WSSE: A Web Service Search Engine for Large Scale Web Service Discovery based on the Probabilistic Topic Modeling and Clustering

Afnan Bukhari
Rochester Institute of technology
Rochester, New York
aab2790@rit.edu

ABSTRACT

With the ever increasing number of web services, discovering the appropriate web service requested by users has become a vital yet challenging task. The aim of this project is to provide an efficient search engine that can retrieve the most relevant web services in a short time. The proposed search engine WSSE is based on the probabilistic topic modeling and clustering techniques that are integrated to support each other. The Latent Dirichlet Allocation (LDA) is used to extract topics from web service descriptions. These topics are then used to group services together. The query is represented as a topic vector and related services are found in the closed cluster using the B-tree index. The topic model is an efficient technique to reduce the dimensionality of word content vectors and to discover the semantic meaning that is hidden in service descriptions. Also, the web service description is represented as a word vector to address the drawbacks of the keyword-based search system. The accuracy of the proposed web service search engine (WSSE) is compared with the keyword-based clustering approach. Also, the precision and recall metrics are used to evaluate the performance of the proposed approach and the keyword-based search system. The results show that the proposed web service search engine (WSSE) based on LDA and clustering outperforms the keyword-based approach.

1. INTRODUCTION

Nowadays the amount of existing web services is growing rapidly. Users need to obtain the appropriate services that satisfy their desired needs. There are various web services discovery approaches that help the end user to find suitable services. In the service-oriented architecture, the web service is an application based on XML. The web service is described by Web Service Description Language (WSDL). The WSDL documents should be published in the Universal Description Discovery and Integration (UDDI) by the web service provider in order to select the appropriate one later. These web services could communicate together over the Internet by the Simple Object Access Protocol (SOAP).

In the most basic scenario of the web service discovery, the web service must be published by the web service provider in the web services registry (UDDI). The web service user sends a request for a required web service. The web service discovery is the responsible for answering the user’s requirements. The traditional approach is based on keyword searching in the UDDIs. However, the UDDI registry is unavailable and not efficient since the huge number of available web services has been published on the web. Web services are discovered based on keywords in four ways. There are (1) Universal Description Discovery and Integration (UDDI); (2) web services websites; (3) generic search engines; or (4) special web service search engines.

Due to the limitations and weakness of the first way (UDDI), the UDDI becomes unfavorable to discover web services. UDDI cannot measure the QoS (quality of service) values of web services. The validity and quality of web services information are not guaranteed. UDDI is used just for web service discovery. There is no connection between the UDDI registry and current services. Moreover, UDDI cannot handle an enormous amount of data across several UBRs and the search queries on several UBRs become time consuming. So, these limitations impact the performance, reliability, scalability, and efficiency of the web service discovery and the quality and validity of results.

The second way is the web service website that also has some vulnerabilities that make these sources unreliable. Web service websites are websites that gather all web services and provide an interface for web service searching. One example of web service websites is XMethods.com. XMethods provides a list of available public web services by describing the title, description, and function of web services. Another example is the programmableweb directory [2] that organized the web services by category to allow the user to search for his desired web service. A programmableweb directory [2] is a catalog that is organized by people who discover the web services through the Internet. The provided description of web services is a human readable way; the machine cannot understand it. The searching by category is inefficient because it’s time-consuming for the user to locate the suitable one. Some of these websites are shut down after several years of launching.

The third way is generic search engines such as Google, Yahoo, Bing and so on. This way also cannot provide appropriate results for several reasons: 1) Even after specifying the correct keywords of the query, the typical number of the retrieved result from Google is extremely large. 2) Only 12% of retrieved results are web services. The small fraction of the result is web services. 3) It is also time-consuming for users to filter through the whole web
service results. 4) The general-purpose search engines are based on the keywords search and cannot allow the search for the non-functional requirements[11].

The fourth way is the special web service search engine that is a distinctive search engine for web service descriptions such as OWL-S and WSDL files. The special web service search engine overcomes the drawbacks of the three ways that are explained above. The main idea of the special web service search engine turns around crawling on multiple UDDI registries and the web to gather all available web services. Web service descriptions (WSDL documents) will be fetched and stored for further processing. Users query the search engine for particular web services, and the relevant results should be retrieved to satisfy the users’ requests. Using the keywords is the main and best method to describe the user’s objective. The keyword-based method can correctly match the query with the web Service name or description [8].

The aim of this project is to provide an efficient search engine that can retrieve the most relevant web services in a short time. The purpose of this search engine is to provide high precision and recall rating. The keyword-based search engine suffered from low recall and precision. To improve the precision and recall rating, the search engine needs to understand the semantic meaning of web service description documents and the user’s query. The topic modeling can put the synonymous terms under the same topic. Describing the service descriptions as topic vectors which are the probability distributions of the topic over documents will reduce the dimensionality of word vectors and help to provide efficient clustering results. Using the probabilistic topic modeling with clustering technique can achieve the semantic matching between service descriptions and the user’s query. Moreover, using indexing will provide the good performance and make the search engine faster and easier to use.

In this project, the semantic web services search engine WSSE based on clustering will be developed. The semantic similarity between web service descriptions is computed based on the topic modeling. The K-mean algorithm is used in order to cluster web services based on their related topics. The search engine is developed to discover both the Representational State Transfer (REST) and Simple Object Access Protocol (SOAP) web services.

