously, the space would be 768 dimensions, which is considered a high dimensional space.

To overcome this issue, we introduce the density-based approach. The approach is inspired by the LOF anomaly detection technique; the algorithm takes each query point and all the document points, and assumes that the space only contains $1 + d$ points, where d is the document length (number of document tokens). After that, the algorithm performs the LOF technique to find the LOF score for each query point, then takes the arithmetic mean of the LOF scores for the query point, resulting in a final score for each document for a certain query. This score is used as ranking for the document, where the lower the score, the more the document is related to the query.

This algorithm uses the concept of LOF to find how much the density of the query is overlapped (considered inlier) with the document density. In other words, the less the LOF score for each query point, the more the point to be considered an inlier, the more the query to be considered an inlier with the document, and the more the document is related.

The algorithm follows these steps:

1. Take one query point and all document points, and assume the space consist of only these points:

The reason why it relates to query points is to optimize the number of iterations needed since it iterates over each token on all documents, and since queries tend to have much fewer tokens than the document, it's more logic to iterate over them.

2. Calculate the distance matrix using the Euclidean distance:

This method is to optimize the calculation performance. Instead of calculating the distance between every two points on each iteration, the distance matrix calculates it at once, and the distances are stored offline.

3. Find the k^{th} neighbor for each point.
4. Calculate the reachability distances:

By iterating over the space points and comparing the distance of a point from its neighbor with the k^{th} neighbor distance and take the maximum.

5. Calculate the local reachability density.
6. Calculate the Local Outlier Factor LOF score for each query point.
7. Re-rank documents according to LOF scores (Ascending).

The following represents the Pseudo Code for the proposed algorithm in its basic form. This code is to be optimized through the testing experiments in the following sections.

Algorithm 1 Proposed Algorithm Pseudo Code

1: **for** Every Query point q_i **do**

2: **for** Every Document D_j in database **do**

3: $D_j = \{d_1, d_2 ... d_n\}$

4: $space = \{q_i, d_1, d_2 ... d_n\}$

5: Calculate distance matrix *DM*

6: **for** Each point in space **do**

7: Find the K^{th} nearest neighbor

8: Calculate RD $don\ (A, B) = MaxK - distance(B), d\ (A, B)$

9: Calculate LRD $LRD(A) = \frac{1}{\sum_{B \in N_K(A)} RD(A,B)/N_K(A)}$

10: Calculate LOF $LOF(A) = \frac{\sum_{B \in N_K(A)} LRD(B)}{N_K(A) \times LRD(A)}$

11: **end for**

12: **end for**

13: **end for**

Chapter 5: Experimental Environment

In the previous chapter, we proposed a novel density-based matching algorithm that is inspired by the Local Outlier Factor for matching and ranking documents. In this chapter, we will discuss the experiments made to evaluate the performance of the model and compare it with other baseline models published.

5.1 The Model Parameters

Since the model is inspired by LOF, the model has the advantage of having only one hyper-parameter to be used for optimization, which is the K^{th} nearest neighbor. K defines the number of nearest points that are assumed to be relevant to the point, or in other words, considered inliers with the point. Referring to Figure 2.2, $K = 4$ means that only the third closest neighbor is considered relevant to a point, and any point within the distance between point *o* and neighbor *k* is considered an inlier point.

In our model, at each iteration, the EVSM contains all document points and only one query point, so $1 < k < m+1$ where m is the number of document points.

To test the effect of changing K on the performance of the model, the model is tested over several values of K, and results are compared.

5.2 The Datasets

The model was tested over different datasets that are known for IR tasks.

We used three different publicly-available datasets that contain the following:

1. The document set: A set of documents collected from a known source.
2. The query set: A set of queries that aims to find information in the document set.

3. The Qrel set: A relevance assessment that defines the set of relevant documents to each query, knowing that this Qrel is set by a group of researchers manually for different IR tasks.

5.2.1 LISA Corpus

LISA stands for Library and Information Science Abstracts. The dataset is developed based on the 1982 Library and Information Science Abstracts database by Peter Willett of Sheffield University. The set contains around 6000 documents that have only the title and the abstract of each document. Also, the dataset contains 35 natural-language queries that are defined by the researchers, and the Qrel is set manually by them.

The dataset is extracted from the test collection stored at the Glasgow University Information retrieval group website. The corpus can be accessed through this link: http://ir.dcs.gla.ac.uk/resources/test_collections/lisa/. The dataset is formatted like the following:

1. The Documents: separated in files of 1000 documents delimited by a line of stars ”****”. The document format starts with an identifier of the document, the title, and the abstract. An example:

Document 1

”The Indian council of library and information services research and training:

A proposal for consideration.

Critically examines the Indian library scene. The management, operation, and services of most libraries are inefficient and ineffective, resulting in grave misallocation and underutilization of library resources. Little major research has been carried out. University librarianship courses are not much help for managing and operating libraries efficiently. The major national institutions have serious limitations. To improve the

situation proposes the establishment of an Indian council of library and information services research and training. The council would be set up by the government and have well-defined roles and functions covering all aspects of Indian librarianship."

2. Queries: The query format starts with an identifier, the query text, and a delimiter "". An example:

" I am interested in the identification and evaluation of novel Computer architectures, for instance, increased parallelism, both In SMID and MIMD machines. I am also interested in information about Associative stores or memories and associative processors. Computer architectures, associative processors, associative stores Associative memory."

3. The Qrel: The query relevance-assessment which starts with a query identifier, number of relevant documents, the relevant document identifiers, and a delimiter "-1". An example:

"Query 1

2 Relevant Refs:

3392 3396 -1"

5.2.2 Cranfield Collection

The collection contains comes in two forms: 1400, and 200 documents. The first one is used. The collection contains scientific research papers. The topics are mainly in different types of engineering and science. Like the LISA corpus, the collection contains a collection of 1400 documents, a set of 224 queries, and their Qrel.

The dataset is extracted from the test collection stored at the Glasgow University Information retrieval group website. The collection can be accessed through this link:

http://ir.dcs.gla.ac.uk/resources/test_collections/cran/. The dataset is formatted like the following:

1. The Documents: The documents are listed in one file. The document format starts with an identifier of the document ".I", the title (with ".T" identifier), the Authors (with ".A" identifier), the book or publishing info (with ".B" identifier) and the abstract (with ".W" identifier). An example:

".I 1

.T experimental investigation of the aerodynamics of a wing in a slipstream .

.A brenckman,m.

.B

j. ae. scs. 25, 1958, 324. .W

experimental investigation of the aerodynamics of a wing in a slipstream . an experimental study of a wing in a propeller slipstream was made in order to determine the spanwise distribution of the lift increase due to slipstream at different angles of attack of the wing and at different free stream to slipstream velocity ratios. the results were intended in part as an evaluation basis for different theoretical treatments of this problem. the comparative span loading curves, together with supporting evidence, showed that a substantial part of the lift increment produced by the slipstream was due to a /destalling/ or boundary-layer-control effect . the integrated remaining lift increment, after subtracting this destalling lift, was found to agree well with a potential flow theory . an empirical evaluation of the destalling effects was made for the specific configuration of the experiment ."

2. The Queries: The query format starts with an identifier ".I", and the query text (with an identifier ".W". An example:

”.I 001

.W

what similarity laws must be obeyed when constructing aeroelastic models of heated high speed aircraft .”

3. The Qrel: The query relevance assessment where each line starts with a query identifier, the relevant document, and the degree of relevancy of the document.

The query identifier is repeated over each relevant document. An example:

”4 236 3

4 166 3

4 488 -1”

5.2.3 AILA 2019

AILA stands for Artificial Intelligence For Legal Assistance. The corpus contains Statutes, which are the established laws that are applied to this case and led to filing it, and Precedents, which are the previous cases and how the court dealt with it. The dataset is developed for two main tasks: Task1- Identifying relevant prior cases for a given situation, and Task2- Identifying most relevant statutes for a given situation.

The set contains around 2914 documents that describe the case, the parties, the date of the case, and other info. This set is for Task1. Moreover, the set contains another 197 statutes that contain titles and descriptions of the statute. Also, the dataset contains 50 long queries that describe certain cases, and tend to find the relevant cases or statutes.

The dataset is developed by the Fire AILA team. The collection can be accessed through this link:

https://sites.google.com/view/fire-2019-aila/dataset-evaluation-plan.

The dataset is formatted like the following:

1. The Documents: Each document is stored in a separated .txt file. The document contains the case rivals, date, and other information about the case.

