data

MDPI

Article

Multi-Layer Web Services Discovery Using Word Embedding and Clustering Techniques

Waeal J. Obidallah [1,*], Bijan Raahemi [2] and Waleed Rashideh [1]

[1] College of Computer and Information Sciences, Imam Mohammad Ibn Saud Islamic University (IMSIU), Riyadh 11673, Saudi Arabia; wmrashideh@imamu.edu.sa
[2] Knowledge Discovery and Data Mining Lab, Telfer School of Management University of Ottawa, Ottawa, ON K1H 8M5, Canada; braahemi@uottawa.ca
* Correspondence: wobaidallah@imamu.edu.sa

Abstract: We propose a multi-layer data mining architecture for web services discovery using word embedding and clustering techniques to improve the web service discovery process. The proposed architecture consists of five layers: web services description and data preprocessing; word embedding and representation; syntactic similarity; semantic similarity; and clustering. In the first layer, we identify the steps to parse and preprocess the web services documents. In the second layer, Bag of Words with Term Frequency–Inverse Document Frequency and three word-embedding models are employed for web services representation. In the third layer, four distance measures, namely, Cosine, Euclidean, Minkowski, and Word Mover, are considered to find the similarities between Web services documents. In layer four, WordNet and Normalized Google Distance are employed to represent and find the similarity between web services documents. Finally, in the fifth layer, three clustering algorithms, namely, affinity propagation, K-means, and hierarchical agglomerative clustering, are investigated for clustering of web services based on observed similarities in documents. We demonstrate how each component of the five layers is employed in web services clustering using randomly selected web services documents. We conduct experimental analysis to cluster web services using a collected dataset consisting of web services documents and evaluate their clustering performances. Using a ground truth for evaluation purposes, we observe that clusters built based on the word embedding models performed better than those built using the Bag of Words with Term Frequency–Inverse Document Frequency model. Among the three word embedding models, the pre-trained Word2Vec's skip-gram model reported higher performance in clustering web services. Among the three semantic similarity measures, path-based WordNet similarity reported higher clustering performance. By considering the different word representations models and syntactic and semantic similarity measures, we found that the affinity propagation clustering technique performed better in discovering similarities among Web services.

Keywords: web services clustering; web services discovery; word embedding; clustering; semantic similarity; syntactic similarity

check for updates

Citation: Obidallah, W.J.; Raahemi, B.; Rashideh, W. Multi-Layer Web Services Discovery Using Word Embedding and Clustering Technique. *Data* **2022**, *7*, 57. https://doi.org/10.3390/data7050057

Academic Editor: Panagiotis Karras

Received: 2 April 2022
Accepted: 28 April 2022
Published: 4 May 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Web services are a set of loosely coupled software components that are developed, described, published, discovered, utilized, and reused to achieve particular functionalities [1]. Web services are of different types, and provide various functionalities over the internet. Types of web services include SOAP-based, REST-based, XML-RBC-based, and JSON-RBC-based web services. The most prevalent forms are REST-based and SOAP-based services. Recently, global and local enterprises have leveraged their assets by widely publishing their web services or application programming interfaces (APIs), allowing for scaling of their functionalities over the internet using different business models.

Web service discovery involves finding and locating existing web services based on a service requester's requirements. The discovery process must match services' functional

and non-functional descriptions. Functional requirements refer to the elements and features that indicate the system's capabilities, including interfaces, protocol bindings, and operations. These features are typically described in the service profile and its description. Non-functional requirements refer to the elements and features that indicate the system's performance parameters, including quality of service (QoS) considerations as well as service policies such as service cost and security features [2,3]. QoS considerations can include parameters such as availability, reputation, privacy, price, response time, and usability.

Web services providers publish their newly developed web services in diverse online repositories and portals to allow for their use. Newly developed web services are added to the list of web services, which consists of previously identified groups and categories on the portal. In the absence of automation, the process of adding new web services to the right category within the portal is carried out manually by a domain expert who reviews the web services and categorizes them based on their functionalities. This process requires more time and effort as the number of added web services increases. In a real scenario, a web services aggregator or collector gathers the description of web services along with their details from different repositories and portals to allow users to access them. After the web services descriptions are collected, the web services files need to be indexed and categorized. The process of categorizing collected web services based on their domain (e.g., functional and non-functional similarity) is performed by a domain expert. Artificial intelligence, data and text mining, and NLP techniques can be employed instead of a domain expert to cluster the collected web services into categories (e.g., WSDL, text, etc.) while considering their descriptions. Clustering techniques can automatically cluster Wwb services based on different similarity measures, with similar web services being assigned to the same cluster [4]. Furthermore, having clusters of web services based on their functional or non-functional specifications minimizes the search space for the query matching algorithm.

The large number of web services available over the internet with different formats and functionalities motivated us to develop a web services discovery approach based on clustering similar web services with syntactic- and semantic-based similarities in order to minimize the search space. By clustering related web services, service matchmakers do not need to match user queries against all the service offerings; instead, the matchmaker can match user queries against web services clusters. We propose the use of text and data mining methods to find similarities between web services while considering various word representations, embedding models, and similarity measures.

2. The Proposed Multi-Layer Architecture for Clustering of Web Services

In Figure 1, we identify the components and the characteristics by relating them to the conceptual model and typology of building Web services discovery systems [5]. The storage and location characteristics of Web services discovery systems are demonstrated by a centralized approach in Web services storage. The formalization characteristics of Web services discovery systems are demonstrated through the ability to use both syntax and semantic Web services descriptions. This configuration emphasizes the need for a Web services discovery system that works with all types of Web services description languages. The proposed solution to cluster Web services based on employing different word representation models, similarity measures, and clustering algorithms is part of the matchmaking characteristics. The matchmaker, supported by a matching algorithm, responds to the service requester's query with a list of Web services by referring to the clustered Web services, which is where we propose our solution. The selection characteristics of Web services discovery systems are demonstrated by allowing the user to express the functional and non-functional requirements with the discovery interface. The automation characteristics are demonstrated by allowing the service requester to manually invoke the discovery process.

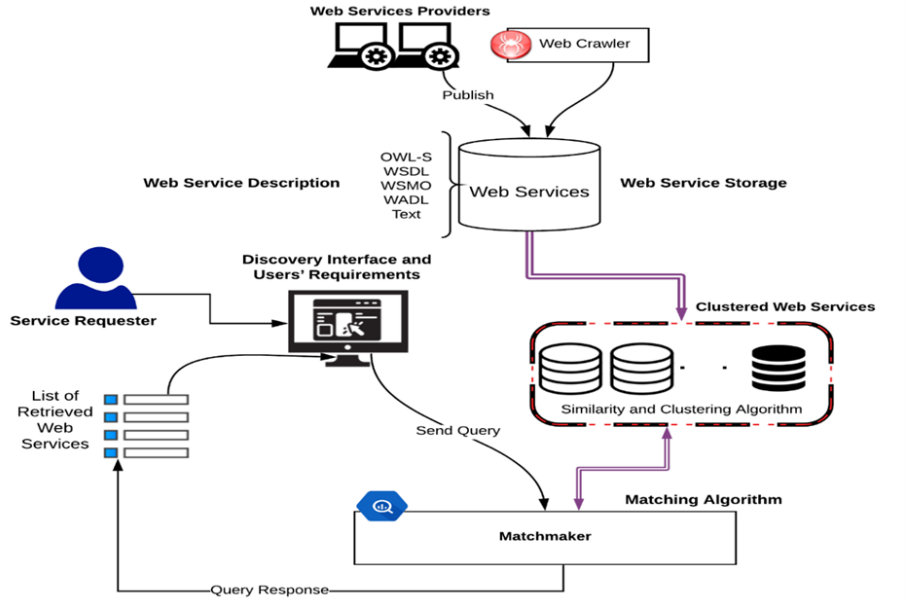

Figure 1. The General Architecture of Web Services Discovery.