The proposed web services discovery approach uses the clustering, topic modeling, and indexing techniques. For each web service, the vector of topics should be generated. Then using these topic vectors to perform clustering algorithms such as k-means and agglomerative to obtain the best clusters. The incorporation between the topic modeling and clustering techniques will boost the performance and the ability of the web service search engine to retrieve the most relevant results. This semantic clustering provides the higher precision and recall rate. The higher precision score of the proposed web service search engine WSSE indicates that the proposed search engine can provide accurate results such as the state-of-the-art web service discovery approach. Then, the indexing technique such as the B-tree index is applied to each cluster to improve the query response time and disk space usage.

This paper is organized as follows. Section 2 gives a brief background and related work. Section 3 provides the design of the service query system. Section 4 discusses the experimental results and performance evaluation. Lastly, section 5 concludes the paper.

2. RELATED WORK

There are many web service discovery approaches that have been developed to optimize the discovery mechanism in the SOA. Some approaches are based on the criteria of functionality while others are based on the QoS (Quality of Service) criteria such as the web service price, availability, etc. or services popularity. Some approaches found web services based on syntactic or semantic matching. Most of web service discovery approaches search for the web service using the functional attributes as the basic criteria while non-functional attributes are considered complement criteria in the web services discovery. The keyword matching is the basic matching method. However, the keyword matching cannot understand the semantic meaning of the web service descriptions and the user’s query.

2.1 Discovery based on functional criteria

The web services discovery based on functional criteria extracted the unique terms and vocabulary that represented the functionality of web service from the web service description documents. Then, the similarity measure was calculated to retrieve the appropriate results. Most of the web services discovery approaches used the information retrieval techniques to extract the functionality meaning of services and find the relevant web services. Web services were described as XML style documents that needed to extract the meaningful information that represented the services functionality. Information retrieval approach represented the web services description documents as vectors.

Elshater et al. [9] used the vector space model (TF-IDF model) that extracted the relevant information conveyed within service description documents. Then, the service corpus was represented as the TF-IDF matrix. Since, the TF-IDF vector was sparse so to retrieve the relevant documents efficiently in a short time, the K-D tree index was applied on TF-IDF matrix for easily navigate the matrix. The K-nearest neighbors algorithm was used to find the description that is similar to the query and the nearest neighbors descriptions in the tree. This approach provided satisfied recall and precision rating and improved the performance of the web service discovery by reducing the query response time. In general, the indexing process could boost the speed of searching for web services but suffered from several drawbacks. The index needed to be stored in an additional space. The index must be updated when the data is changed. Also, the indexing process is expensive computational process.

Moreover, the WESS (Web Service Search Engine) [11] is similar to the previous work. But it developed to discover the semantic (OWL-S, and SAWSDL) and non-semantic web service descriptions (WSDL) on the web. In this approach, the web service documents will be parsed and stored information in the intermediate files in order to index them. Hatzi et al. [11] used the inverted index that was represented as a hashtable where the key was the term led
to all web services description documents that contained it. The TF-IDF model also was used to retrieve the appropriate results and the cosine similarity computed the similarity between the query and web services. Finally, the interface was developed to represent the results. This approach provided good results by involving the semantic and non-semantic web services, however, the index needed to be stored in an additional space and needed to frequently updating. Also, it didn’t support the RESTful web services.

All the above search engines that were used the information retrieval (IR) and based on keyword-based search technique were insufficient for obtaining relevant results. They could not retrieve the web service if the web service did not have at least one exact word of search query terms, and could not retrieve the web service that used different synonyms or variations of query terms. Moreover, these search engines suffered from the low recall rate. The information retrieval could not reveal the semantic meaning of web service descriptions. There was a lot of works have been developed to handle the lack of understanding the semantic meaning of the web services. Grouping web services into similar clusters handled this drawback and improved the precision and recall significantly. Many efforts have been done to clustering web services.

2.2 Web service clustering

Grouping web services into similar clusters and pruning the query space improved the performance of web service discovery by reducing the time and space overheads. Using a good clustering algorithm could increase the precision and recall for web service searches and retrieve more accurate results. Integrating the clustering approach into the search engine, improved the quality and efficiency of the web service discovery. It helped to prune the search space for web services relevant to the user’s requests. Also, some missed web services that may provide the most quality services were more likely to be retrieved by applying the clustering approach. Clustering web services might be based on some different properties such as the similarity of functional properties, non-functional properties, and social properties. Elgazzar et al. [8] and Chen et al. [6] focused on clustering web services based on functional similarities. Chen et al. [6] performed clustering based on the similarity of functional properties and tags. However, Yi et al. [18] focused on the QoS while Zhou et al. [21] used functional and non-functional properties but for semantic web services.

Elgazzar et al. [8] focused on clustering web services based on the functional similarities. They extracted features from WSDL documents based on the text mining technique. The features were the content, types, messages, ports of the WSDL, and the web service name. These five features represented the service functionality of web services. First, in the WSDL Content feature, the vector of content words was generated for each cluster and used TF-IDF for the similarity measure. Second, in the WSDL type feature, the complexType was used, and the element types were extracted to determine the number of similar types between two web services. The WSDL Messages feature focused on the number of messages matched between two web services, while the WSDL Ports feature determined the matched port-Type among the pair of web services. The last feature was the web service name. Web services were clustered by using the Quality Threshold (QT) clustering algorithm based on these features. The semantic similarity between web services was calculated based on the five similarity features using the equation below. Where \( \theta(s_i, s_j) \) is 1 that means the web services are similar.

\[
\begin{align*}
\theta(s_i, s_j) &= 0.2S(T_i, T_j) + 0.2\text{sim}(sname_i, sname_j) \\
&+ 0.2\text{match}(typ_i, typ_j) + 0.2\text{match}(msg_i, msg_j) \\
&+ 0.2\text{match}(port_i, port_j)
\end{align*}
\]  

(1)

The approach [8] provided high recall and precision of web service search.