2. Queries: The query format starts with an identifier ($AILA_Q1||$), and the query text, which is a long text describing a case.

3. The Qrel: The query relevance-assessment in the format of <query-id> Q0 <document-id> <relevance>, where relevance can be either 1 (if the document is relevant) or 0 (if it is not relevant).

5.3 The Evaluation Metrics

The mentioned collections were selected to test the performance of the proposed model on long and short queries and documents, and to different topics, where language structure changes.

To evaluate the model performance, Precision, Recall, and F-Score are used. The metrics are popular for IR tasks since we have a pre-defined relevance assessment for each dataset, which makes the task similar to a classification task. Moreover, the metrics test the performance from different perspectives: (1) testing how much relevant documents returned from overall results, (2) testing how much relevant documents returned from all relevant documents, (3) testing the overall performance.

Since the model is a ranking model that re-arranges all the documents according to the average of LOF scores for each document, the model will return the whole dataset for each query but in a different order. Thus, we will assume that the first 10 results are returned and test the performance according to them. The more results are considered, the higher the recall would be.

The reason behind considering the first 10 results is due to the relevance assessment, where most of the queries have 3-6 relevant documents, which makes considering the

first 10 feasible.

Finally, the model will be compared with another model that uses the same BERT embedding, but takes the mean of embeddings, ending in representing the document in one vector in the EVSM (Tanaka et al. 2020). This model will be referred to as “Mean of Embeddings” in the testing experiments.

5.4 Testing Experiments

Several parameters should be tested in the model to see the effect of changing its values on the overall performance. The parameters that should be considered are:

1. Changing the Distance Metric: Changing the distance used in the EVSM using Euclidean and Cosine. The reason for this scenario is to test if the model suffers from the curse of dimensionality since the dimensions are 768 which is relatively high.

2. Stop Words Removal: removing stop words is one of the essential scenarios that should be tested in NLP and IR tasks.

3. Weighting document terms: giving weights to document terms based on their importance using TFIDF.

4. Number of Nearest Neighbors *K*: changing the value of *K* and iterating over several ranges (1−10, 10−20, *maxK*). Note that *maxK* would be considering all document points as neighbors in each query point iteration.

5. Document normalization: normalizing the document size by multiplying the LOF score by $(1 + log(document\_size))$

6. Comparing To a Benchmark Model: Comparing the proposed model to the state-of-the-art Mean of Embedding Model.

7. Hybrid Solution: combining our model with another ranking model (BM25), in a way that BM25 selects the top *n* documents based on its algorithm (mainly TFIDF) and

then passes the selected documents to the proposed model, where its re-ranked based on the density.

8. Changing the Sampling Method: since one of the datasets (LISA) is relatively big, and due to computational limitations, the mentioned dataset is sampled in two ways: (1) for each query, take the relevant document, and other random 20 irrelevant documents (for example 7 relevant, 20 irrelevant), and (2) take the relevant documents, and randomly select the same number of irrelevant documents (for example 7 relevant, 7 irrelevant).

9. Testing Over Different Datasets: Testing the final optimized model using a set of testing experiments over different datasets and samples from the collections mentioned previously.

5.5 Testing Environment

The testing was made using a high-performance virtual machine provided by Palestinian telecommunication company “PALTEL”, which provided a virtual lab on their data center servers as a fund for the project. The VM contains 96 logical 1.9 GHz processors and 262 GB of RAM.

The model is developed using Python. The development of the model used the following libraries:

1. Pandas, NumPy, re, math: basic libraries used for data pre-processing and data preparation.

2. transformers: used for pre-trained BERT model that is developed by hugging face team. The used functions are BertTokenizer and BertModel.

3. concurrent.futures: used for parallel processing to increase the computation performance.

4. sklearn.feature extraction.text: used for stop words removal and TFIDF

5. rank bm25: for BM25 model development
6. TQDM: for monitoring progress
7. JSON: for managing importing and exporting data
8. scipy: for distance calculations and tokenization for BM25

The proposed algorithm has some mathematical operations that are made on every iteration. At first, the embedding; the algorithm needs to calculate the embedding of each token to use it later for distance calculations. Since the number of tokens is large, and the embedding results in vectors of 768 dimensions for each token, then it is recommended to be stored offline. The data structure used is dictionaries stored in JSON files, where each key is a document id that contains two lists: (1) the tokens, and (2) the corresponding vectors (embedding). Secondly, the distance matrix. The matrix is needed to find k^{th} nearest neighbor in the LOF score. The matrix is $M \times M$ where $M = d + q$ and d is the number of documents points, and q is the number of query points, knowing that the q contains the points of all queries. To enhance the run time of the algorithm, this matrix is built once, and pointers are used to take needed chunks for a certain iteration. Moreover, to decrease the processing time, the matrix is prepared and stored in ".npy" files on the disk as offline indexing files, and the needed matrices are loaded to memory and used. This method decreases the needed processing to achieve the same results, and reduces the needed RAM to process the data, resulting in a more efficient algorithm.

Finally, the results. Each run in the model is iterating over all queries and on a range of K nearest neighbors. The results are in dictionary structure and stored in JSON files. The stored data is then used for calculating precision, recall, and F-score for model evaluation.

Chapter 6: Results

In the previous chapter, 9 different experiments were discussed. In This chapter, the results of the experiments are discussed, and the optimization of the model is based on the tested experiments. The proposed algorithm is tested over the mentioned experiments in the previous chapter, and the experiments were applied to the datasets discussed previously. LISA dataset is used in 3 different ways; using the whole dataset, a small sample of 27 documents, and a sample of 1042 documents.

The following sections discuss the results for each dataset mentioned.

6.1 Changing The Distance Metric

The first thing to test is changing the distance metric. Since literature shows that using Cosine distance in VSM showed better performance, the proposed model is tested over both Euclidean and Cosine distances.

Table 6.1 shows the results from testing the algorithm on LISA dataset. The run on full dataset shows that using Euclidean distance is more feasible than the cosine. A reason why the performance is better using the Euclidean is the fact that the document points are very dense and close to each other, which will make the differences in cosine values almost similar, and will be hard for the algorithm to distinguish between inliers and outliers.

Table 6.1: Changing distance metric (LISA Dataset)

(a) *Euclidean*

	Precision	Recall	F-Score
Top 10	**0.208**	**0.235**	**0.22**
Top 5	0.09	0.09	0.09
Top 3	0	0	0

(b) Cosine

	Precision	Recall	F-Score
Top 10	0.157	0.176	0.165
Top 5	0.135	0.147	0.14
Top 3	0.112	0.123	0.117

From the results in Table 6.1, using Euclidean distance provided better results, and this is due to the concept of density-based algorithm, which doesn't use the vector itself, but assumes that the vector is represented as a point in space, which makes using Euclidean distance more feasible.

However, the experiments made using the Euclidean distance failed to achieve better performance than Cosine in the top 3, and top 5 results, which leads to needing further investigation to understand how the Euclidean distance can be employed to produce a better ranking. To recap, using Euclidean distance metric is more feasible due to the ease of calculating. The adopted distance metric does not influence the performance of the method consistently. Further investigation is required to test whether the algorithm can adopt other improvements and parameters.

6.2 Stop Words Removal

A reason for bad performance in the previous test is that the density of the documents is increased due to the existence of stop words, where each stop word is embedded differently according to its context. To solve this issue, the stop words were removed from the EVSM using the NLTK stop words corpus. The removal process was tested on LISA dataset, and it increased the F-score up to 28%.

Table 6.2: Stop words removal (LISA Dataset)

(a) *Euclidean*

	Precision	Recall	F-Score
Top 10	**0.262**	**0.288**	**0.273**
Top 5	0.257	0.257	0.257
Top 3	0.278	0.278	0.278

(b) Cosine

	Precision	Recall	F-Score
Top 10	0.241	0.259	0.249
Top 5	0.257	0.257	0.257
Top 3	0.278	0.278	0.278

Table 6.2 shows how removing stop words enhanced the performance for both distance metrics, and neutralized the effect of the distance metric. The result shows how

Euclidean performance is improved and met the Cosine performance for top 3 and 5, but outperformed at top 10. This step decreased the complexity of document representation and affected the density distribution of the documents in a way that focused more on the tokens that differentiate the document concepts, which makes both metrics feasible for use in the model since the differences between points distances is larger, with a slight advantage for Euclidean.