The proposed clustering-based multi-layer Web services discovery architecture is demonstrated in Figure 2 based on five layers. It is multi-layer because it consists of various layers, including representation and the embedding models and similarity measures within each layer. The proposed solution uses natural language processing (NLP), text and data mining models, and various similarity measures and clustering techniques to cluster Web services. It should be noted that the focus of the proposed architecture is only on clustering Web services based on their functional requirements, not on matching the query with the final results; i.e., query is not part of the proposed architecture.The proposed architecture includes a stack of five layers, as follows:

1. **Web Services Description and Data Preprocessing**: In this layer, NLP techniques are applied to select and extract terms (features) from Web services description files. This includes several different steps, starting with parsing selected features from the Web services description files, then tokenizing, removing stopwords, and lemmatizing words. Feature selection depends on the type of Web services description documents.
2. **Word Representation, Embedding, and Transformation**: In this layer, two types of NLP models are considered to transform the extracted terms into a computer-understandable and processable form. This includes Bag of Words (BOW) with Term Frequency-Inverse Document Frequency (TFIDF) as a weighted scheme as well as three word-embedding models; these models include two pre-trained models and one self-trained model.
3. **Syntactic Similarity**: Four syntactic similarity measures are implemented in this layer to find the similarity between Web services, namely, Cosine, Euclidean, Minkowski, and Word Mover's distance;
4. **Semantic Similarity**: In this layer, two WordNet-based similarity measures are employed to find similarities between Web services, including path and WUP similarities. Normalized Google Distance (NGD), sometimes called Normalized Web Distance (NWD), is employed to find the semantic similarity between Web services.
5. **Web Services Clustering**: In this layer, affinity propagation (AP), K-means, and hierarchical agglomerative clustering (HAC) are studied and implemented in order to cluster Web services into a certain number of clusters based on their functionalities. Search engines or matchmakers use this step as a precursor by categorizing Web services based on their functionality in Web services sources.

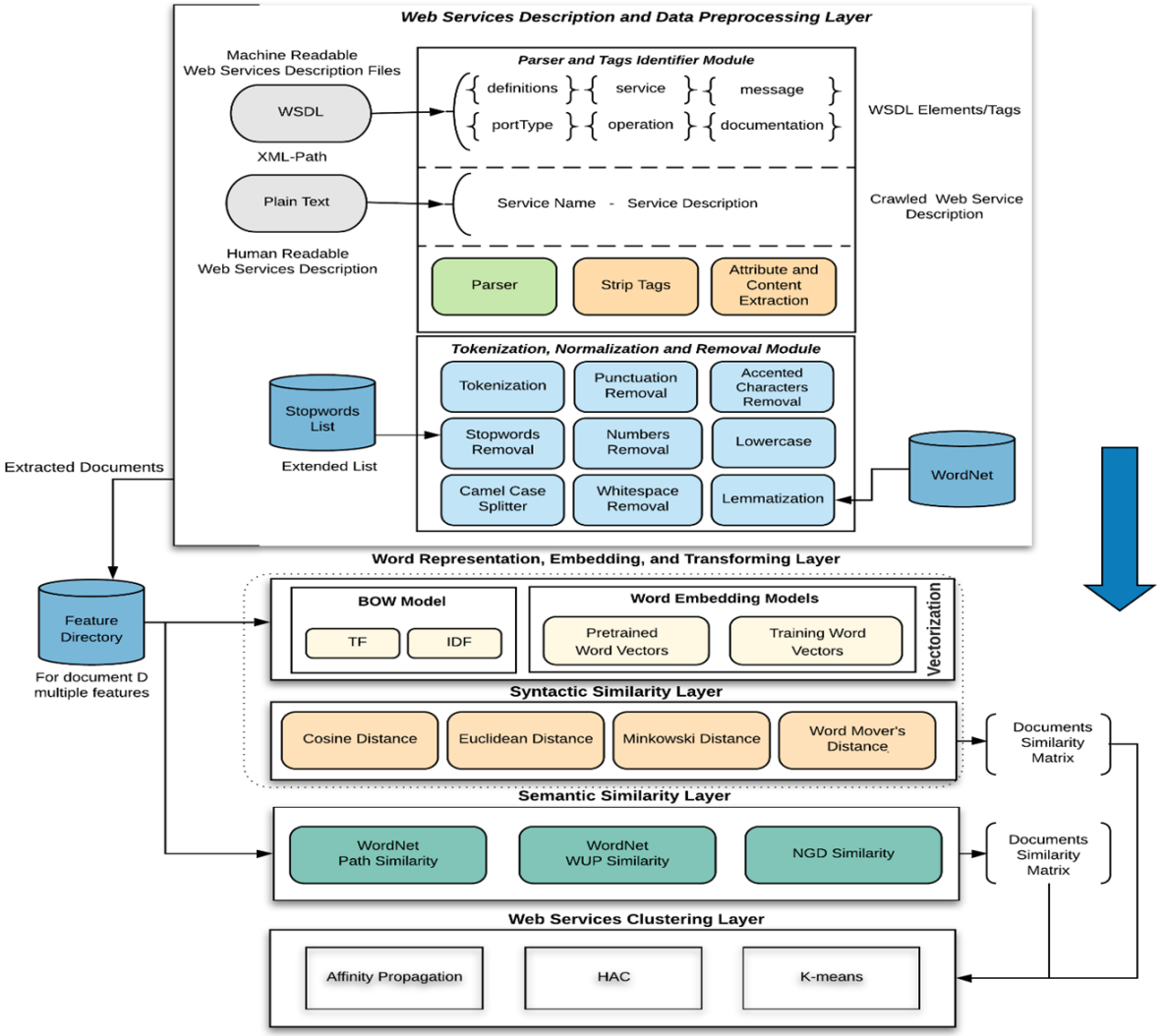

Figure 2. The Proposed Clustering-based Multi-Layer Web Services Discovery Architecture.

2.1. Layer 1: Web Services Description and Data Preprocessing

The first layer is intended to prepare Web services description files for the mining process. Web services have different types of formalization based on the markup language or other description languages utilized to describe Web services, which can be either syntax-based or semantic-based description languages [6]. This layer consists of two modules, namely, the parser and tags identifier module and the tokenization, normalization, and removal module. The parser and tags identifier module relies on the type of Web services description language. The features (i.e., tags or terms) are identified, including their attributes and the content to be selected and extracted. The tokenization, normalization, and removal module applies several operations to the extracted text in order to prepare it for the mining process. Figure 3 illustrates the four main steps in Web services data preprocessing. The collected Web services data include WSDL, which is a machine-readable format, and a textual, human-readable Web services description. The proposed solution uses text mining and various similarity calculations to cluster Web services; this makes the solution applicable to any type of Web services description, such as WADL or OWL-S. The key is to parse and identify from the Web services files the related data which best help in the process of Web services clustering. This layer works to identify and select the most important and useful features (terms), extract them from any Web services description files, and then apply NLP preprocessing techniques.

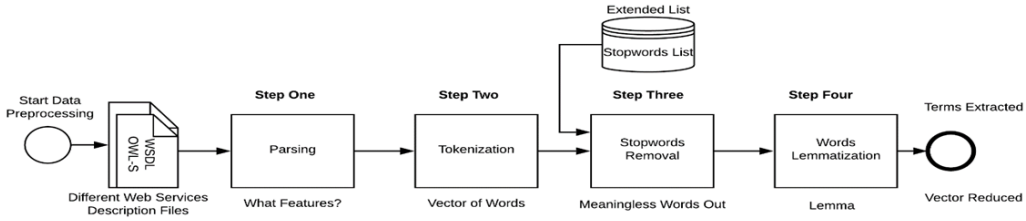

Figure 3. Data Preprocessing Steps.

The parsing step focuses on identifying which data and features are to be extracted from the Web services description file. A parser is employed to extract the required features and terms from the Web services description in cases where the extracted features are different based on different considerations and strategies. For WSDL, a variety of XML-based parsers are available to provide such functions, including Beautiful Soup [7], a Python XML-based parser utilized here to extract WSDL elements that contain meaningful and useful information. The first step depends on identifying useful information in Web services descriptions, which can then be parsed and extracted for use in the mining processes. The collected Web services description dataset is based on WSDL. However, this is an implementation detail; any format of Web services description language or documentation can be selected for this purpose. We identified the WSDL elements Definition, Message, PortType, and Service as well as their attributes, including all Names, Parts, Operations, and the Documentation contents to be extracted.

The tokenization step splits the text into a string (sequence of characters) and sub-units called tokens to identify the boundaries and breaks between words [3]. To extract tokens from Web services description, we include the sequence of uppercase and lowercase letters (e.g., GetAddress to "get" "address") based on camel case splitting and the sequence of words with symbols in between (e.g., money-amount-transfer to "money", "amount", "transfer") based on punctuation removal. All words are transformed to lowercase, and numbers, and whitespace and accent marks are removed. The result of this step is a vector of words extracted from the Web services description which includes meaningful parts (e.g., words) while discarding other meaningless chunks (e.g., whitespace).

The Stopwords removal step is thought to improve the performance of clustering by eliminating words such as 'the', 'is', 'at', 'which', and 'on' for dimensionality reduction. The list of stopwords can be created based on sorting the terms in Web services document collection by frequency of occurrence, then titling the number of high-frequency terms as stopwords [8]. We employed the publicly available NLTK [9] stopwords list to eliminate very frequent and rarely useful words from Web services-extracted text. We then updated this list by adding and removing words based on the WSDL specification and its elements. For example, words such as "http", "get", "post", "soap", "cfc", "service", "webservice", and "port" were added to the list of stopwords. The extended stopwords list contains 213 words.