Chen et al.’s approach [6] is based on the features of WSDL documents and web service tags. Chen et al. [6] extended the approach of clustering based on the function similarities in [8] and they incorporated tags. In this approach [6], first Chen et al. extracted five features from the web service description to compute the similarity between them. Then, Chen et al. [6] proposed an approach that recommended suitable tags to web services. After that, the K-means clustering algorithm is used to cluster web services. The similarity between web service descriptions is computed by the composite similarity. This similarity integrated the web service descriptions similarity and the tags similarity [6] as the equation below:

\[
C\text{Sim}(s_i, s_j) = (1 - \lambda)S\text{im}_{\text{wsdl}}(s_i, s_j) + \lambda S\text{im}_{\text{tag}}(s_i, s_j)
\]

(2)

Chen et al. [6] used the precision and recall metrics to compare their approach with the clustering WSDL documents based on the functional similarity approach [8]. The results shows that the ability of Chen et al.’s approach outperformed the approach without using tags. Tags could provide more information to web services, and using this information improved the performance of the web service clustering, discovery, and results.

Wei et al. [12] aim to enhance using tags for the web service discovery by providing high-quality tags to each service. Incorporating the web service tags helped to include expected web services that were disregarded because it didn’t match the user’s query terms. Using tags with web service functional criteria achieved an efficient performance for web service discovery as it showed in [6]. Wei et al. proposed a web service tag learning system called WTLearning [12] that provided the high-quality tag resource and was designed to learn and understand the tag annotations for the web service discovery. WTLearning could recommend a relevant tag list for the web service to support further operations.

Zhou et al. [21] clustered web services based on the functional and non-functional properties for semantic Web services. Zhou et al. [21] seek to represent an approach to discover semantic web services based on the functional and non-functional properties using the genetic algorithm (GA) clustering. The euclidean distance measured the QoS similarity between two services. The QoS-based service discovery considers the web service as tri-tuple, WS=[D, F, QoS], D means the basic information such as the service
name, categories, etc.; F means inputs, output, precon-
dition and effects, that was known as IOPe; and QoS means
the quality of the service. All the web services candidates
were represented as a matrix which then measured the dis-
tance between QoS of requesters and QoS of web service
candidates, and then selected the most appropriate web ser-
vice that gave the shortest distance. The provider published
his/her web service that was clustered by GA. The requester
send a query that was matched with clustered web services.
Then the accurate web service candidates were provided to
the users.

2.3 Discovery based on functional and non-
functional criteria

Zhang et al. [20] proposed a WSExpress search en-
gine that provided the desirable web services based on the
functional and non-functional requirements of users. WS-
Express considered both the functional and non-functional
attributes for retrieving the appropriate web services. First,
the similarity based on functional attributes was computed
by using the TF/IDF model. Also, the cosine similarity be-
tween the web service operation and user’s query was com-
puted. Then, the similarity based on QoS attribute was
computed. Once the functional and non-functional crite-
rion was calculated, the relevant web services were ranked,
the search engine found the high rating score by combining
the score of functional and non-functional similarities. The
search engine presented its ability to outperform another ap-
proach that did not consider the non-functional attributes.
The recall and precision results of the WSExpress approach
were higher. The results of WSExpress indicated that the
top web service candidates in the recommending list was
strongly pertinent to the user’s query even without consider-
ing the QoS values. However, this approach could not han-
tle the historical behavior of users.

3. DESIGN SERVICE QUERY SYSTEM

3.1 Web Service Discovery Framework

This section describes the web service discovery
framework. The architecture of the proposed web service
search engine (WSSE) consists of four major phases:

1. Data preprocessing
2. Topic modeling
3. Service clustering
4. Service discovery

The figure(1) illustrates the framework of the web
service discovery.

3.2 Data preprocessing

In this phase, all features should be extracted from
service descriptions. Each service description will be con-
verted to a word vector. The vector space model technique
that is used is the TF-IDF (Term Frequency-Inverse Docu-
ment Frequency) model. The information retrieval method
is used to obtain the important information from web ser-
vices. The vector space model is used to represent the corpus
of web services by a word-document matrix. The word vec-
tor contains unique terms that have the most meaningful

Figure 1: A framework of a web service discovery
system

functionality of each web service. Then, the IR system will
generate the word-document matrix for the whole vectors.
The word vectors are then used in the LDA to generate topic
vectors for web services descriptions. To represent web ser-
vices as word vectors, the preprocessing process includes the
following steps:

1. Extract words from the web service: All terms in the
service descriptions are extracted in the bag of words.
2. Removing XML tags.
3. Tokenization: Break the composite terms into simple
words. The term is broken based on the case change
(camelCase), comma, dash, underscore, period, white
space, numbers, parenthesis, etc.
4. Splitting: Split the composite terms that are combined
with lowercase. For instance, zipcode will not be tok-
enized in the Tokenization process. So split the largest
two words by using wordnet dictionary.
5. Removing English stop words: e.g. a, the, it, etc.
6. Stemming: A porter stemmer is used to obtain the
root of words. The porter2 (snowball) algorithm is
applied. For example: "Performed" and "Performing" be-
come "perform".