However, each of the mentioned tests took more than 168 hours of run time, and that is why we only took the top 3, 5, and 10 results. To overcome this issue, and do more thorough testing, a small sample is taken to focus more on the effect of parameters and develop a proof of concept. This helps develop the algorithm in a faster rhythm. The sample contains the relevant documents of query 4 (7 documents) from the LISA dataset, and another 20 random irrelevant documents from the same dataset, resulting in a subset of 27 documents and one query. This sample will be referred as" LISA Sample I".

The results of running the algorithm on the mentioned dataset with stop words removal is shown in Table 6.3. The results show better performance, and it is expected since working on smaller set means less complex space and less overlapped documents. In conclusion, using stop words removal positively affect the performance

Table 6.3: First run on LISA Sample I

	Precision	Recall	F-Score
Top 10	0.61	0.654	0.629
Top 5	0.713	0.713	0.713
Top 3	**0.795**	**0.795**	**0.795**

of the proposed model, and this optimization is to be added to the basic settings of the model.

6.3 Weighting Document Terms

The huge improvement in the performance is due to the decrease in the number of points in the space. In this context, the model can be optimized by giving more importance to the points that are most important to the context and moving the points that are less important away. In other words, the points are weighted according to their importance to the context.

To formalize the concept, a lexicon for the whole dataset should be constructed, and a TFIDF score should be calculated for each token in the lexicon. Then, two different tests were made to check the effect of applying the TFIDF score: (1) divide the distance between a query token and a document token by the TFIDF score, and (2) divide the LOF score of the mentioned pair of points by the TFIDF score. The results of the mentioned tests are shown in Table 6.4.

Table 6.4: Term Weighting (LISA Sample I)

(a) *Dist./TFIDF*

	Precision	Recall	F-Score
Top 10	0.144	0.144	0.144
Top 5	0.163	0.163	0.163
Top 3	0.163	0.177	0.169

(b) *LOF/TFIDF*

	Precision	Recall	F-Score
Top 10	**0.889**	**0.889**	**0.889**
Top 5	0.753	0.753	0.753
Top 3	0.57	0.598	0.582

From the results, it is clearly seen how TFIDF increased the performance when dividing the LOF score by the token weight, especially on the top 3 levels. In other words, it helped retrieve more relevant documents in the first three results. However, it affected the results negatively when applied to the distance level. One possible reason for this is that the density of the document is scattered due to the fact the contexts discussed in the dataset are in the same topic, which decreases the importance of some important words due to their frequency, resulting in moving them away from the context center in the

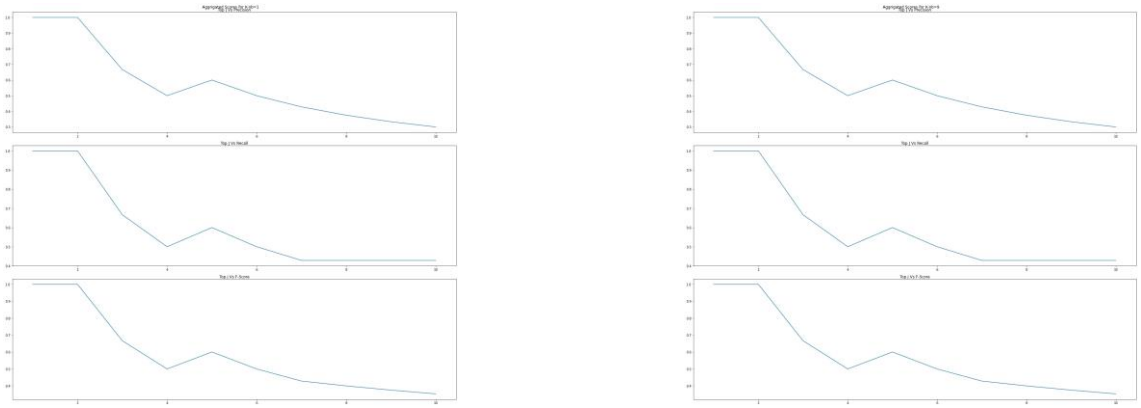
EVSM, which in turn increases the distance of the k^{th} neighbor and finally leaving documents in almost same LOF score.

This experiment results in the importance of applying the TFIDF term weighting to LOF calculations as one of the basic settings in the proposed model.

6.4 Number Of Nearest Neighbors *K*

The effect of changing the number of *K* nearest neighbors is one of the most important experiments to test since changing *K* affects the definition of inliers and outliers for the LOF calculations.

Changing *K* from 1 to 10 has no effect on the performance measures using this subset. Figures 6.1 a and b shows how precision, recall, and F-score changes over top *J* results for $K = 1$ and $K = 9$. as a result, changing *K* doesn't affect the performance of the algorithm. The figures were constructed as follows:



(a) $K = 1$ (b) $K = 9$

Figure 6.1: Changing Number of Nearest Neighbours K (LISA Sample I)

Take the first ranked result from the query results, calculate the precision, recall, and F-score on as if the algorithm returns only the first result, then repeat for top 2, 3, 4, …etc.

The performance on the top 2 results reaches 1, which means that the algorithm was able to accurately return the first 2 results correctly, then the performance degrades until the fifth result, which was relevant.

Note that the results obtained from this small subset (LISA Sample I) need to be generalized on a bigger scale. Thus, another sample of 1042 documents contains all relevant documents from each query and 20 random irrelevant documents. This sample will be referred as" LISA Sample II".

The results in Table 6.5 shows the performance of the algorithm on the new larger sample. The stop words removal and the term weighting using TFIDF were applied, and the performance is tested over $1 <= K <= 10$ and using $Max(K)$, where $Max(K)$ is the number of document points since the space contains all document points and one query point. The results show that using $max(K)$ provides better results than selecting different values for K.

Table 6.5: Changing Number of Nearest Neighbours K (LISA sample II)

(a) $1 <= K <= 10$

Top J	Precision	Recall	F-Score
1	0.463	0.463	0.463
2	0.383	0.409	0.391
3	0.36	0.407	0.374
4	0.334	0.404	0.353
5	0.317	0.407	0.338
6	0.291	0.39	0.312
7	0.271	0.385	0.294
8	0.264	0.397	0.29
9	0.246	0.388	0.273
10	0.24	0.391	0.267

(b) $Max(K)$

Top J	Precision	Recall	F-Score
1	**0.514**	**0.514**	**0.514**
2	**0.429**	**0.457**	**0.438**
3	**0.39**	**0.443**	**0.406**
4	**0.364**	**0.436**	**0.383**
5	**0.337**	**0.424**	**0.358**
6	**0.319**	**0.424**	**0.342**
7	**0.302**	**0.428**	**0.328**
8	**0.282**	**0.419**	**0.309**
9	**0.263**	**0.41**	**0.291**
10	**0.249**	**0.403**	**0.276**

As a result, using maximum value of K enhanced the performance significantly, and this improvement is to be added to the model's basic settings.

6.5 Document Normalization

The results shown in previous sections show that the algorithm is affected by the number of documents in space. A reason could be that the documents differ in their size (number of tokens), which leads the document to have multiple contexts, and can be seen

as a close result to the query and returned. To check this hypothesis, normalization to the document size was made. To neutralize the effect of document size, the LOF score is multiplied by (1+*log*(*len*(*doc*))) where *len*(*doc*) is the length of the document. Table 6.6 shows how normalization affected the performance negatively.

Table 6.6: Document Normalization (LISA Sample II)

(a) 1 <= *K* <= 10

Top J	Precision	Recall	F-Score
1	**0.486**	**0.486**	**0.486**
2	0.443	0.471	0.452
3	0.4	0.452	0.415
4	0.371	0.445	0.391
5	0.343	0.438	0.366
6	0.329	0.443	0.354
7	0.306	0.432	0.332
8	0.286	0.423	0.313
9	0.263	0.41	0.291
10	0.249	0.403	0.276

As a result, normalizing the document has negative effect on the performance of the proposed model, and thus it wouldn't be considered in the optimizations of the model.

6.6 Comparing To A Benchmark Model

Up to this step, the model achieved an F-score of 51.4% on the first result using the maximum value of *K*, with stop words removal, TFIDF, and using the Euclidean distance. Nevertheless, the performance should be compared with other benchmark models. The benchmark is done against the mean of the embedding model. The model takes BERT embedding of each token, then applies arithmetic mean on the values, resulting in a single-vector representation of the document. The comparison in Tables 6.7 a, b, and c shows that the proposed algorithm outperforms the mean of embedding model for both Euclidean and Cosine distances.