The WordNet lemmatization step reduces words into their root form, known as lemma, in order to save time and memory space. In natural language processing (NLP) both stemming and lemmatizing can be utilized for this purpose, which represents a special case of normalization [10]. The main difference between the two processes is that stemming is based on rules which trim word beginnings and endings. In contrast, lemmatization uses more complex morphological analysis and dictionaries. A stemmed word might not be an actual word (i.e., nonexistent words), whereas a lemma is always an actual word. We chose to apply WordNet lemmatization to all extracted words instead of Porter stemmer. As we consider different similarity measures that require a dictionary and hierarchy-based measures, lemmatization is more suitable. The reason for this is that while we can find and locate a lemma in a dictionary, the root stem may not always be available.

This layer concerns the selection and extraction of features (terms) from the Web services description files. The process of selecting features is based on their usefulness

and meaningfulness in differentiating between the various Web services. This is important in the process of measuring the similarities between Web services, as similar features indicate similar or related Web services. The extracted and preprocessed terms are stored in the feature directory for future similarity and clustering analysis. The feature directory contains all the extracted words from the Web services document collection, which are stored in a way that facilitates future text mining and analysis tasks, allowing simple or more complicated analyses to be run with different techniques and models.

2.2. Layer 2: Word Representation, Embedding, and Transforming

In order to convert the unstructured text extracted from the Web services documents into a format acceptable to algorithms. We convert these documents to vectors of words, a numerical array in which values are specific weights for each word that could be its frequency, its occurrence, or various other representations. For a Web service document (WSd),

$$WSd = wsd_i, i = 1, 2, 3, \ldots n \tag{1}$$

where wsd_i represents a Web service document and n is the number of the Web service documents. The vector space model (VSM) [11] is employed to represent each wsd_i of WSd as a numerical value or point ns_i in the numerical space (NS). In the VSM, each Web service document wsd_i is represented as a vector of term/feature weights as:

$$wsd_i = t_{1i}, t_{2i}, t_{3i}, \ldots t_{ji} \tag{2}$$

where t_{ji} represents a real number indicating the weight and importance of the term j in the Web service document i. The process of vectorization converts textual information into numbers.

Bag-of-Words (BOW) with the Term Frequency-Inverse Document Frequency (TFIDF) term weighting scheme is utilized in part of this layer to represent the extracted Web services features from the feature directory. Furthermore, word embedding, which is the new state-of-the-art approach in NLP, is employed to represent the words extracted from the Web services documents. Word embedding provides a dense vector representation of a word based on its context [12]. Word embedding models focus on representing the idea of a word (looking to the context) by a vector. On the other hand, BOW with TFIDF focuses on representing a word (looking to the frequency) as a vector.

TFIDF uses real values to capture the term distribution among Web services documents in the collection in order to assign a weight to each term in every member Web services document. The TFIDF perception is that the more times a term occurs in a Web service document, the more important this term is to this Web service document. Consequently, the assigned weight increases corresponding to the number of times (frequency of occurrence) this term appears in the document. On the contrary, a term that appears in many Web service documents in the collection will be penalized by assigning a lower weight to it [13]. The following are the steps and equations for calculating BOW with TFIDF.

Term Frequency (TF): measures the number of times a term occurs in a Web service document.

$$TF_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \tag{3}$$

Inverse Document Frequency (IDF): measures the importance of a term by considering all documents.

$$IDF_i = \log \frac{|\, Wsd \,|}{|\, \{j : t_i \epsilon wsd_j\} \,|} \tag{4}$$

Term Frequency-Inverse Document Frequency (TFIDF): for a term (t_i) in an extracted word vector of a document (wsd_i)

$$TFIDF_{i,j} = TF_{i,j} \times IDF_i \tag{5}$$

where $n_{i,j}$ represents the number of occurrences of a term (t_i) in a Web service document (wsdj), $|Wsd|$ is the total number of Web services documents in the collection, and $|\{j : t_i \epsilon wsd_j\}|$ represents the number of Web service documents where the term t_i occurs. After calculating the TFIDF for all terms in the Web services documents, it can be represented by an m by n matrix $A \epsilon R^{m \times n}$, where n represents the number of unique terms occurring in the Web service documents and m represents the number of Web service documents in the collection.

Because measuring the similarity between Web services documents with TFIDF depends on the word overlap [14], the probability of having common words decreases, especially in the present case where the length of the extracted text is relatively short. Limitations of TFIDF include ignoring synonyms, any semantic relatedness, and the correlation between words in the process of text representation [15]. However, BOW with the TFIDF term weighting scheme remains one of the most frequently cited text representations [13]. To minimize these limitations, we consider word embedding for representation learning as an additional and supporting step in finding the similarities between Web services.

Word embedding models such as Word2Vec, FastText, and GloVe [12] provide a dense vector representation of words that capture words' meaning based on context. Based on the generated vectors, words are placed in such a way that words having similar meanings appear together and dissimilar words are placed far away from each other. We incorporated word embedding to represent extracted words from the Web services documents for several reasons. First, word embedding models are capable of representing each extracted word as a dense vector instead of one number, as is the case with the BOW and TFIDF model. Second, the word embedding representation allows for adding the word's semantic meaning by considering its context. Third, we use word embedding for text enrichment, as we have a limited amount of extracted text from the Web service documents. We use two approaches: a pre-trained word embedding model and a self-trained word embedding model trained based on a specific domain and on corpora we created and collected.

Pre-trained word embedding models are a set of word vectors that have been created and trained, usually on a general-purpose corpus such as Wikipedia [16] and English Gigaword [17]. The first employed word embedding model is based on training the Word2Vec-based skip-gram model on text from English Wikipedia. The corpus has 3.5 billion tokens and knows 249,212 English words. The preprocessing step was applied to the corpus before training, which includes splitting it into sentences, tokenizing, lemmatizing, part of speech (POS) tagging, and removing stopwords. The second employed word embedding model is based on training a GloVe model on the Common Crawl corpus [18]. The result of the learning process, which is the representation of words (aka word embedding) depends on the corpus utilized in the learning process. One drawback of pre-trained models is that when working on an application or a specific domain, the result of the model will not be optimal due to the generality of the word embedding.

We trained a word embedding model with the Word2Vec skip-gram model based on a collected corpus in a domain-specific area related to Web services, which fit this problem-specific domain. The corpus was collected from the description of Web services from different Web services repositories and portals such as PW and GitHub. The Web services documents include both a human-readable description and a machine-readable description of Web services. The preprocessing step was applied to the corpus before training, including tokenizing, lemmatizing, and removing stopwords and numbers. Table 1 illustrates the details of the pre-trained and self-trained models.

Table 1. Word Embedding Models Details.

Name	Algorithm	Vector Size	Window	Vocabulary Size
English Wikipedia	Word2Vec	300	3	249,212
ENC3 Common Crawl	GloVe	300	10	2,000,000
Web Services Documents	Word2Vec	50	3	5563

The process of training the Word2Vec model involves the following steps:

1. Preprocessing and conversion of text into a suitable format for training: the collected Web services corpus include 918 Web services documents utilized for training purposes. The documents need to go through preprocessing steps, which include tokenization and removal of accent marks, stopwords, punctuation, and white space. Then, the lowercase and lemmatization steps are applied. Training a Word2Vec model requires that every Web service document is represented as a list and every list contain a list of tokens/words in that document. This requirement leads to the conversion of all text into a list of lists for each token in the collected corpus.
2. Choosing the right Word2Vec models: the choice of the right training algorithm is a task-specific decision [19]. The Word2Vec algorithm has two models for representing words as a dense vector, the CBOW (Continuous Bag of Words) and skip-gram models [20]. Both models use shallow neural networks to map words to the target variable considering the context. We selected the skip-gram architecture to train our model based on the fact that we had a limited amount of text for training purposes. According to [19–21], the process of learning is faster for CBOW than for skip-gram, which takes a long time. Skip-gram works better for a small amount of training data due to its ability better represent rare words. As we had a small amount of text for training purposes, skip-gram was a better choice to train our model based on the collected corpus, despite the added time and resource restrictions.
3. Setting up the learning hyperparameter configurations. We adjusted the following hyperparameters:
    - Size: each word or token is represented as a dense vector, and the size of the vector is significant in the learning process. As we had limited data, the size of the vector was set to a smaller value. This was because there are only a few unique neighbors for any given word. We set the size of the vector as 50.
    - Window: this is related to the maximum distance between the target word and its surrounding words (neighbors). The value of the window indicates the number of neighbors to the left and right of the target word. As such, considering our limited data a smaller window can result in words that are more closely related [20]. We selected value three as the window size for training the model.
    - Minimum count: this is related to the frequency count of words, with uncommon words usually being unimportant. As our data had a small amount of text, we set the minimum count as one to allow all words to be considered when training the model.