Figure(2) represented an example of preprocessing the data.

Then, service descriptions will be represented as
word vectors. By using the term frequency-inverse docu-
ment frequency model, words are converted to the TF-IDF
vector. The TF-IDF weight is calculated for each term in each document. The TF-IDF is calculated by the equation below for each term i in document j:

$$TF - IDF = TF_{i,j} - IDF_i$$

(3)

where the $TF_{i,j}$ is the term frequency for each term i in document j. The equation below calculates the $TF_{i,j}$:

$$TF_{i,j} = f_{i,j}/max_k f_k$$

(4)

The $IDF_i$ is the Inverse Document Frequency of a word i collection of documents. The equation below calculates the $IDF_i$:

$$IDF_i = \log(N/n_i)$$

(5)

where $N$ is the total number of documents, and $n_i$ is the number of documents that only contains the word $i$[4][11][9][8][20].

For generating the TF-IDF matrix, the scikit-learn python library is used [3]. The scikit-learn is an open source python library for data mining and data analysis [3]. The actual size of the matrix is (7560 X 7751). The figure(3) below is a snapshot of the generated TF-IDF matrix.

Figure 3: A snapshot of the TF/IDF matrix

3.3 Topic modeling

The probabilistic topic model that is used in our approach is the Latent Dirichlet Allocation (LDA) [5]. The probabilistic topic model is an efficient method for storing and computing sparse vectors. The aim of using LDA is to generate the topic vector of the web service description that reduces the dimensionality of the word vector. The LDA uses a probabilistic view to generate topics from service descriptions. Then, these topic vectors are used to group service descriptions into clusters. LDA is based on measuring the word frequency from heterogeneous service descriptions. The topic model generates mixtures of topics for each service description, where a topic is a probability distribution that measures how many times the words are assigned to a topic. In our approach, we set the $\alpha = 0.1$ and $\beta = 0.01$. The best set of topics should be found by estimating $\phi(j)$ which computes the importance of words under each topics and $\theta(s)$ which measures the probability proportion of topics in each service document [5].

LDA could significantly reduce the dimensionality of word vectors. Moreover, the LDA can capture the semantic concepts from web service descriptions based on probability distributions. In each service description S, the word W will be produced by sampling a topic Z from topic distribution. After that, it will sample a word from topic-word distribution. The probability of the i th word frequency in each service is calculated by the equation:

$$P(W_i) = \sum P(W_i|Z_i = f)P(Z_i = f)$$

(6)

Where $Z_i$ is a topic of the word $W_i$ and $P(Z_i = f)$ indicates the probability of the topic f that becomes the topic of the word $W_i$. The $\theta(s) = P(Z)$ is the topic-document distribution that means the distribution of service description S over topics. The $\phi(j) = P(W|Z = j)$ refers to the word-topic distribution that means the distribution of topic j over words[5]. The Gibbs sampling algorithm [10] is used in this approach for estimating parameters in the LDA.

The hyperparameter $\alpha$ and $\beta$ are used as dirichlet prior on $\theta$ and $\phi$. The dirichlet prior $\alpha$ is a prior observation that measures how many times the topic j is represented in a document [5]. The dirichlet prior $\beta$ is the prior observation that measures how many times the words are assigned to a topic. In our approach, we set the $\alpha = 0.1$ and $\beta = 0.01$. The best set of topics should be found by estimating $\phi(j)$ which computes the importance of words under each topics and $\theta(s)$ which measures the probability proportion of topics in each service document [5].

Figure(4) shows the topic modeling visualization for ten web service descriptions (APIs). After training the LDA model, it maps the high-dimensional representation of web services descriptions that is word vectors of a 7560 X 7751 matrix, to a low dimensional representation which is topic vectors of a 7560 X 100 matrix. The MALLET toolkit [15] is used to run LDA. The optimal number of topics will be determined by the perplexity. The perplexity is a common measure to evaluate the topic model with the k topics [16]. MALLET is used to generate the perplexity value for each number of topics. The perplexity is based on measuring the log-likelihood of a testing set [16]. The dataset will be separated into two parts: training and testing sets. The training set has 80% of the data while the testing set has 20%. The testing set is unseen documents Wd. The perplexity formula is [16]:

$$Perplexity(w) = \exp \left( -\frac{L_w}{\text{countoftokens}} \right)$$

(7)

where W is test set $L_w$ is the log-likelihood of the test set W [16].
The lower perplexity value indicates how better the topic model is. The result of the perplexity of our data is:

\[ L_w = \log(P(w|\varphi, \alpha)) = \sum (\log(P(w|\varphi, \alpha))) \quad (8) \]

The second plan uses the hierarchical clustering algorithms (agglomerative). In the hierarchical clustering method, two primary parameters should be specified: the metric and linkage criterion as shown in the agglomerative algorithm 2[17].

**Algorithm 1** K-means algorithm

1. procedure K-MEANS(S,K)  \( S \) is a set of service descriptions as word vectors or topic vectors, \( K \) is the number of clusters.
2. 1- Select \( K \) points as the initial centroids.
3. 2- Repeat:
4. a. Form \( K \) clusters by assigning all services to the closest centroid by the cosine similarity measure.
5. b. Recomputed the centroid of each cluster.
6. 7: Until the centroids don’t change.