Table 6.7: The Proposed Model Vs Mean Of Embedding (LISA Sample II)

(a) The Proposed Model

Top J	Precision	Recall	F-Score
1	**0.514**	**0.514**	**0.514**
2	0.429	0.457	0.438
3	0.39	0.443	0.406
4	0.364	0.436	0.383
5	0.337	0.424	0.358
6	0.319	0.424	0.342
7	0.302	0.428	0.328
8	0.282	0.419	0.309
9	0.263	0.41	0.291
10	0.249	0.403	0.276

(b) Mean Of Embedding (Euclidean)

Top J	Precision	Recall	F-Score
1	0.486	0.486	0.486
2	0.471	0.479	0.474
3	0.454	0.478	0.461
4	0.433	0.47	0.444
5	0.416	0.462	0.428
6	0.403	0.456	0.416
7	0.388	0.448	0.402
8	0.375	0.444	0.391
9	0.364	0.44	0.381
10	0.355	0.439	0.372

(c) Mean Of Embedding (Cosine)

Top J	Precision	Recall	F-Score
1	0.446	0.446	0.446
2	0.451	0.456	0.452
3	0.439	0.459	0.445
4	0.428	0.46	0.437
5	0.414	0.456	0.426
6	0.403	0.453	0.415
7	0.391	0.45	0.405
8	0.379	0.447	0.395
9	0.37	0.446	0.386
10	0.361	0.445	0.379

Although the proposed model performs better on the first result, it performs less when compared to the top 2, 3...etc.

This indicates that the enhancements worked on better ranking the documents and providing more relevant documents on the top results. Moreover, if results are compared with the previous small subset (27 sample documents), the proposed model works better on smaller datasets and provides better performance on retrieving relevant documents on the top 1, 3, and 5 results.

6.7 Hybrid Solution

The experiment in previous section shows how the proposed model performs better on smaller datasets and ranks documents better than bigger datasets, which can limit the scale of application of this model. Thus, to enhance the ranking model, we propose a hybrid solution that combines one of the state-of-the-art ranking models (BM25), and the proposed model. The hybrid solution ranks the documents based on TFIDF concepts using BM25 and returns the top j documents, then the returned documents are processed as a subset using the proposed model, and the subset is re-ranked again. This solution is hypothesized to provide better results. The hypothesis is tested over several subsets extracted from BM25. Using subsets of top 50, 100, 200, 500, and 1000 documents ranked by BM25 and extracted and the result shown in Table 6.8.

Table 6.8: Hybrid Solution (LISA Sample II)

(a) Subset of top 50 documents Top J

Top J	Precision	Recall	F-Score
1	0.514	0.514	0.514
2	0.429	0.457	0.438
3	0.39	0.443	0.406
4	0.364	0.436	0.383
5	0.331	0.414	0.351
6	0.314	0.414	0.335
7	0.298	0.419	0.322
8	0.279	0.41	0.304
9	0.263	0.404	0.29
10	0.246	0.394	0.272

(b) Subset of top 100 documents

Top J	Precision	Recall	F-Score
1	0.514	0.514	0.514
2	0.429	0.457	0.438
3	0.39	0.443	0.406
4	0.364	0.436	0.383
5	0.331	0.414	0.351
6	0.314	0.414	0.335
7	0.298	0.419	0.322
8	0.279	0.41	0.304
9	0.26	0.4	0.286
10	0.246	0.394	0.272

(c) Subset of top 200 documents

Top J	Precision	Recall	F-Score
1	0.514	0.514	0.514
2	0.429	0.457	0.438
3	0.39	0.443	0.406
4	0.364	0.436	0.383
5	0.337	0.424	0.358
6	0.319	0.424	0.342
7	0.302	0.428	0.328
8	0.282	0.419	0.309
9	0.263	0.41	0.291
10	0.249	0.403	0.276

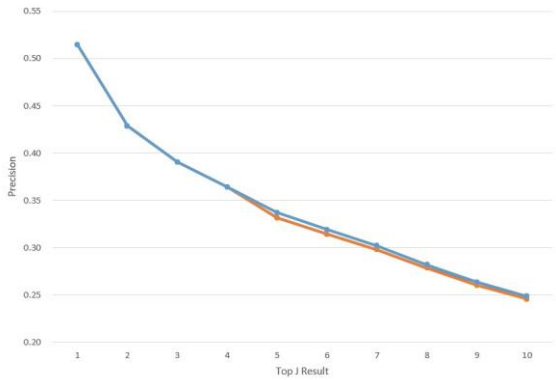
(d) Subset of top 500 documents

Top J	Precision	Recall	F-Score
1	0.514	0.514	0.514
2	0.429	0.457	0.438
3	0.39	0.443	0.406
4	0.364	0.436	0.383
5	0.337	0.424	0.358
6	0.319	0.424	0.342
7	0.302	0.428	0.328
8	0.282	0.419	0.309
9	0.263	0.41	0.291
10	0.249	0.403	0.276

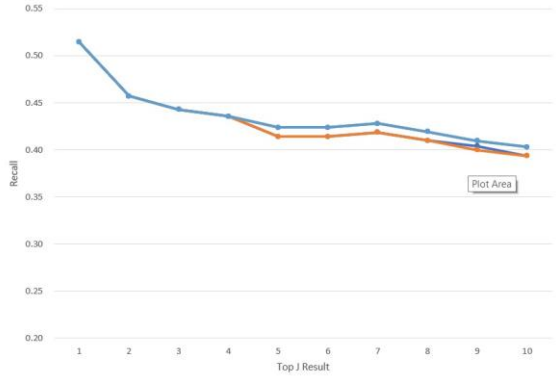
(e) Subset of top 1000 documents

Top J	Precision	Recall	F-Score
1	0.514	0.514	0.514
2	0.429	0.457	0.438
3	0.39	0.443	0.406
4	0.364	0.436	0.383
5	0.337	0.424	0.358
6	0.319	0.424	0.342
7	0.302	0.428	0.328
8	0.282	0.419	0.309
9	0.263	0.41	0.291
10	0.249	0.403	0.276

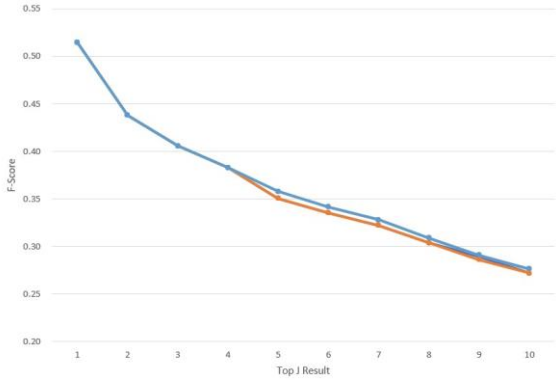
The results in Table 6.8 are better interpreted in Figures 6.2 a, b, and c. Although the scores weren't affected much, the ranking model reduced the computations needed for the algorithm; instead of processing 1042 documents, the results are almost the same with 50 or 100 documents. In other words, there is no need to process the whole dataset using the proposed algorithm to get only 0.01 improvement, and since BM25 is a very fast algorithm, it is more feasible to use the hybrid solution for improving the run time.



(a) Hybrid Solution (Top J Result vs Precision)



(b) Hybrid Solution (Top J Result vs Recall)



(c) Hybrid Solution (Top J Result vs F-Score)

Figure 6.2: Hybrid Solution (LISA Sample II)

As a result, using the hybrid solution is preferable since it lowers the computations needed while providing same performance.

6.8 Changing The Sampling Method

Another test is to check the effect of sampling on the performance. To do so, we took another sample of the same size, but with a different methodology; the queries defined in the dataset differ in the number of relevant documents, which sometimes might introduce more weight for irrelevant documents and causes lower performance. To solve this issue, and treat all queries the same, the sample is taken by taking all relevant documents for the query, and selecting random n_i documents, where n_i = number of relevant documents of query q_i, and i is the query ID. The total sample size is 739 documents for 35 queries. This sample will be referred as" LISA Sample III". The sample is tested on the proposed algorithm with $K = max(d_i)$ where d_i is the set of points in space for iteration i, with stop words removal, and using TFIDF for term weighting. The mentioned settings on the proposed algorithm will be referred as" Basic Settings". The results in Table 6.9 a show an increase in the performance up to 60%, and this implies better performance especially when compared with the mean of embedding approach, which achieved up to 54% for

F-score. On the other hand, the hybrid approach shows better results on the top 1, 2, and 3 results, and better results on lower top K documents ranked by the BM25 model. Results are shown in Table 6.9 c-e.