2.3. Layer 3: Syntactic Similarity

The distance between two Web services documents' vectors can be utilized to show how similar or dissimilar the two documents are. For syntactic similarity, we use Cosine distance to measure the similarity between Web services (vector of words) in the vector space model. Furthermore, Euclidean distance and Minkowski distance are employed to measure the distance between Web services by utilizing the generated Web services' vectors. The Word Mover's Distance (WMD), which is a specific distance metric for word embedding, is utilized as well. In order to find the similarities between Web services, we process the TFDIF document-term matrix and document embedding matrix. The outputs of this layer are different document similarity matrices based on four different distance measures.

Cosine distance measures the similarity or distance between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors projected in a multidimensional space, and determines whether two vectors are pointing in roughly the same direction [22]. Euclidean distance is a measure of the actual straight line distance between two points or vectors in Euclidean space. Minkowski distance is a generalization of the Euclidean and Manhattan distances for calculating distance similarity between two points or vectors in the normed vector space [22]. The word vectors extracted from Web services documents are converted into numbers with weights, making them ready to apply

the similarity measure; the similarity can then be calculated. The vector representations of Web services documents are utilized in measuring the similarity and distance. To measure the syntactic similarity/distance between Web services documents, we use

$$Cosin(\overrightarrow{x}, \overrightarrow{y}) = \frac{\overrightarrow{x}.\overrightarrow{y}}{\mid \overrightarrow{x} \mid \mid \overrightarrow{y} \mid} = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2 \sum_{i=1}^{n} y_i^2}} \quad (6)$$

$$EuclideanD(\overrightarrow{x}, \overrightarrow{y}) = \| \overrightarrow{x} - \overrightarrow{y} \| = \sqrt{(x_1 - y_1)^2 + ... + (x_n - y_n)^2} \quad (7)$$

$$MinkowskiD(\overrightarrow{x}, \overrightarrow{y}) = (\sum_{i=1}^{n} \mid x_i - y_i \mid)^{\frac{1}{p}} = \sqrt[p]{\mid x_1 - y_1 \mid^p + ... + \mid x_n - y_n \mid^p} \quad (8)$$

where $\overrightarrow{x}, \overrightarrow{y}$ refer to the word vectors and n refers to the number of attributes. The result is a document–document matrix showing similarity/distance between Web services documents.

Word Mover's Distance (WMD) has been proposed by [23] as a distance function between text documents based on word embedding, which shows that the distance between embedded word vectors is semantically related. WMD measures the dissimilarity between two documents based on the minimum amount of distance that an embedding word in one document needs to travel to reach another embedding word in the other document. The word embedding in each Web service document is utilized to find the distance between other word embedding belonging to other Web services documents. Based on the provided word embedding, WMD works by generating a normalized Bag of Words (nBow) and calculating word travel cost, which is the distance between words based on Euclidean distance. Finally, document distance is calculated as the minimum (weighted) cumulative cost required to move all words from one document to another.

2.4. Layer 4: Semantic Similarity

Words extracted from the Web services documents are consumed in this layer without transforming the text into numbers with the VSM. In this layer, we consider knowledge-based approaches based on WordNet [24] and Normalized Google Distance (NGD) [25,26] to find the similarities between Web services.

WordNet is an extensive lexical database of English which includes nouns, verbs, adjectives, and adverbs. Words are grouped into sets of cognitive synonyms called synsets, each expressing a distinct concept, employed to find the relations between different synsets [24]. WordNet-based similarities depend on the lexical semantics, sense relations, and additional information to identify the similarities between words. Each synset in WordNet is labeled with three parts, namely, the word, part of speech (POS), and index of the word's sense. For example, the synset "service.n.01" indicates the first meaning of "service" as a noun. WordNet's structure makes it a useful tool for computational linguistics and natural language processing (NLP). To measure the similarities between Web services, we utilize two WordNet-based similarities, the path and Wu–Palmer [27] similarities

Path similarity measures the similarity of two-word senses based on the shortest path that connects the senses in the WordNet hypernym and hyponym taxonomy. Two words in the Web service vector of words are similar to each other if they are near each other in the WordNet hierarchy. Wu–Palmer (WUP) semantic similarity measures the similarity and relatedness between two-word senses based on the path length between their senses located in the taxonomy along with the depth of their Least Common Subsumer (LCS), which is the closest ancestor to both senses. The returned score denotes how similar and related two-word senses are, ranging from 0 to 1, where 0 is least similar and 1 is identical. In order to measure the similarity between two words' synsets, we use

$$PathSim(s_1, s_2) = \frac{1}{EdgeDistance + 1} \quad (9)$$

$$WUPSim(s_1, s_2) = \frac{2 * depth(LCS)}{depth(s_1) + depth(s_2)} \quad (10)$$

The process of generating the Web service documents' similarity matrix is not straightforward; several processes are needed, as the WordNet similarity measures are based on the words' similarity. Figure 4 illustrates the pseudocode for generating the semantic similarity matrix between Web services documents with both WordNet's path and WUP.

Input: Web services Documents
Output: Semantic Similarity Matrix between Web services

FOR each Web service document:
    FOR words in a Web service document:
        Tokenize each word
        Get NLTK POS tagging
        Convert NLTK's POS tag into WordNet tags
        Get the first synset from WordNet
    END FOR
END FOR
Output: list of synsets for all words in Web service documents
To generate the pairwise similarity matrix between Web service documents.
FOR each Web service documents:
To calculate normalized similarity score between pair of Web service documents using the list of synset based on Path and WUP similarity metric.
    FOR each synset in Web service document 1:
        Get similarity score to synset in Web service document 2
        Sum all largest score
        Divide the sum by the number of synsets
    END FOR
    Output 1: similarity score document 1 to document 2
    FOR each synset in Web service document 2
        Get similarity score to synset in Web service document 1
        Sum all largest score
        Divide the sum by the number of synsets
    END FOR
    Output 2: similarity score document 2 to document 1
    Take the sum of Output 1 and Output 2, then divide by 2 for averaging.
    Output 3: normalized similarity score between a pair of Web services.
END FOR
Output: pairwise semantic similarity matrix between Web service documents generated based on Output 3.

Figure 4. Pseudocode for generating Similarity Matrix between Web Services Documents.

Normalized Google Distance (NGD) is a semantic similarity proposed by [26] which measures the semantic distance between words based on the number of search hits returned by search engines for a given set of words. Words with the same, similar, or related meanings in the natural language sense tend to be close in terms of NGD units. On the other hand, words with dissimilar meanings tend to be farther apart. We utilized NGD to measure the similarity between words in the Web services documents to obtain a similarity matrix between the words in two Web services. Search engines such as Google or Bing work for NGD (i.e., Normalized Web Distance) as the frequency count extractor, and the corpus is the Web NGD between two words w_i, w_j calculated with

$$NGD(w_i, w_j) = \frac{\max\{\log f(w_i), \log f(w_j)\} - \log f(w_i, w_j)}{\log N - \min\{\log f(w_i), \log f(w_j)\}} \quad (11)$$

where $f(w_i)$ is the number of hits returned for the first word w_i, $f(w_j)$ is the number of hits returned for the second word w_j, and $f(w_i, w_j)$ is the number of hits returned for both w_i and w_j. N is the total number of pages indexed by the search engine; if $NGD(w_i, w_j) = 0$, then the two words are as alike as possible; however, if $NGD(w_i, w_j) \geq 1$, then the two

words are very different. The similarity between two words in Web service documents based on NGD is

$$NGDSim(w_i, w_j) = 1 - NGD(w_i, w_j) \tag{12}$$

2.5. Layer 5: Web Services Clustering

Three clustering methods are studied and implemented to cluster Web services based on the observed similarity and dissimilarity matrices. The first method uses the affinity propagation (AP) clustering algorithm [28]. The second method uses a partition-based clustering method where K-means clustering is employed to cluster Web services. The third method uses a hierarchical-based clustering method where hierarchical agglomerative clustering (HAC) is employed to cluster Web services.