**Algorithm 2** Agglomerative algorithm

procedure AGGLOMERATIVE(S) \( S \) is a set of service descriptions as word vectors or topic vectors.

1. 1- Place each data point into its own singleton group
2. 2- Repeat: iteratively merge the two closest groups
3. 4: Until: all the data are merged into a single cluster

### 3.5 Service Discovery

The data is grouped under suitable clusters. The discovery approach then matches the query with the more similar cluster. Then, it searches inside the related cluster rather than searching in the whole data. The user’s query is modeled as a vector. Then, the similarity between the query and services is computed. The cosine similarity measures the difference or divergence between query \( p \) and \( q \) service, as depicted by the equation 9:

\[ \text{CosSim}(p,q) = \frac{\text{Dotproduct}(p,q)}{||p|| \times ||q||} \quad (9) \]

Then, if the retrieved results are not satisfying, the second closed cluster is chosen to find the relevant web services. Finally, the result is ranked based on the similarity score between the query and web service descriptions. The B-tree index is applied on the id of services to fetch the web service information from the database. The B-tree index increases the speed of the search engine and avoids the latency. The B-tree index speeds up the process of searching by minimizing the number of database accesses to locate the web services. Without an index in the database, the search would scan the whole collection to fetch the desired data.

### 4. EXPERIMENTS AND EVALUATION

#### 4.1 Dataset
The experiments were performed on the programmableweb dataset. There are around 11,199 web services (WSDL and Restful), organized into 46 categories. Only 7560 web services were used in the proposed approach from the largest 20 categories. The service data was stored in .txt file. In the file, each line corresponded to a service (API). The fields are separated by a delimiter $\#$. The multiple value of fields is separated by $\#\#$. The format of each API record is: $\#id\#\#$ title $\#\#summary\#\#rating\#\#name\#\#label\#\#author\#\#description\#\#type\#\#download\#\#useCount\#\#sampleUrl\#\#dataUpload\#\#dateModified\#\#remoteFeed\#\#numComments\#\#commentsUrl\#\#tag1\#\#tag2\#\#tag3\#\#category\#\#protocols\#\#serviceEndpoint\#\#version\#\#wsdl\#\#dataFormats\#\#apigroups\#\#example\#\#clientInstall\#\#authentication\#\#ssl\#\#readonly\#\#VendorApiKits\#\#CommunityApiKits\#\#blog\#\#forum\#\#support\#\#accountReq\#\#commercial\#\#provider\#\#managedBy\#\#nonCommercial\#\#dataLicensing\#\#fees\#\#limits\#\#terms\#\#company\#\#update$.

The services data was parsed and stored in the database. The MongoDB [1] was used. The fields of the database were [ id, title, summary, rating, name, label, author, description, type, downloads, use Count, sample Url, download Url, date Modified, Remote Feed, num Comments, comments Url, tags, category, protocols, service Endpoint, version, wsdl, data Formats, apigroups, example, client Install, authentication, ssl, readonly, VendorApiKits, CommunityApiKits, blog, forum, support, accountReq, commercial, provider, managedBy, non-Commercial, data Licensing, fees, limits, terms, company, updated]. For our approach, some fields are disregarded that does not provide any functionality meaning such as serviceEndpoint, version, wsdldata Formats, apigroups, example, clientInstall, authentication, ssl, readonly, VendorApiKits, CommunityApiKits, blog, forum, support, accountReq, commercial, provider, managedBy, non-Commercial, data Licensing, fees, limits, terms, company, updated. Besides these some features(fields) of the service data that display the functionality of the service were used, such as the service title, summary, description, tags, and category. As mentioned previously, the largest 20 categories were taken for the experiments. The data contained 7560 service descriptions. The table(1) represents the number of service descriptions in each category.

<table>
<thead>
<tr>
<th>Tools</th>
<th>858</th>
<th>Messaging</th>
<th>412</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet</td>
<td>645</td>
<td>Mapping</td>
<td>379</td>
</tr>
<tr>
<td>Financial</td>
<td>543</td>
<td>Reference</td>
<td>356</td>
</tr>
<tr>
<td>Social</td>
<td>528</td>
<td>Shopping</td>
<td>355</td>
</tr>
<tr>
<td>Enterprise</td>
<td>489</td>
<td>Government</td>
<td>345</td>
</tr>
<tr>
<td>Science</td>
<td>345</td>
<td>Photos</td>
<td>252</td>
</tr>
<tr>
<td>Payment</td>
<td>325</td>
<td>Other</td>
<td>237</td>
</tr>
<tr>
<td>Telephony</td>
<td>299</td>
<td>Video</td>
<td>232</td>
</tr>
<tr>
<td>Advertising</td>
<td>256</td>
<td>Travel</td>
<td>227</td>
</tr>
<tr>
<td>Search</td>
<td>253</td>
<td>Education</td>
<td>224</td>
</tr>
</tbody>
</table>

Table 1: 20 categories of Web services

Web service descriptions were preprocessed to extract the service functionality. The word vector and topic vector were generated for each service. The size of word vectors was 7560 X 7751 which occupied 296.3 MB while the size of topic vectors was 7560 X 100 which occupied 22 MB. The data representation size, using the topic modeling, reduced significantly i.e. by more than 96%. The table(2) shows the effect of using word vectors and topic vectors to the memory size, the CPU time of creating each kind of vectors and the CPU time of K-means clustering with each vectors.