Table 6.9: Sampling Effect (LISA Sample III)

(a) Proposed Algorithm (Basic Settings)

Top J	Precision	Recall	F-Score
1	0.6	**0.6**	**0.6**
2	0.514	**0.543**	**0.524**
3	0.438	0.49	0.453
4	0.407	0.488	0.429
5	0.406	0.51	0.431
6	0.371	0.487	0.397
7	0.355	0.502	0.386
8	0.321	0.481	0.353
9	0.305	0.476	0.337
10	0.291	0.471	0.324

(b) Mean Of Embedding

Top J	Precision	Recall	F-Score
1	0.543	0.543	0.543
2	0.514	0.529	0.519
3	0.492	0.521	0.501
4	0.473	0.513	0.484
5	0.456	0.504	0.469
6	0.439	0.494	0.452
7	0.425	0.488	0.44
8	0.414	0.484	0.43
9	0.403	0.483	0.421
10	0.394	0.483	0.413

(c) Hybrid Solution (top 50 documents)

Top J	Precision	Recall	F-Score
1	0.6	**0.6**	**0.6**
2	0.529	**0.557**	**0.538**
3	0.457	0.51	0.472
4	0.407	0.486	0.428
5	0.406	0.506	0.43
6	0.381	0.491	0.405
7	0.359	0.491	0.386
8	0.339	0.489	0.369
9	0.327	0.487	0.357
10	0.317	0.488	0.349

(d) Hybrid Solution (top 100 documents)

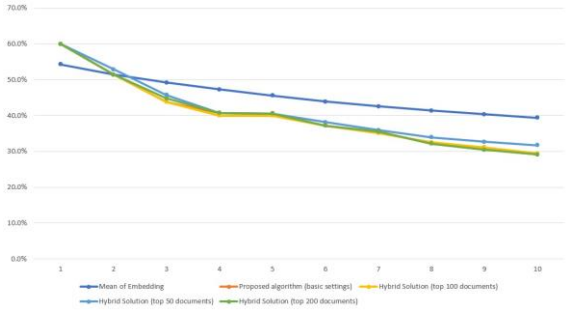
Top J	Precision	Recall	F-Score
1	0.6	0.6	0.6
2	0.514	0.543	0.524
3	0.438	0.49	0.453
4	0.4	0.479	0.421
5	0.4	0.5	0.424
6	0.371	0.491	0.398
7	0.351	0.492	0.38
8	0.325	0.484	0.357
9	0.311	0.482	0.344
10	0.294	0.475	0.327

(e) Hybrid Solution (top 200 documents)

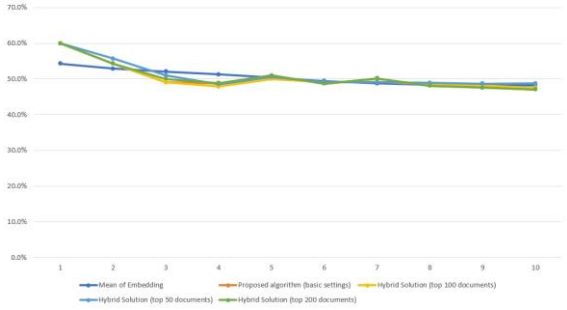
Top J	Precision	Recall	F-Score
1	0.6	0.6	0.6
2	0.514	0.543	0.524
3	0.448	0.5	0.463
4	0.407	0.488	0.429
5	0.406	0.51	0.431
6	0.371	0.487	0.397
7	0.355	0.502	0.386
8	0.321	0.481	0.353
9	0.305	0.476	0.337
10	0.291	0.471	0.324

Figure 6.3 shows how the performance differs over different approaches. Both basic settings and the hybrid solution perform better than the mean of embedding for the first

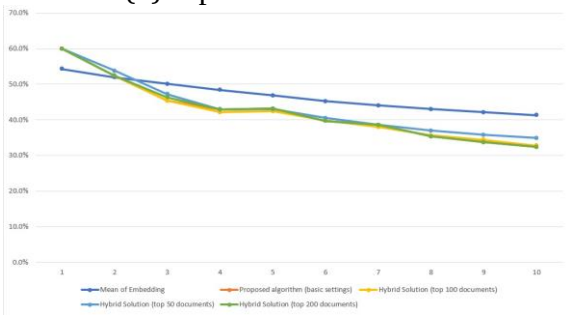
two results, then the latter outperforms other approaches for other top J values. Moreover, having a lower subset of top results extracted by BM25 enhances the results. Having only the top 50 documents provided better precision, recall, and F-score results for larger top J values, while maintaining the performance for top 1 and 2 results.



(a) Top J Result vs Precision



(b) Top J Result vs Recall



(c) Top J Result vs F-Score

Figure 6.3: Sampling Effect (LISA Sample III)

From the results in Figure 6.3, the recall of the hybrid solution is better than the mean of embedding approach since it outperforms the latter in the first two results significantly, and has almost the same performance for other results. On the other hand, the precision of the hybrid solution is only better for the first result, but falls below the mean of embedding.

However, the results show high potential for the hybrid solution and can be optimized to provide results that might outperform other rivals significantly.

This test is to make sure the sampling doesn't change the results obtained previously. On contrast, it showed the potential of performance increase on different datasets.

6.9 Testing Over Different Datasets

This section applies different testing experiments to the Cranfield dataset. This dataset is different from the LISA dataset in its size and topic enclosed. The reason for testing the algorithm over this dataset is to test the change of topic on the performance.

The tests include the algorithm in the basic settings, the hybrid solution using a subset of top 100 documents, and the mean of embedding approach. The results in Table 6.10 show that the hybrid solution significantly outperforms the mean of the embedding approach. The precision, recall, and F-score achieved 50% while the mean of embedding didn't exceed 8%. Also, figure 6.4 interprets the results.

Table 6.10: Cranfield Dataset

(a) Proposed Algorithm (Basic Settings)

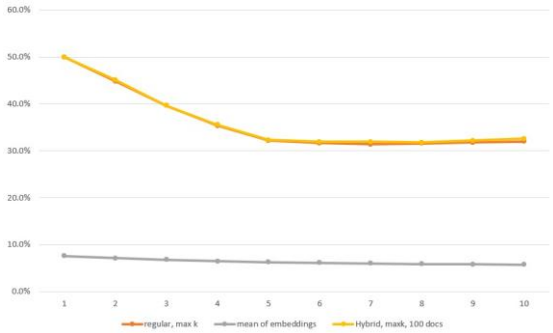
Top J	Precision	Recall	F-Score
1	0.5	**0.5**	**0.5**
2	0.449	0.449	0.449
3	0.394	0.397	0.395
4	0.337	0.353	0.344
5	0.291	0.322	0.303
6	0.271	0.317	0.287
7	0.252	0.314	0.273
8	0.238	0.316	0.263
9	0.223	0.318	0.253
10	0.208	0.32	0.242

(b) Mean Of Embedding

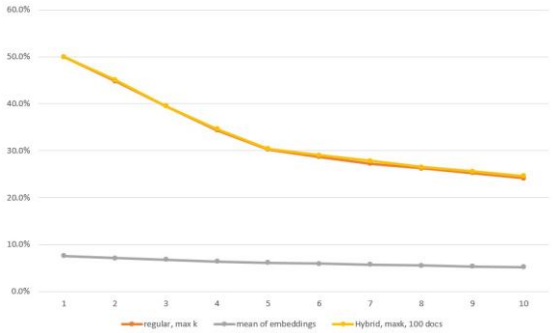
Top J	Precision	Recall	F-Score
1	0.076	0.076	0.076
2	0.071	0.071	0.071
3	0.067	0.068	0.068
4	0.064	0.065	0.064
5	0.061	0.063	0.061
6	0.058	0.061	0.059
7	0.056	0.06	0.057
8	0.053	0.059	0.055
9	0.051	0.058	0.054
10	0.05	0.057	0.052

(c) Hybrid Solution (top 100 documents)

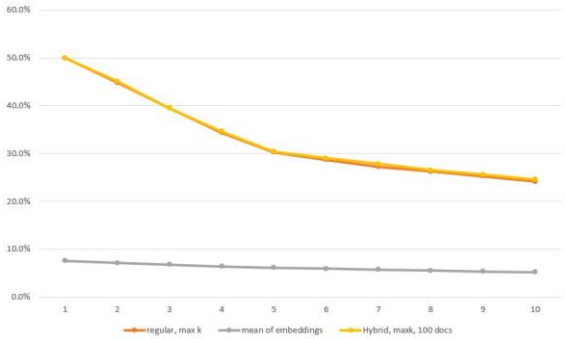
Top J	Precision	Recall	F-Score
1	0.5	0.5	0.5
2	**0.451**	**0.451**	**0.451**
3	0.394	0.397	0.395
4	0.339	0.356	0.346
5	0.293	0.323	0.304
6	0.274	0.319	0.29
7	0.258	0.319	0.278
8	0.24	0.318	0.265
9	0.226	0.322	0.256
10	0.212	0.326	0.246



(a) Top J Result vs Precision



(b) Top J Result vs Recall



(c) Top J Result vs F-Score

Figure 6.4: Evaluation Over Cranfield Dataset

The same set of testing experiments are applied on the AILA sample. The mentioned dataset contains large documents that discuss legal topics, and due to the size of the documents, a sample of 381 documents and 50 queries is used for testing.