The affinity propagation (AP) clustering algorithm can find the number of clusters simultaneously and automatically based on the similarity matrix (S) received as an input. Web services are treated as data points and sending messages between all Web services can find the exemplars, which are the most significant Web services to their clusters. These messages are stored in responsibility and availability matrices. Three matrices are needed to cluster Web services based on AP. The first (input) matrix consists of the Web services' generated similarity matrix (S), where similar Web services have larger values. The second (initialized) matrix is the responsibility matrix $r(w_i, w_j)$, where the similarities between two Web services are utilized to calculate the responsibility for Web service w_i to be the cluster center (exemplar) for another Web service w_i. The third (initialized) matrix is the availability matrix $a(w_i, w_j)$ to calculate how appropriate it is for a Web service w_i to select a Web service w_j as its cluster center. The number of clusters is influenced by the preference values $S(w_i, w_j)$ and the messaging procedures defined in the $r(w_i, w_j)$ and $a(w_i, w_j)$ calculations [28]. Both responsibility and availability matrices are initialized to all zeroes and then updated with the input from the similarity matrix; for example:

$$r(w_i, w_j) \leftarrow S(w_i, w_j) \max_{w'_j \neq w_j} \left\{ a(w_i, w'_j) + S(w_i, w'_j) \right\} \tag{13}$$

$$a(w_i, w_j) \leftarrow \min \left\{ 0, r(wj, w_j) + \sum_{w'_i \notin (w_i, w_j)} \max(0, r(w'_i, w_j)) \right\} \tag{14}$$

$$a(w_j, w_j) \leftarrow \sum_{w' \neq w_j} \max(0, r(w'_i, w_j)) \tag{15}$$

The agglomerative clustering algorithm is utilized for clustering Web services based on the observed distance matrix. Hierarchical agglomerative clustering (HAC) is a bottom-up clustering method that starts assigning each Web service to its own cluster (singleton cluster) [29], then iteratively finds the most similar pair of clusters and merges them into a single cluster until the stop conditions are met or until all clusters have been merged into a single cluster that contains all Web services documents. K-means [30] is a partition-based clustering algorithm utilized to organize Web services documents into clusters. The first step is initialization, which starts by identifying the number of Web services clusters (k) to be found. The second step is choosing k random points as the initial clusters' centers. The third step is to assign Web services to their nearest cluster center. The fourth step is to update the cluster centers by replacing them with the mean of the coordinates of all Web services assigned to that cluster. The third and fourth steps are then repeated until the clusters converge. The final clustering result depends on the first and second steps, where we need to select the best number of clusters to find and to select the initial centroids. However, in the present case we already have the number of clusters k as part of the ground truth.

3. Performance Evaluation and Analysis

Most studies [4] in the area of Web services discovery systems rely on different evaluation methods to determine the effectiveness of the proposed solutions. We follow the evaluation methods and metrics used in similar research as specified in [4] to measure the performance of Web services clustering. An offline evaluation is usually employed to measure how the proposed solution facilitates Web services discovery. With its different models and methods, the proposed architecture facilitates the process of Web services discovery by focusing on Web services clustering based on different similarity measures, algorithms, and considerations prior to the matchmaking steps. We use offline experiments to cluster Web services considering a collected dataset that contains Web services description files to measure the clustering performance based on the clustering algorithms.

Our performance evaluation strategy is to find the agreements and disagreements between the ground truth and the Web services clusters resulting from the proposed architecture. This requires a ground truth that specifies each Web service classification based on their similarity, primarily employing human labeling for this task. This strategy is the most widely employed for evaluating Web services clustering performance, with accuracy, recall, precision, and F-measure evaluation metrics being used [4]. We need to find the best cluster-to-class association between the resulting clusters and the ground truth classes in this strategy. The proposed architecture contains various models, similarity measures, and three clustering algorithms. We measure Web services clustering performance based on the different models and similarity measures by considering the selected clustering algorithm.

3.1. Datasets

One of the critical challenges in clustering Web services considering the proposed architecture is to find publicly available Web services description files that provide balanced and less sparse data. As the proposed architecture is configured to work with Web services description files based on text and WSDL, we scraped Web service documents, including text and WSDL files, from PW and GitHub. WSDL files represent a machine-readable format, and text represents a human-readable format. In general, the Web services description format can be in any format and is not restricted to WSDL and text. As stated in layer one of the proposed architecture, the parser and tags identifier module identifies the required text (features) to be extracted from the Web services descriptions for the mining process.

The initially collected Web services dataset contains 459 human-readable files and 459 machine-readable (WSDL) files. We reviewed the Web service documents to identify any duplication and invalid WSDL formats. The overall collected Web services dataset was employed to train the Word2Vec model as part of the self-trained word embedding model used in layer two. We selected 102 Web service documents to evaluate the performance of clustering based on the proposed architecture. We manually labeled the 102 Web services into different domains and classes based on their functionality to serve as ground truth. The generated ground truth was utilized to measure Web services clustering performance by comparing the ground truth with the resulting clusters (agreements and disagreements) using external evaluation methods.

3.2. Experimental Environment

The experiments were conducted on Microsoft Windows 10 with an Intel Core i7-10510U 1.8 GHz CPU and 16 GB of RAM. Python was employed as the programming language. Various Python-based data mining and text mining libraries were utilized in the implementation of the proposed architecture. The libraries utilized included NLTK [9], Gensim [31], and scikit-learn [32]. Modifications to the built-in functions were occasionally needed to control the input and the output of each layer of the proposed architecture.

3.3. Evaluation Metrics

We followed the external evaluation strategy by employing accuracy, precision, recall, and F-measure based on the developed ground truth. Accuracy measures the overall

performance of a model and is calculated as the number of correct predictions (agreements) divided by the total number of predictions. Precision measures exactness and quality by showing how many Web services were predicted correctly out of the ones that were predicted as belonging to a given cluster. Recall measures the completeness, correctness, and quantity by showing how many Web services were predicted correctly out of the ones that should have been predicted as belonging to a given cluster. F-measure is a harmonic mean of precision and recall which provides a better indicator of clustering algorithm performance by considering both misplaced and missed Web services. The evaluation metrics can be calculated using

$$\begin{aligned} Accuracy(C_i, CL_j) &= \frac{\text{number of members of } C_i \text{ in } CL_j}{\text{total number of members}} \\ &= \frac{TP + TN}{TP + TN + FP + FN} \end{aligned} \tag{16}$$

$$\begin{aligned} Precision(C_i, CL_j) &= \frac{\text{number of members of } C_i \text{ in } CL_j}{\text{number of members in } CL_j} \\ &= \frac{\text{Successfully } (CL_j)}{\text{Successfully}(CL_j) + Misplaced(CL_j)} \end{aligned} \tag{17}$$

$$\begin{aligned} Recall(C_i, CL_j) &= \frac{\text{number of members of } C_i \text{ in } CL_j}{\text{number of members } C_i} \\ &= \frac{\text{Successfully } (CL_j)}{\text{Successfully}(CL_j) + Missed(CL_j)} \end{aligned} \tag{18}$$

$$F\text{-}measure(C_i, CL_j) = 2 \cdot \frac{Precision(C_i, CL_j) \times Recall(C_i, CL_j)}{Precision(C_i, CL_j) + Recall(C_i, CL_j)} \tag{19}$$

where C_i represents the class and CL_j is the cluster.

3.4. Experiments and Discussions

We ran five experiments to demonstrate how the proposed artifacts of Web services clustering can provide improvement by employing various word representation models and syntactic and semantic similarity measures in the process of Web services discovery.

Experiment I aimed to measure clustering performance when the BOW with TFIDF model is employed in the word representation, embedding, and transformation layer. We measured clustering performance considering syntactic similarity measures and clustering algorithms. The results indicate that among the tested clustering algorithms, AP provided better performance in clustering Web services. It shows greater agreement with the ground truth, as illustrated in Figure 5. With Cosine, Euclidean, and Minkowski, almost similar accuracy and F-measure scores were reported for AP clustering. Both K-means and HAC performed poorly in clustering Web services when compared to ground truth. K-means performed better than HAC based on two of the three syntactic similarity measures.

Experiment II aimed to measure the performance of clustering when the pre-trained Word2Vec model was employed. Four syntactic similarity measures were employed to cluster Web services with AP, K-means, and HAC. The results indicate that AP performed better in clustering Web services based on syntactic similarities methods. Furthermore, considering a specialized similarity measure for dense vector representations, such as Word Mover's Distance (WMD), Web service clustering performance improved compared to Cosine, Euclidean, and Minkowski similarity measures. In Figure 6, we report the accuracy and F-measure performance metrics.

Experiment III aimed to measure clustering performance when the pre-trained GloVe model was employed. Four syntactic similarity measures were employed to cluster Web services with AP, K-means, and HAC. In this experiment, AP performed better in clustering Web services based on syntactic similarities measures. By considering WMD, Web service

clustering performance improved compared to Cosine, Euclidean, and Minkowski similarity measures. We report accuracy and F-measure metrics for the performance evaluation in Figure 7.