<table>
<thead>
<tr>
<th>Vectors</th>
<th>Word vectors</th>
<th>Topic vectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>7560 X 7751 = 296.3 MB</td>
<td>7560 X 100 = 22 MB</td>
</tr>
<tr>
<td>CPU time creating the matrix</td>
<td>821 Minutes</td>
<td>13.6 Minutes</td>
</tr>
<tr>
<td>CPU time for K-means clustering</td>
<td>3.1528 Minutes</td>
<td>0.092 Minutes</td>
</tr>
</tbody>
</table>

Table 2: Comparison of two vectors

4.2 Web Service Clustering Evaluation

This section shows the comparison results of the clustering performance using the two types of vectors; the K-means and agglomerative clustering algorithms. The MATLAB [14] was used to apply both the clustering algorithms. The MATLAB provides a statistics and machine learning toolbox for data mining and analyzing [14]. The word vectors and topic vectors were input to the K-means and agglomerative hierarchical algorithms. Different number of Ks were tested in the range of [5,40]. The clustering results of the two vectors were compared by using the internal evaluation (such as Davies-Bouldin index measure) and the external evaluation (such as the Accuracy, Entropy and Purity).

The Davies-Bouldin index measure is an internal evaluation measure that is used to evaluate the consistency within clusters of data by measuring the cohesion (how similar an instance is into the cluster) and separation (compared to other clusters) of clusters. If the Davies-Bouldin index of the algorithm is low, that means the algorithm (K-means) produces clusters that have more similar instances inside the cluster, and dissimilar instances to other clusters. The smallest Davies-Bouldin index value indicates that the best clustering results are produced. The figure(6) shows Davies-Bouldin index values of k-means clustering with the word vector approach and the topic vector approach with different value of ks.

![Figure 6: K-means internal clustering evolution](image-url)
hierarchical clustering algorithm (agglomerative) was performed[17]. In the hierarchical clustering method, two primary parameters should be specified: the metric and the linkage. The cosine similarity was chosen as the metric since since the experiment deals with texts. The minimum or single-linkage clustering was selected to merge the two clusters that contain the closest pair of elements. Both agglomerative clustering algorithm on topic vectors and word vectors were performed. The results of this agglomerative hierarchical clustering is presented as a dendrogram as figure(7).

![Hierarchical clustering dendrogram](image)

**Figure 7: Hierarchical clustering dendrogram**

The Cophenetic Correlation Coefficient method is used to evaluate the dendrogram. If the Cophenetic Correlation Coefficient value of the clustering is closer to 1, the better the clustering is. For the experimental data, the cophenetic correlation coefficient of word vectors was 0.5819 while the cophenetic correlation coefficient of topic vectors was 0.6061. Several number of clusters were used in the range of [5,40]. Then the Davies-bouldin index measure evaluated the quality of clustering results. Also, the Davies-bouldin index measure showed the performance and the quality of the word and topic vectors. The clustering results of the agglomerative clustering algorithm on topic vectors outperformed the agglomerative clustering on word vectors as shown in figure(8). However, the drawbacks of this approach is that the complexity of agglomerative clustering is $O(n^3)$.

![Hierarchical internal clustering evaluation](image)

**Figure 8: Hierarchical internal clustering evaluation**

The external clustering evaluation measures such as the accuracy, entropy, and purity metrics were also computed to evaluate the clustering performance. The four clustering methods were compared by using the three metrics results.

The data is classified into 20 different categories. The categories as q classes of web services were used. The total number of services is n and the k is the number of clusters. The equations below explain each metric:

$$\text{Accuracy} = \frac{\text{Thenumberofservicesareclusteredcorrectly}}{\text{Totalnumberofservices}}$$  \hspace{1cm} (10)

**Entropy:** The entropy measures the distribution of classes in each cluster. The smallest value of entropy indicates the best clustering solution.

$$E(C_j) = -\frac{1}{\log(q)} \sum_{i=1}^{q} \frac{n_{ij}}{n_j} \log \left( \frac{n_{ij}}{n_j} \right)$$  \hspace{1cm} (11)

Where q represented the number of classes (labels), and $n_{ij}$ is the number of web services of the i th class that were assigned to the j th cluster. Then, the average of entropy of clustering solution is calculated by the equation(12).

$$\text{Entropy} = \frac{1}{k} \sum_{j=1}^{k} E(C_j)$$  \hspace{1cm} (12)

**Purity:** The purity computes the coherence of clusters. If the purity value is closer to 1, it indicates a good clustering result.

$$P(C_j) = \frac{1}{n_j} \max_{i} (n_{ij})$$  \hspace{1cm} (13)

Then, the average of purity of clustering solution is calculated by the equation(14).

$$\text{Purity} = \frac{1}{k} \sum_{i=1}^{k} \frac{n_i}{n} P(C_i)$$  \hspace{1cm} (14)

The table(3) shows the results of the accuracy while the figure(9) shows the results of the average of the three metrics values. The table(3) and figure(9) shows that the use of topic vectors with k-means clustering technique outperforms the other methods. Using the topic vectors with k-means provides the smallest entropy value and the highest purity and accuracy values which indicates that the topic vectors with k-means approach is the best clustering solution. So, it confirms that the contribution of the topic modeling with clustering can improve the performance of clustering; improve the accuracy of clustering results, and they can mutually promote each other. Also, the method of using topics as clusters, by assigning the service to the topic (a service is assigned to the cluster x if $x = \arg \max_{x} P(C_{x})$), gives better result as compared to the previous works [19] that used the same method. The accuracy of the experiments performed is 56% while [19] gives 54.88% accuracy value. Also, the proposed method of using topic vectors with K-means provides better results as compared to the [19] that provided 30%.