The results in Table 6.11 show that the basic settings of the proposed model performed better than both the hybrid solution and the mean of embedding approach. The

difference in performance shows that the document size really affects the performance of the algorithm. However, the hybrid solution is still very close to the results of the basic settings, and with some optimization, it might provide the same or even better results.

Table 6.11: Evaluation Over AILA Sample

(a) Proposed Algorithm (Basic Settings)

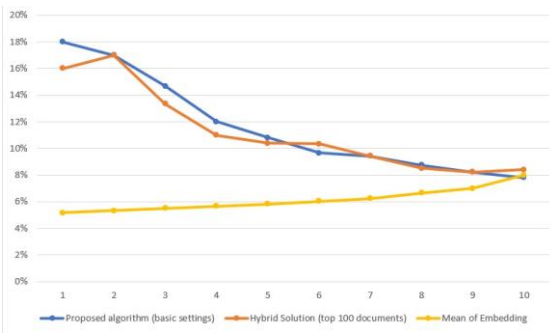
Top J	Precision	Recall	F-Score
1	0.18	**0.18**	**0.18**
2	0.17	0.2	**0.18**
3	0.147	0.2	0.162
4	0.12	0.185	0.136
5	0.108	0.185	0.125
6	0.097	0.188	0.116
7	0.094	0.207	0.118
8	0.088	0.23	0.114
9	0.082	0.236	0.11
10	0.078	0.239	0.105

(b) Mean Of Embedding

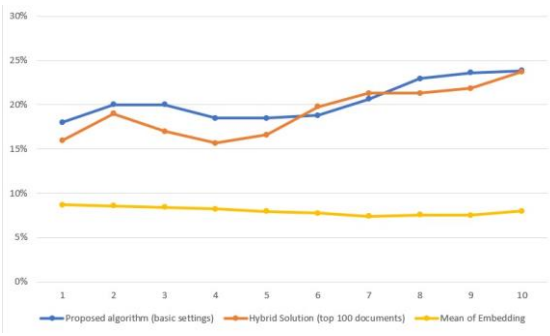
Top J	Precision	Recall	F-Score
1	0.08	0.08	0.08
2	0.07	0.075	0.072
3	0.067	0.076	0.069
4	0.063	0.074	0.066
5	0.06	0.078	0.064
6	0.058	0.08	0.063
7	0.057	0.082	0.062
8	0.055	0.084	0.061
9	0.053	0.086	0.06
10	0.052	0.087	0.058

(c) Hybrid Solution (top 100 documents)

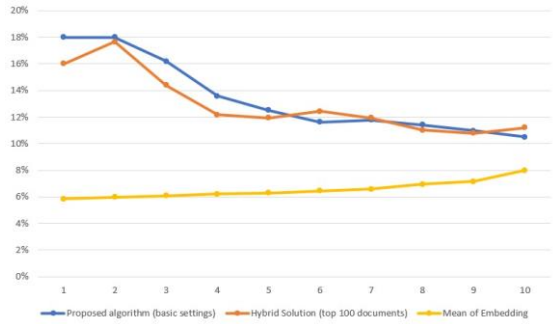
Top J	Precision	Recall	F-Score
1	0.16	0.16	0.16
2	0.17	0.19	0.177
3	0.133	0.17	0.144
4	0.11	0.157	0.122
5	0.104	0.166	0.119
6	0.103	0.198	0.124
7	0.094	0.213	0.119
8	0.085	0.213	0.11
9	0.082	0.219	0.108
10	0.084	0.237	0.112



(a) Top J Result vs Precision



(b) Top J Result vs Recall



(c) Top J Result vs F-Score

Figure 6.5: Evaluation Over AILA Sample

The results are illustrated in Figure 6.5. Figures a, b, and c show how the proposed model in both its basic settings and hybrid solution compete for the best, while the mean of embedding has less performance. However, the performance results for the three models are very low, and this is expected due to the fact that the dataset is large, the cases contain a lot of different topics that would introduce multiple densities for the same document.

Moreover, the queries are also large, which increases the complexity in the EVSM and increases the challenge of the retrieval model to find the best answer. Another reason is the challenge introduced in the dataset by its source. This dataset is set for competition over several tasks, which introduces difficulties that most retrieval models would fall in. On the other hand, the proposed approaches performed better than other state-of-the-art models and achieved remarkable performance of 18% for the top one against the mean of embedding which performed less than 6% in F-score.

The results for the different datasets and samples shows how the proposed model outperformed the state-of-the-art mean of embedding model, and tested the generalization of the proposed model on datasets that differs in size, topics, and document lengths.

6.10 Final Model

The testing experiments in previous section shows how the performance is significantly improved by the optimizations on the basic model. The final model can be illustrated by the following Pseudo Code:

Algorithm 2 Proposed Algorithm Pseudo Code

1: Calculate TFIDF for the Dataset

2: **for** Every Query point q_i **do**

3: Run BM25 and select *TopJ* documents

4: **for** Every Document D_j in *TopJ* **do**

5: $D_j = \{d_1, d_2 ... d_n\}$

6: $space = \{q_i, d_1, d_2 ... d_n\}$

7: Remove Stop words from space points

8: Calculate distance matrix *DM*

9: **for** Each point in space **do**

10: Find the K^{th} nearest neighbor

11: Calculate RD $RD(A, B) = MaxK - distance(B), d(A, B)$

12: Calculate LRD $LRD(A) = \frac{1}{\sum_{B \in N_K(A)} RD(A,B)/N_K(A)}$

13: Calculate LOF $LOF(A) = \frac{\sum_{B \in N_K(A)} LRD(B)}{N_K(A) \times LRD(A)}$

14: $LOF = LOF/TFIDF(q_i)$

15: **end for**

16: **end for**

17: **end for**

The final model results over different is compared with the results of the state-of-the art mean of embedding model. The comparison is illustrated in table 6.12. The proposed model outperforms the benchmark model in different situations and for different dataset types. However, it performs less than the benchmark on one sample for the third to tenth results. On the other hand, it significantly shows that it outperforms the model on the other two datasets.

Table 6.12: Proposed Algorithm Vs Mean Of Embedding Performance

(a) LISA Sample III Hybrid Solution (top 50 documents)

Top J	Precision	Recall	F-Score
1	0.6	0.6	0.6
2	0.529	0.557	0.538
3	0.457	0.51	0.472
4	0.407	0.486	0.428
5	0.406	0.506	0.43
6	0.381	0.491	0.405
7	0.359	0.491	0.386
8	0.339	0.489	0.369
9	0.327	0.487	0.357
10	0.317	0.488	0.349

(b) LISA Sample III Mean Of Embedding

Top J	Precision	Recall	F-Score
1	0.543	0.543	0.543
2	0.514	0.529	0.519
3	0.492	0.521	0.501
4	0.473	0.513	0.484
5	0.456	0.504	0.469
6	0.439	0.494	0.452
7	0.425	0.488	0.44
8	0.414	0.484	0.43
9	0.403	0.483	0.421
10	0.394	0.483	0.413

(c) Cranfield Dataset Hybrid Solution (top 100 documents)

Top J	Precision	Recall	F-Score
1	0.5	0.5	0.5
2	0.451	0.451	0.451
3	0.394	0.397	0.395
4	0.339	0.356	0.346
5	0.293	0.323	0.304
6	0.274	0.319	0.29
7	0.258	0.319	0.278
8	0.24	0.318	0.265
9	0.226	0.322	0.256
10	0.212	0.326	0.246