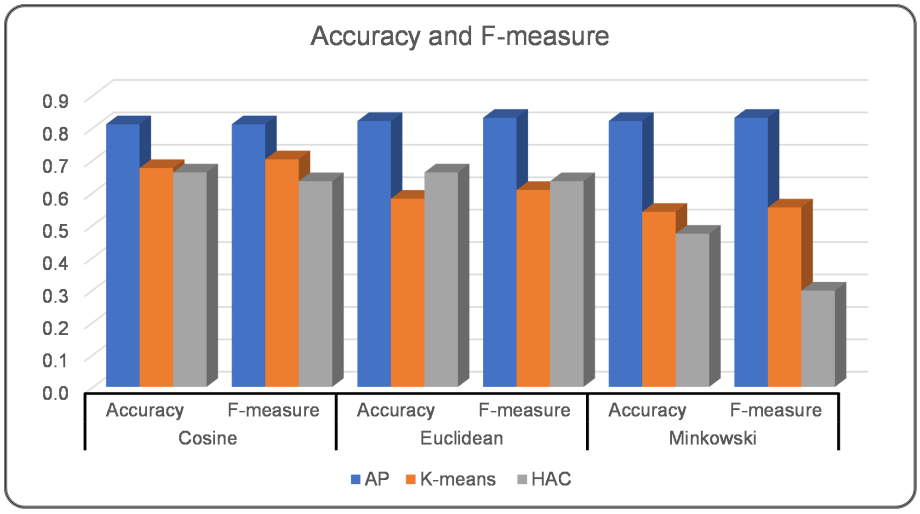

Figure 5. Accuracy and F-measure for Experiment I (BOW with TFIDF Model).

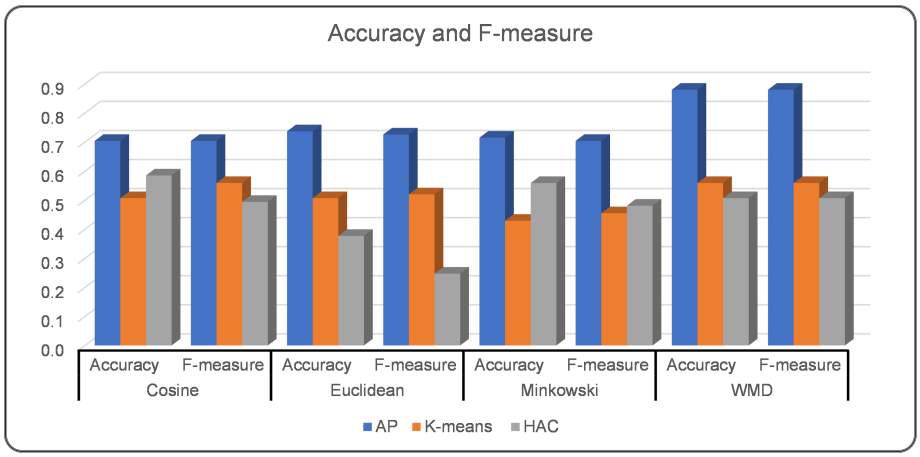

Figure 6. Accuracy and F-measure for Experiment II (Pre-trained Word2Vec Model).

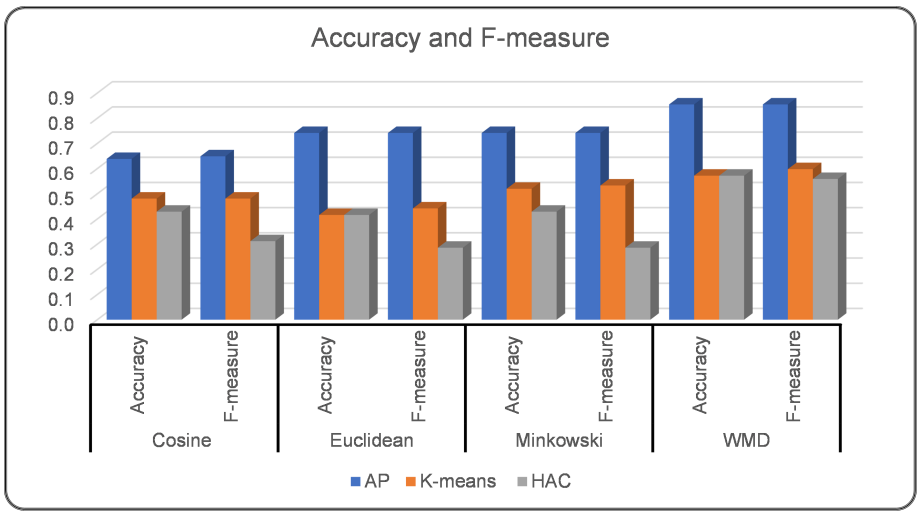

Figure 7. Accuracy and F-measure for Experiment III (Pre-trained GloVe Model).

Experiment IV aimed to measure Web services clustering performance when the self-trained Word2Vec model was employed. In this experiment, four syntactic similarity measures were employed to cluster Web services by employing AP, K-means, and HAC clustering algorithms. Based on the reported performance metric, as illustrated in Figure 8, AP performed better in clustering Web services based on the syntactic similarity measures. Furthermore, Web services clustering performance improved compared to the other similarity measures when WMD was employed. K-means performed better than HAC across all syntactic similarity measures.

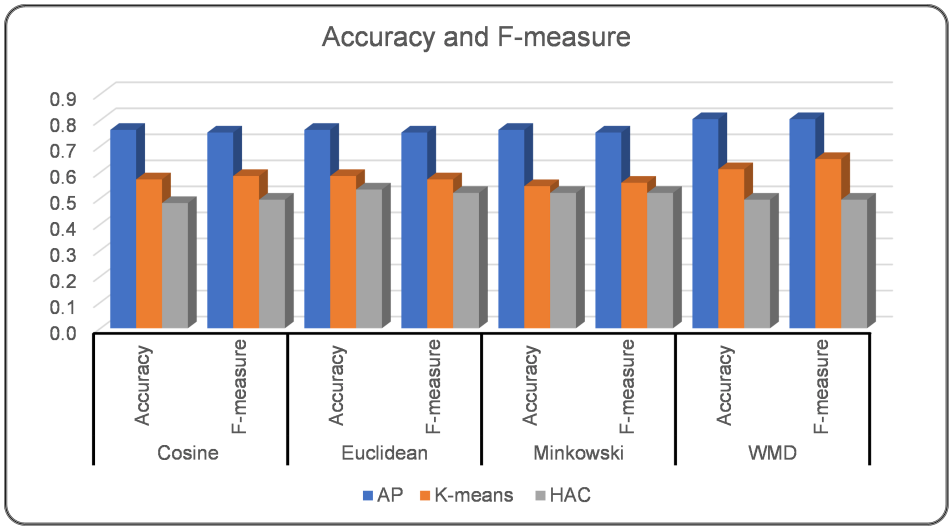

Figure 8. Accuracy and F-measure for Experiment IV (Self-trained Word2Vec Model).

Experiment V aimed to measure the performance of clustering when semantic similarity measures were employed. WordNet's Path and WUP and NGD were employed as semantic similarity measures to cluster Web services with AP, K-means, and HAC. The results indicate that AP performs better in clustering Web services based on the employed semantic similarities, while WordNet Path similarity provides a better similarity measure to cluster Web services among the three semantic similarity measures. In Figure 9, we report accuracy and F-measure to evaluate the clustering performance.

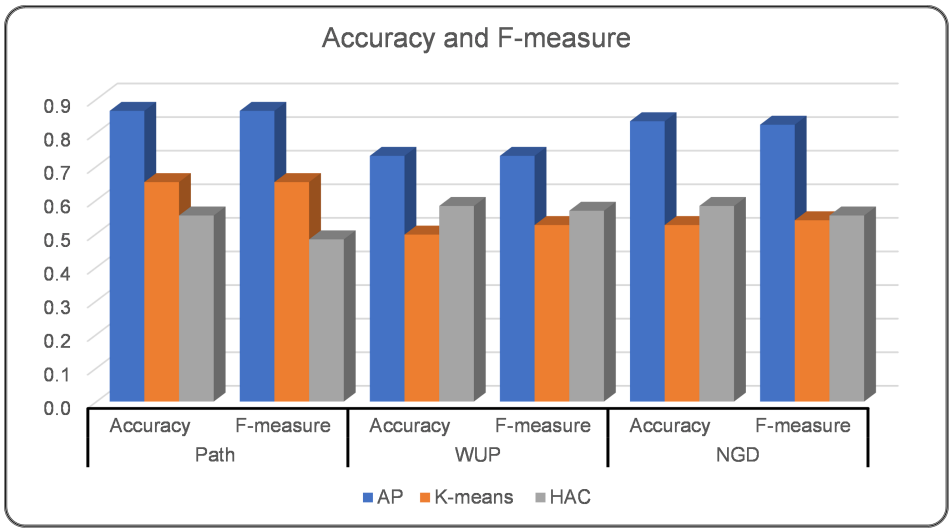

Figure 9. Accuracy and F-measure for Experiment V (WordNet and NGD).