### 4.3 Web Service Discovery Evaluation

Through the experimentation it has been proven that the cooperation between the topic modeling and clustering improved the performance of the individual techniques. Representing the service descriptions based on LDA as topic vectors, outperformed the performance of using TF-IDF as word vectors. The LDA significantly decreased the dimensionality of the vectors and provided good performance in expressing semantic concepts. The services were grouped under suitable clusters corresponding to their similar topics. The discovery approach then matched the topics of the
query with the more similar group. It then searched inside the group with related topics rather than searching the entire data. The user query was modeled as a topic vector using MALLET [15] (a free Java Implementation of LDA), generating the word-topic distribution and topic-document distribution. Then, the similarity between the query and services was computed based on their corresponding topic distributions. The search system was web-based search system. It was written in Java, JSP, and Java Servlet that ran on the Tomcat server. It uses servlet as a middleware between the web browser and database. Also, Java servlet allows to develop a dynamic web application. The figure(10) shows the main page while figure(11) shows the list retrieved from such a query.

Table 3: Accuracy results

<table>
<thead>
<tr>
<th>Vector</th>
<th>Algorithm</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word vectors</td>
<td>K-means</td>
<td>47%</td>
</tr>
<tr>
<td>Word vectors</td>
<td>agglomerative</td>
<td>36.6%</td>
</tr>
<tr>
<td>Topic vectors</td>
<td>K-means, topics=20</td>
<td>53%</td>
</tr>
<tr>
<td>Topic vectors</td>
<td>K-means, topics=100</td>
<td>59%</td>
</tr>
<tr>
<td>Topic vectors</td>
<td>K-means, topics=250</td>
<td>46%</td>
</tr>
<tr>
<td>Topic vectors</td>
<td>agglomerative</td>
<td>39%</td>
</tr>
<tr>
<td>Topic vectors</td>
<td>Assigning to cluster x if x = argmax, 56%</td>
<td></td>
</tr>
</tbody>
</table>

Figure 9: External clustering evaluation

Also, precision computed at the top k measured how relevant the web service results in the top k are to the user query. When the precision was measured, the first k retrieved services were taken into account, K=5,10,15,20. The average of these matrices was computed for 20 queries.

\[
Recall = \frac{|relevantservices \cap retrievedservices|}{|relevantservices|} \quad (15)
\]

\[
Precision = \frac{|relevantservices \cap retrievedservices|}{|retrievedservices|} \quad (16)
\]

\[
Precision@k = \frac{|relevantservices \cap retrievedservices|}{K} \quad (17)
\]

The experiments tested around 20 different generated queries, such as one-word query, two to three words query, and a sentence query. Since the programmableweb organized web services into several categories, some of queries were selected in order to target one category for each of them.

Based on the experiments, the search was based on categories which meant that if the query was the keyword of a category, results of using the proposed search engine would be similar to the programmableweb website. However, if the query was any word such as "picture", the proposed search engine retrieved more accurate results and more relevant web services than the programmableweb and text-search. The drawback of the programmableweb is that it is keyword-based matching. So if the query doesn’t match exactly i.e. the same words do not happen to be in the query as in the web service descriptions, the relevant results would not be retrieved by the programmableweb and text-search. So, programmableweb and text-search retrieved all web services that contained the “picture” word in the service name, title, summary or description, even if these services were not about picture API. Also, the keyword-based approach suffered from understanding the semantic meaning of the query. For example, programmableweb retrieved different results with the same meaning of queries such as “picture” and “photo”. If the query was “picture”, the programmableweb website retrieved 22 web services with Precision@20 is 0.6. But, when the query was “photo”, the programmableweb website retrieved 132 web services with Precision@20 is 0.9.

On the other hand, the proposed search engine (WSSE) could understand the user’s query and generated

Figure 10: WSSE: a web service search engine

The accuracy and effectiveness of the web service search system must be evaluated. The proposed web service search system based on the probabilistic topic model was compared with the available web services website i.e. programmableweb website (directory) [2] and the simple text-matching approach that is based on keyword matching technique. The two metrics, precision and recall were computed.
the hidden topics from the query to compare them with service topics. The proposed search engine (WSSE) based on the probabilistic method could easily discover the latent variables and hidden topics of the user’s query. The proposed probabilistic search engine (WSSE) was found to support the semantic meaning and the retrieved results were same for both queries, i.e. "picture" and "photo", with Precision@20 equal to 1. Moreover, when using programmableweb and text-search, the query must be chosen carefully and the keyword must be specified with the query. Providing a keyword with the query is not needed in the proposed search system (WSSE). So, if the query is 'biology', the precision of the probabilistic method was found to be better as compared to the other approaches.

The second type of the query was the two to three words query, such as "city weather event". The programmableweb and text-search could not retrieve any corresponding web service. The Precision@20 of both approaches was 0. However, the proposed search engine (WSSE) could find the common topic between these words and retrieved the most similar services that combined the three words. The Precision@20 of the proposed search engine was 0.85.