(d) Cranfield Dataset Mean Of Embedding

Top J	Precision	Recall	F-Score
1	0.076	0.076	0.076
2	0.071	0.071	0.071
3	0.067	0.068	0.068
4	0.064	0.065	0.064
5	0.061	0.063	0.061
6	0.058	0.061	0.059
7	0.056	0.06	0.057
8	0.053	0.059	0.055
9	0.051	0.058	0.054
10	0.05	0.057	0.052

(e) Aila Sample (Basic Settings)

Top J	Precision	Recall	F-Score
1	0.18	0.18	0.18
2	0.17	0.2	0.18
3	0.147	0.2	0.162
4	0.12	0.185	0.136
5	0.108	0.185	0.125
6	0.097	0.188	0.116
7	0.094	0.207	0.118
8	0.088	0.23	0.114
9	0.082	0.236	0.11
10	0.078	0.239	0.105

(f) Aila Sample Mean Of Embedding

Top J	Precision	Recall	F-Score
1	0.076	0.076	0.076
2	0.071	0.071	0.071
3	0.067	0.068	0.068
4	0.064	0.065	0.064
5	0.061	0.063	0.061
6	0.058	0.061	0.059
7	0.056	0.06	0.057
8	0.053	0.059	0.055
9	0.051	0.058	0.054
10	0.05	0.057	0.052

Table 6.13 shows the mean of the results for the novel algorithm and the benchmark model. The proposed model shows significant increase in the performance vs the benchmark model, which follows the assumptions of this researches that there is high potential for using density-based approaches for document matching.

Table 6.13: Final Model Vs Mean Of Embedding

(a) Proposed Model

Top J	Precision	Recall	F-Score
1	0.427	0.427	0.427
2	0.383	0.403	0.39
3	0.333	0.369	0.343
4	0.289	0.342	0.303
5	0.269	0.338	0.286
6	0.251	0.333	0.27
7	0.237	0.339	0.261
8	0.222	0.346	0.249
9	0.212	0.348	0.241
10	0.202	0.351	0.233

(b) Mean Of Embedding Model

Top J	Precision	Recall	F-Score
1	0.233	0.233	0.233
2	0.218	0.225	0.221
3	0.209	0.222	0.213
4	0.2	0.217	0.205
5	0.192	0.215	0.198
6	0.185	0.212	0.191
7	0.179	0.21	0.186
8	0.174	0.209	0.182
9	0.169	0.209	0.178
10	0.165	0.209	0.174

Chapter 7: Conclusions And Future Works

In this work, we propose a novel density algorithm inspired by LOF. The proposed algorithm lies in an unexplored area of research that utilizes density-based algorithms in IR tasks. In the previous chapter, a set of tests on three different datasets was used to show how the performance of the proposed algorithm outperformed the reference model mean of the embedding. Moreover, the conducted experiments have shown how the proposed algorithm was developed and how its performances are affected by each optimization step. Using the Euclidean distance matrix, the stop words removal and the term-weighting strategy based on TFIDF improve the algorithm performances.

Secondly, using BM25 to rank the dataset and use a subset of the top 100 documents and pass it to the proposed algorithm increased recall performance and F-score for results in the top 2 and top 3.

Finally, the algorithm is tested over several values of *K*, and the results showed that the larger the *K*, the higher the performance. Thus, using the maximum value for *K* (the number of document points) provides the best performance.

Table 6.12 shows the performance of the proposed model on different datasets.

The final model achieved a mean F-score of up to 43%, where the benchmark means of the embedding approach didn't exceed 23%.

The performance of the proposed model outperforms the mean of embedding in general. However, for LISA sample III, the mean of embedding performs better than the proposed model when taking the larger value of top J.

Moreover, although the hybrid solution didn't improve the F-score for the first result, it affected the recall positively. It also reduced the run time significantly since dealing with a lower set of documents reduces the needed computation and resources for the

algorithm. Nevertheless, as seen in the tests, the proposed algorithm works better in less complex spaces. In other words, the smaller the set of documents processed by the algorithm, the better the results.

This research proves the concept of using the density-based algorithm in matching models, which is an untapped area that the thesis explored. Moreover, the study proposes a novel density algorithm that outperforms the benchmark model. From the results discussed, the approach opens the opportunity for further investigation and development and sets the ground for further research and the use of other density approaches.

The main issues faced during the research are computational power and time limitation. For future work, the proposed algorithm needs further investigation on metrics that enhance the model's performance. Moreover, the model could be tested against other models that don't rely on neural embedding. Also, the model is tested over English only, but further investigation for other languages would enrich the approach. Finally, using different density-based algorithms and other optimization added methods and algorithms that enhance the algorithm is an additional research area that should be investigated.

Bibliography

Breuniq, M. M., Kriegel, H. P., Ng, R. T. & Sander, J. (2000), 'Lof: Identifying density-based local outliers', *SIGMOD Record (ACM Special Interest Group on Management of Data)* **29**.

Caˆmara, A. & Hauff, C. (2020), 'Diagnosing bert with retrieval heuristics', *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **12035 LNCS**, 605–618.

Dai, Z., Wang, X., Ni, P., Li, Y., Li, G. & Bai, X. (2019), Named entity recognition using bert bilstm crf for chinese electronic health records, *in* '2019 12th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)', pp. 1–5.

Devlin, J., Chang, M. W., Lee, K. & Toutanova, K. (2019), 'Bert: Pre-training of deep bidirectional transformers for language understanding', *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference* **1**, 4171–4186.

Eminagaoglu, M. (2020), 'A new similarity measure for vector space models in text classification and information retrieval', *Journal of Information Science* p.0165551520968055.

Evangelopoulos, N. E. (2013), 'Latent semantic analysis', *Wiley Interdisciplinary Reviews: Cognitive Science* **4**, 683–692.

Karnalim, O. (2015), 'Extended vector space model with semantic relatedness on java archive search engine', *Jurnal Teknik Informatika dan Sistem Informasi* **1**.

Khalid, M. J., Irfan, M., Ali, T., Gull, M., Draz, U., Glowacz, A., Sulowicz, M., Dziechciarz, A., Alkahtani, F. S. & Hussain, S. (2020), 'Integration of discrete wavelet transform, dbscan, and classifiers for efficient content based image retrieval', *Electronics (Switzerland)* **9**, 1–15.

Lazarevic, A., Ertoz, L., Kumar, V., Ozgur, A. & Srivastava, J. (2003), A comparative study of anomaly detection schemes in network intrusion detection, *in* 'Proceedings of the 2003 SIAM international conference on data mining', SIAM, pp. 25–36.

Manning, C. D., Raghavan, P. & Schu¨tze, H. (2010), 'Introduction to modern information retrieval, 3rd edition', *Australian Academic and Research Libraries* **41**, 305–306.

Mitra, B. & Craswell, N. (2018), 'An introduction to neural information retrieval', *Foundations and Trends in Information Retrieval* **13**, 1–129.

Paskaleva, B. S. & Bochev, P. B. (2012), A vector space model for information retrieval with generalized similarity measures., Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).

Pennington, J., Socher, R. & Manning, C.D.(2014), 'Glove:Global vectors for word representation', *Association for Computational Linguistics* **Proceeding**, 1532–1543.

Peterson, J. C., Chen, D. & Griffiths, T. L. (2020), 'Parallelograms revisited: Exploring the limitations of vector space models for simple analogies', *Cognition* **205**, 104440. **URL:** *https://doi.org/10.1016/j.cognition.2020.104440*

Reshma, P., Rajagopal, S. & Lajish, V. (2020), A novel document and query similarity indexing using vsm for unstructured documents, *in* '2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)', IEEE, pp. 676–681.

Roshdi, A. & Roohparvar, A. (2015), 'Review: Information retrieval techniques and applications'. **URL:** *www.ijcncs.org*

Rumelhart, D. E. & Abrahamson, A. A. (1973), 'A model for analogical reasoning', *Cognitive Psychology* **5**, 1–28.

Sahlgren, M. (2005), An introduction to random indexing, *in* 'Methods and applications of semantic indexing workshop at the 7th international conference on terminology and knowledge engineering'.

Salton, G., Wong, A. & Yang, C. S. (1975), 'A vector space model for automatic indexing', *Communications of the ACM* **18**, 613–620.