Based on the conducted experiments, we demonstrate that the clustering of Web services is best aligned with the ground truth when the AP clustering algorithm is employed,

showing high accuracy and F-measure. Table 2 provides additional details about clustering performance among the employed word representation, embedding, and transformation models, syntactic and semantic similarity measures, and clustering algorithms. To provide a summary of the conducted experiments, AP with the Euclidean similarity measure reported the best performance considering the various syntactic similarity measures, referring to experiment I in Figure 10. In experiment II, AP with WMD reported the best performance, with high accuracy, precision (minimized false positive), recall (minimized false negative), and F-measure. In experiment III and experiment IV, considering WMD with AP improved clustering performance. In experiment V, employing WordNet's path similarity reported the best performance for AP clustering. Word embedding models reported the best result in clustering Web services when specialized similarity such as WMD was employed.

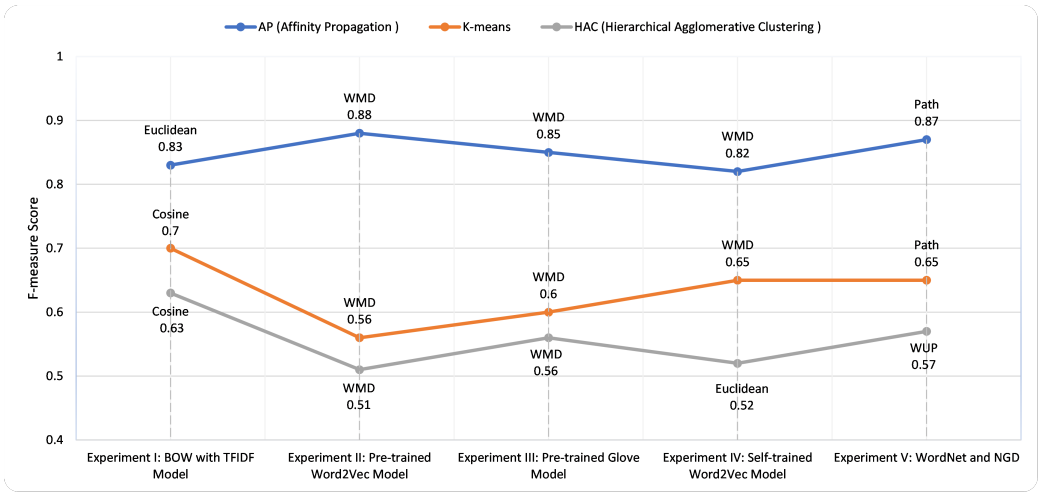

Figure 10. The Best F-measure scores from five experiments using different clustering techniques and similarity measures: WMD, Word Mover's Distance; WUP, Wu–Palmer Semantic Similarity.

While the AP clustering algorithm has not been studied in previous research on clustering of Web services documents, as indicated in [4], we achieved good performance with it in clustering Web services compared to other clustering algorithms such as K-means and HAC. Based on the same datasets and configurations, as part of the proposed multi-layer data mining architecture we reported the performance of K-means and HAC in clustering Web service documents.

Table 2. Performance Evaluation for Clustering Web services (WUP (Wu–Palmer Semantic Similarity), WMD (Word Mover's Distance), and NGD (Normalized Google Distance)).

Clustering Algorithms	**AP**				**HAC**				**K-Means**			
Syntactic Similarity	**Cosine**	**Euclidean**	**Minkowski**	**WMD**	**Cosine**	**Euclidean**	**Minkowski**	**WMD**	**Cosine**	**Euclidean**	**Minkowski**	**WMD**
First Experiment: BOW with TFIDF Model												
Accuracy	**0.81**	**0.82**	**0.82**		0.66	0.66	0.47		0.68	0.58	0.54	
Precision	0.83	**0.84**	0.84		0.81	0.81	**0.89**		**0.86**	0.77	0.65	
Recall	**0.81**	**0.82**	**0.82**		0.66	0.66	0.46		0.68	0.58	0.54	
F-measure	**0.81**	**0.83**	**0.83**		0.63	0.63	0.30		0.70	0.61	0.55	
Second Experiment: Pre-trained Word2Vec Model												
Accuracy	**0.70**	**0.74**	**0.72**	**|0.88|**	0.59	0.38	0.56	0.51	0.51	0.51	0.43	0.56
Precision	**0.76**	**0.78**	**0.76**	**|0.91|**	0.70	0.68	0.70	0.61	0.73	0.70	0.57	0.73
Recall	**0.70**	**0.74**	**0.72**	**|0.88|**	0.59	0.38	0.56	0.51	0.51	0.51	0.43	0.56
F-measure	**0.70**	**0.73**	**0.70**	**|0.88|**	0.49	0.25	0.48	0.51	0.56	0.52	0.46	0.56
Third Experiment: Pre-trained Glove Model												
Accuracy	**0.64**	**0.74**	**0.74**	**0.85**	0.48	0.42	0.52	0.57	0.43	0.42	0.43	0.57
Precision	0.73	0.79	0.79	**0.88**	0.59	0.53	0.62	0.77	**0.81**	**0.81**	0.81	0.61
Recall	**0.64**	**0.74**	**0.74**	**0.85**	0.48	0.42	0.52	0.57	0.43	0.42	0.43	0.56
F-measure	**0.65**	**0.74**	**0.74**	**0.85**	0.48	0.44	0.53	0.60	0.31	0.29	0.29	0.56
Fourth Experiment: Self-trained Word2Vec Model												
Accuracy	**0.76**	**0.76**	**0.76**	**0.80**	0.48	0.53	0.52	0.49	0.57	0.59	0.55	0.61
Precision	**0.80**	**0.82**	**0.81**	**0.82**	0.60	0.61	0.61	0.60	0.73	0.68	0.72	0.81
Recall	**0.76**	**0.76**	**0.76**	**0.80**	0.48	0.53	0.52	0.49	0.57	0.59	0.55	0.61
F-measure	**0.75**	**0.75**	**0.75**	**0.80**	0.49	0.52	0.52	0.49	0.59	0.57	0.56	0.65

Table 2. *Cont.*

Clustering Algorithms	**AP**				**HAC**				**K-Means**			
Syntactic Similarity	**Cosine**	**Euclidean**	**Minkowski**	**WMD**	**Cosine**	**Euclidean**	**Minkowski**	**WMD**	**Cosine**	**Euclidean**	**Minkowski**	**WMD**
Fifth Experiment: WordNet and NGD												
Semantic Similarity	Path	WUP	NGD		Path	WUP	NGD		Path	WUP	NGD	
Accuracy	**0.87**	**0.73**	**0.83**		0.55	0.58	0.58		0.65	0.50	0.53	
Precision	**0.89**	**0.78**	**0.84**		0.55	0.67	0.75		0.84	0.67	0.71	
Recall	**0.85**	**0.72**	**0.82**		0.55	0.58	0.58		0.65	0.50	0.53	
F-measure	**0.87**	**0.73**	**0.82**		0.48	0.57	0.55		0.65	0.53	0.54	

4. Conclusions

Data mining, text mining, and machine learning can provide solutions to facilitate the process of web services discovery. We proposed a multi-layer data mining architecture for Web services discovery to cluster web services based on five layers: Web services description and data preprocessing; word representation, embedding, and transformation; syntactic similarity; semantic similarity; and Web services clustering. We utilized BOW with TFIDF and three word-embedding models for the word representation, embedding, and transformation layers. Four syntactic similarity measures (Cosine, Euclidean, Minkowski, and Word's Mover) along with three semantic similarity measures (WordNet Path, WordNet WUP, and NGD) were employed to estimate the similarity between web services. Finally, three clustering algorithms (AP, K-means, and HAC) were studied to cluster web services based on their similarities. We ran five different experiments considering the different layers employed. According to their respective agreements and disagreements with ground truth, their respective clustering performance was reported based on several evaluation metrics, namely, accuracy, precision, recall, and F-measure. The results indicate that clusters built based on word embedding models perform better than clusters built based on Bag of Words with TFIDF models. Among the three word embedding models, the pre-trained Word2Vec model reported higher performance in clustering Web services. Consideration of the WMD based on word embedding models increased clustering performance compared to other syntactic measures. Among the three semantic similarity measures, WordNet path similarity showed higher clustering performance. AP performed better in clustering web services and discovering hidden relations between previously assigned clusters among the clustering algorithms.