The third type of query was a sentence such as "API for editing images and videos". For such queries, the precision of the proposed search engine (WSSE) was much better than the other approaches. Figure(12) depicts the average of the precision at top 20 for the three types of queries. Three different queries were used for each query type. The average precision was computed by the equation (18).

\[
\text{AveragePrecision}_{@k} = \frac{\sum_{i=1}^{n} \text{Precision}_{@k_i}}{n}
\]  

where n is the number of queries.

Figure 12: Top-k average precision using 3 queries of each query type.

Figure(13) describes the average of precision results of the three search systems when 20 queries were used. The top K retrieved results were 5,10,15,20. The highest precision value indicates the best retrieved results that were obtained from that approach. As can be seen from the figure, the proposed probabilistic search engine (WSSE) outperforms the other approaches which are based on keywords matching technique.

Moreover, the mashup descriptions were used as queries. Mashup is an application that uses at least one API. The number of APIs of each mashup was used as the ground truth against which the proposed search engine’s results were compared. The recall and precision of the proposed search engine and the text-search system were computed over 40 queries which were descriptions of the mashups. For example, in the query "Centralized environment to efficiently manage your online advertisement campaigns no matter which network you are using for your messages. Open source" [2], the number of APIs used by this mashup are 5.

The proposed search engine (WSSE) retrieved 98 APIs only 3 APIs of them were relevant. On the other hand, the text-search system retrieved 0 API. The recall and precision of the proposed search engine (WSSE) was 0.6 and 0.0306 respectively while the recall and precision values for text-search system was 0. Table (4) depicts the result of recall and precision of the two approaches for 40 queries. The results show that the proposed search engine (WSSE) performs better than keyword-based discovery approaches.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Recall</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSSE</td>
<td>63%</td>
<td>5.4%</td>
</tr>
<tr>
<td>Text-search system</td>
<td>2.5%</td>
<td>1.25%</td>
</tr>
</tbody>
</table>

Table 4: Recall & Precision results

Also other experiments are conducted to compute the average of the query response time for the proposed search engine (WSSE) and text-search approaches. The experiments included 6 different queries and the overall query response time is calculated in term of second. The figure(14) represented that the proposed search engine (WSSE) significantly outperforms the other approaches. The proposed search engine (WSSE) takes very short time to retrieve the results comparing with others. As we can see, the time reduced by more than 30% comparing with the text-search system.

Based on the results of the experiments, the topic model showed to improve the performance of the service discovery. The proposed search engine, based on LDA, performed better than the keyword-based methods. The topic modeling promoted the semantic clustering and semantic discovery, which in turn retrieved more accurate web services. Also, the topic modeling helped the search engine to understand the user’s query and extract the hidden topics from the query. It allowed the user to submit any sentence without having to specify the query keywords. The lack of semantic matching and discovery in the programmableweb
and the text-search methods led to obtaining low precision results for both approaches. However, the recall result of the proposed search engine (WSSE) has shown to increase the output by 60% as compared to the text-search system method.

5. CONCLUSION

In this project, the web service search engine was implemented based on the probabilistic topic modeling and clustering. Combining the two techniques helped the search engine to improve its efficiency and accuracy. The effectiveness of the search engine increased by grouping the web services semantically and reducing the search space. The semantic matching between the service descriptions and the query improved the accuracy and significantly increased the precision and recall rating of the proposed search engine (WSSE). The proposed search engine (WSSE) could successfully capture the hidden topics of any type of query without having to specify the keywords as is the norm in the keyword-based approaches. The accuracy of clustering was compared between the proposed approach and keyword-based discovery approaches. Also, the accuracy of discovery was compared between the proposed search engine (WSSE), programmableweb.com and text-based search system. The results showed that the clustering methods based on LDA outperformed the clustering methods based on TF-IDF. Also, in the proposed approach, the probabilistic topic modeling and clustering was used to discover web services. The proposed search engine (WSSE) was evaluated by calculating the precision@k. The comparison of precision@K values showed that the proposed search engine (WSSE), based on the K-means clustering method and LDA, performed better than other keyword-based approaches. Based on the experiments, the probabilistic topic modeling proved its ability to improve the clustering performance and the quality and efficiency of web service discovery. Moreover, integrating the clustering method and the topic modeling makes the discovery of the more relevant web services rather easier and faster than searching in the whole corpus.

6. REFERENCES

   http://www.programmableweb.com/.
European Conference, ESOCC 2013, Málaga, Spain, September 11-13, 2013. Proceedings, chapter
Probabilistic Topic Models for Web Services Clustering and Discovery, pages 19–33. Springer Berlin
Tagging Augmented LDA for Web Service Clustering, pages 162–176. Springer Berlin Heidelberg, Berlin,
Heidelberg, 2013.
web service discovery. In Web Services (ICWS), 2012 IEEE 19th International Conference on, pages
448–455, June 2012.
analysis of multivariate observations. In Proceedings of
the Fifth Berkeley Symposium on Mathematical
Statistics and Probability, Volume 1: Statistics, pages
objective function. Journal of the American Statistical
qos-aware web service selection algorithm based on
clustering. In Web Services (ICWS), 2011 IEEE
International Conference on, pages 428–435, July
2011.
[19] P. Xie and E. P. Xing. Integrating document clustering