Schubert, E., Zimek, A. & Kriegel, H. P. (2014), 'Local outlier detection reconsidered: A generalized view on locality with applications to spatial, video, and network outlier detection', *Data Mining and Knowledge Discovery* **28**.

Shahmirzadi, O., Lugowski, A. & Younge, K. (2019), Text similarity in vector space models: a comparative study, *in* '2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)', IEEE, pp. 659–666.

Singh, K. N., Mahanta, A. K. & Devi, H. M. (2017), 'Document representation techniques and their effect on the document clustering and classification: A review', *International Journal of Advanced Research in Computer Science* **8**, 1780–1784. **URL:** *https://www.ijarcs.info/index.php/Ijarcs/article/viewFile/3822/3613*

Tanaka, H., Shinnou, H., Cao, R., Bai, J. & Ma, W. (2020), Document classification by word embeddings of bert, Vol. 1215 CCIS, Springer, pp. 145–154.

Tsatsaronis, G. & Panagiotopoulou, V. (2009), A generalized vector space model for text retrieval based on semantic relatedness, *in* 'Proceedings of the Student Research

Workshop at EACL 2009', Association for Computational Linguistics, Athens, Greece, pp. 70–78. **URL:** *https://aclanthology.org/E09-3009*

Van der Maaten, L. & Hinton, G. (2008), 'Visualizing data using t-sne.', *Journal of machine learning research* **9**(11).

Wikipedia (2022), 'Precision and recall — Wikipedia, the free encyclopedia', http://en.wikipedia.org/w/index.php?title=Precision%20and%20recall&oldid=1069330375. [Online; accessed 25-February-2022].

Wong, S. K. M., Ziarko, W. & Wong, P. C. N. (1985), Generalized vector spaces model in information retrieval, *in* 'Proceedings of the 8th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval', SIGIR '85, Association for Computing Machinery, New York, NY, USA, p. 18–25. **URL:** *https://doi.org/10.1145/253495.253506*

Wu, C. & Wang, B. (2017), Extracting topics based on word2vec and improved jaccard similarity coefficient, *in* '2017 IEEE Second International Conference on Data Science in Cyberspace (DSC)', pp. 389–397.

Zhang, Y., Rahman, M. M., Braylan, A., Dang, B., Chang, H., Kim, H., McNamara, Q., Angert, A., Banner, E., Khetan, V., McDonnell, T., Nguyen, A. T., Xu, D., Wallace, B. C. & Lease, M. (2016), 'Neural information retrieval: A literature review', *CoRR* **abs/1611.06792**. **URL:** *http://arxiv.org/abs/1611.06792*

Appendix

The appendix chapter contains the code structure of the proposed algorithm.

The source code can be found on GitHub repository following this link: https: //github.com/loai-bitawi/Thesis_Rep.git The code structure follows object oriented programming that allows the code to be extendable, usable, and easy to understand and modify. the next section describes each file mentioned in the GitHub repository.

A.1 Dataset Folders

The three mentioned datasets in the thesis with the format described previously are located in three folders that hold the names of the three datasets. In each folder, there is the data folder, and the offline files that contain the indexing files and the

results.

A.2 Data Parser.py

This python source code contains the class responsible for parsing the mentioned datasets in a structured format (JSON) format that can be processed later by the algorithm.

A.3 Preprocessing.py

This code contains the needed functions to prepare the data for BERT embedding, stop words removal, generating mapping files, and transform text documents to vector format that can be processed in the EVSM.

Also, the code contains function to select a random sample from any of the mentioned datasets.

A.4 Distance Matrix.py

This code is responsible of calculating the distance between document and query points using the vector representation extracted from preprocessing phase. The code also splits the matrices into chunks of 150 documents that is stored offline for using it later in

calculating LOF instead of calculating the distances on spot. This method enhances the performance of the model in terms of computational power and time needed to run the model.

A.5 Density Algorithm.py

This is the core source code that implements the proposed algorithm. The code allows the user to modify some hyper parameters that can be used for fine tuning the model.

A.6 Mean of Embedding.py

This python file implements and calculates the results of the mean of embedding model.

A.7 BM25.py

This file implements the BM25 ranker that is used in the proposed model.

A.8 results.py

This source code calculates the precision, recall, and F-score from the output of Density algorithm.py.

A.9 main.py

This code is used to run the whole model using single command, and allows for fine tuning the model.

A.10 config.py

this file contains the credentials needed to run the model. The credentials currently contain the main path of the project and the needed dependencies that should be imported to run the model. However, this part can be expanded to contain any

other credentials like passwords, database users, IP addresses, ...etc.

الملخص

تلعب محركات البحث دورًا أساسيًا في تلبية احتياجات حياتنا اليومية من المعلومات. وبالتالي، هناك حاجة دائمة إلى إجراء تحسينات على الخوارزميات الموجودة في انظمة استرجاع المعلومات(Systems Information Retrieval) في هذه المحركات. تظهر الدراسات أن النماذج التي تعتمد على Neural networks وVSM(Vector Space Model) هي أفضل النماذج التي يمكن استخدامها في أنظمة استرجاع المعلومات. علاوة على ذلك ، لم تذكر الدراسات عن استخدام الخوارزميات القائمة على الكثافة (Density-Based Algorithms) لمعرفة نسبة التشابه بين المستندات في انظمة استرجاع المعلومات. لذا، يستكشف هذا البحث مجالا جديدا لم يتم استكشافه، حيث يتم استخدام الخوارزمية القائمة على الكثافة (Density-Based Algorithms) لمطابقة نص البحث (Query) مع المستندات الموجودة في قاعدة البيانات (Documents). تقوم الخوارزمية المقترحة في هذا البحث بتمثيل الكلمات الموجودة في المستندات على شكل متجهات في الفراغ يتم انشاؤها باستخدام Model BERT Embedding، وهي عبارة عن شبكة عصبية ضخمة (Deep Neural Network) تم بناؤها وتدريبها من قبل فرق الابحاث التابعة لشركة جوجل، وبناء عليها يتم تمثيل النصوص (المستندات) كسحب من النقاط (Cloud Points of) في الفراغ، حيث تقوم الشبكة العصبية بتحويل الكلمة الى متجه يمثل الكلمة ويتضمن السياق الخاص بالكلمة، بحيث نفس الكلمة قد يكون لها اكثر من متجه يعبر عنها حسب السياق الذي ذكرت فيه. يستخدم الجزء المطابق حلاً هجينا يزيد من دقة البيانات المسترجعة ويقلل من الوقت اللازم لعملية البحث، حيث تستخدم الخوارزمية نموذج التصنيف BM25 لتصنيف مجموعة البيانات بأكملها واستخراج المستندات التي حصلت على اعلى تصنيف (أعلى 10 او 100 مستند)، وتمريرها إلى الخوارزمية القائمة على الكثافة (Density-Based Matching). الخوارزمية المعتمدة على الكثافة مستوحاة من خوارزمية الكشف عن الانحرافات (LOF) والتي تهدف إلى معرفة ما إذا كانت نقاط نص البحث (Query points) بعيدة عن سحابة النقاط الخاصة بالمستند(Document Density)، ونظرا لأن خوارزمية LOF تقيس درجة الانحراف (البعد) عن مركز كثافة النقاط، يتم استخدام درجات انحراف النقاط الخاصة بنص البحث لمعرفة درجة ارتباط المستند بنص البحث. بعد ذلك، يتم اعادة ترتيب المستندات حسب درجة ارتباطها بنص البحث. علاوة على ذلك، تتضمن الخوارزمية المقترحة إعطاء وزن للنقاط حسب اهميتها في المستند باستخدام خوارزمية TFIDF، والذي يقوم بحسابات احصائية قائمة على تكرار الكلمة في المستند وتكرارها في جميع المستندات وإعطاء كل كلمة وزن بناء على تكرارها مما يقلل

من تأثير الكلمات التي لا صلة لها بموضوع المستند او الكلمات التي تستخدم لربط الجمل وغيرها. تم اختبار الخوارزمية المقترحة ومقارنتها بخوارزمية البحث المشهورة (Mean of Embedding). أظهرت النتائج أن الخوارزمية المقترحة تتفوق في الأداء على الخوارزمية الاخرى وعلى اكثر من قاعدة بيانات وفي العديد من الاختبارات. تظهر الخوارزمية أفقا عاليا يسمح بالاستكشاف اكثر في هذا المجال وإجراء التحسينات المستقبلية التي من المتوقع ان تقدم خوارزمية أفضل لأنظمة استرجاع المعلومات (Information Retrieval Systems).