Our contributions in this paper include the design and verification of a multi-layer data mining architecture for web services discovery based on clustering web services; this can speed up the matchmaking process, as we focused on matchmaking characteristics. The proposed architecture consists of five layers: Web service description and data preprocessing; representation, embedding, and transformation; syntactic similarity; semantic similarity; and clustering. Its implementation and performance evaluation are discussed using appropriate evaluation measures with the aim of clustering web services based on different similarities in order to minimize the search space and measure performance. The results indicate that consideration of word embedding models combined with Affinitive Propagation (AP) clustering can improve clustering performance compared to the K-means and HAC clustering algorithms, which are the most cited in the literature. We propose exploiting the Affinity Propagation (AP) clustering algorithm to cluster web services based on various syntactic and semantic similarity measures, as it showed higher performance than the most cited clustering algorithms (baseline) in the literature for clustering web services.

The proposed multi-layer data mining architecture employed three-word embedding models, two pre-trained and one self-trained model. However, the self-trained Word2Vec model's training process was based on extracted text from a limited number of web services compared to the two pre-trained models. In future works, training the Word2Vec model based on extensive extracted text from web services would improve performance in finding similarities between Web services. However, the performance of clustering based on the self-trained word embedding model is comparative to the other models. We evaluated web services clustering performance based on external evaluation metrics, which requires the existence of ground truth, which we achieved by collecting web services documents and employing human labeling to build a ground truth. Although the collected dataset was based on WSDL and auxiliary text, the proposed solution can work with any format of web services description language or documentation that needs to be evaluated and tested for external validity.

Author Contributions: Conceptualization, W.J.O.; Formal analysis, W.J.O.; Investigation, W.J.O.; Methodology, W.J.O.; Supervision, B.R.; Validation, W.J.O.; Visualization, W.J.O.; Writing—review & editing, W.R. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported and funded by NSERC Discovery Grant Nbr RGPIN/341811-2012.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable, the study does not report any data.

Conflicts of Interest: The authors declare that they have no conflict of interest.

References

1. Crasso, M.; Zunino, A.; Campo, M. Easy web service discovery: A query-by-example approach. *Sci. Comput. Program.* **2008**, *71*, 144–164. https://doi.org/10.1016/j.scico.2008.02.002.
2. Klusch, M. Service Discovery. In *Encyclopedia of Social Networks and Mining (ESNAM)*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 1707–1717. https://doi.org/10.1007/978-1-4614-6170-8_121.
3. Grefenstette, G. Tokenization. In *Syntactic Wordclass Tagging*; Number October, Springer: Dordrecht, The Netherlands, 1999; pp. 117–133. https://doi.org/10.1007/978-94-015-9273-4_9.
4. Obidallah, W.J.; Ruhi, U.; Raahemi, B. Clustering and Association Rules for Web Service Discovery and Recommendation: A Systematic Literature Review. *SN Comput. Sci.* **2020**, *1*, 27. https://doi.org/10.1007/s42979-019-0026-8.
5. Obidallah, W.J.; Ruhi, U.; Raahemi, B. Current Landscape of Web Service Discovery: A Typology Based on Five Characteristics. In Proceedings of the 2016 IEEE/WIC/ACM International Conference on Web Intelligence (WI), Omaha, NE, USA, 13–16 October 2016; pp. 678–683. https://doi.org/10.1109/WI.2016.0121.
6. Obidallah, W.J.; Raahemi, B. A Taxonomy to Characterize Web Service Discovery Approaches, Looking at Five Perspectives. In Proceedings of the 2016 IEEE Symposium on Service-Oriented System Engineering (SOSE), Oxford, UK, 29 March–2 April 2016; pp. 458–459. https://doi.org/10.1109/sose.2016.13.
7. Richardson, L. Beautiful Soup Documentation. 2007. Available online: https://buildmedia.readthedocs.org/media/pdf/beautiful-soup-4/latest/beautiful-soup-4.pdf (accessed on 1 April 2022).
8. Rasmussen, E. Stoplists. In *Encyclopedia of Database Systems*; Springer: Boston, MA, USA, 2009; pp. 2794–2796. https://doi.org/10.1007/978-0-387-39940-9_955.
9. Bird, S. NLTK: The Natural Language Toolkit. In *Proceedings of the COLING/ACL on Interactive Presentation Sessions*; Association for Computational Linguistics: Stroudsburg, PA, 2006; pp. 69–72. Available online: https://dl.acm.org/doi/10.3115/1225403.1225421 (accessed on 1 April 2022).
10. Stanford-University. Stemming and Lemmatization. 2008. Available online: https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html (accessed on 1 April 2022).
11. Salton, G.; Wong, A.; Yang, C.S. A vector space model for automatic indexing. *Commun. ACM* **1975**, *18*, 613–620. https://doi.org/10.1145/361219.361220.
12. Almeida, F.; Xexéo, G. Word Embeddings: A Survey. *arXiv* **2019**, arXiv:1901.09069.
13. Yan, J. Text Representation. In *Encyclopedia of Database Systems*; Springer: Boston, MA, USA, 2009; pp. 3069–3072. https://doi.org/10.1007/978-0-387-39940-9_420.
14. De Boom, C.; Van Canneyt, S.; Demeester, T.; Dhoedt, B. Representation learning for very short texts using weighted word embedding aggregation. *Pattern Recognit. Lett.* **2016**, *80*, 150–156. https://doi.org/10.1016/j.patrec.2016.06.012.
15. Gudivada, V.N.; Rao, D.L.; Gudivada, A.R. Chapter 11—Information Retrieval: Concepts, Models, and Systems. In *Handbook of Statistics: Computational Analysis and Understanding of Natural Languages: Principles, Methods and Applications*; Gudivada, V.N., Rao, C., Eds.; Elsevier: Amsterdam, The Netherlands, 2018; Volume 38, pp. 331–401. https://doi.org/https://doi.org/10.1016/bs.host.2018.07.009.
16. Wikimedia. Wikimedia Downloads. 2005. Available online: https://dumps.wikimedia.org/backup-index.html (accessed on 1 April 2022).
17. Parker, R.; Graff, D.; Kong, J.; Chen, K.; Maeda, K. Gigaword. In *English Gigaword Fifth Edition—Linguistic Data Consortium*; Linguistic Data Consortium: Philadelphia, PA, USA, 2011.
18. Crawl, C. Common Crawl. Available online: https://commoncrawl.org/ (accessed on 1 April 2022).
19. Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.S.; Dean, J. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems 26*; Burges, C.J.C., Bottou, L., Welling, M.; Ghahramani, Z., Weinberger, K.Q., Eds.; Curran Associates, Inc.: New York, NY, USA 2013; pp. 3111–3119.
20. Mikolov, T.; Chen, K.; Corrado, G.; Dean, J. Efficient Estimation of Word Representations in Vector Space. *arXiv* **2013**; pp. 1–12.
21. Camacho-Collados, J.; Pilehvar, M.T. From word to sense embeddings: A survey on vector representations of meaning. *J. Artif. Intell. Res.* **2018**, *63*, 743–788. https://doi.org/10.1613/jair.1.11259.

22. Han, J.; Kamber, M.; Pei, J. Getting to Know Your Data. In *Data Mining*; Elsevier: Amsterdam, The Netherlands, 2012; pp. 39–82. https://doi.org/10.1016/B978-0-12-381479-1.00002-2.
23. Kusner, M.J.; Sun, Y.; Kolkin, N.I.; Weinberger, K.Q. From Word Embeddings to Document Distances. In Proceedings of the 32nd International Conference on International Conference on Machine Learning, Lille, France, 6–11 July 2015; Volume 37, pp. 957–966.
24. Miller, G.A. WordNet: A lexical database for English . *Commun. ACM* **1995**, *38*, 39–41. https://doi.org/10.1145/219717.219748.
25. Cilibrasi, R.L.; Vitanyi, P.M. The Google Similarity Distance. *IEEE Trans. Knowl. Data Eng.* **2007**, *19*, 370–383. https://doi.org/10.1109/TKDE.2007.48.
26. Vitányi, P.M.; Cilibrasi, R.L.; Vitanyi, P.M.B. Normalized web distance and word similarity. *arXiv* **2009**, arXiv:0905.4039.
27. Wu, Z.; Palmer, M. Verbs semantics and lexical selection. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*; Association for Computational Linguistics: Stroudsburg, PA, USA, 1994; pp. 133–138. https://doi.org/10.3115/981732.981751.
28. Frey, B.J.; Dueck, D. Clustering by Passing Messages Between Data Points. *Science* **2007**, *315*, 972–976. https://doi.org/10.1126/science.1136800.
29. Everitt, B.S.; Landau, S.; Leese, M.; Stahl, D. Hierarchical Clustering. In *Cluster Analysis*; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 2011; pp. 71–110. https://doi.org/10.1002/9780470977811.ch4.
30. MacQueen, J. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*; University of California Press: Oakland, CA, USA, 1967; Volume 1, pp. 281–297.
31. Gensim. Gensim: Topic Modelling for Humans. Available online: https://radimrehurek.com/gensim/ (accessed on 1 April 2022).
32. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-Learn: Machine Learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.