CAPABILITY LANGUAGE PROCESSING (CLP): CLASSIFICATION AND RANKING OF MANUFACTURING SUPPLIERS BASED ON UNSTRUCTURED CAPABILITY DATA

by

Kimia Zandbiglari, B.Sc.

A thesis submitted to the Graduate Council of
Texas State University in partial fulfillment of
the requirements for the degree of
Master of Science
with a Major in Engineering Management
May 2022

Committee Members:

Farhad Ameri, Chair

Jaymeen Shah

Meysam Khaleghian

FAIR USE AND AUTHOR'S PERMISSION STATEMENT

Fair Use

Duplication Permission

DEDICATION

To Mohammad, my husband, who unconditionally supported me all the way even from thousands of miles away, to my beloved family, whom I will forever be indebted to, and to all strong, confident, and courageous girls who embrace challenges, persist during setbacks, and believe that being brave is more valuable than being perfect,

this work is dedicated.

ACKNOWLEDGEMENTS

No words could express my deepest gratitude to my supervisor **Prof. Farhad Ameri**, a great scholar who introduced me to the world of ontology and natural language processing, always patiently encouraged me and taught me unforgettable lessons about research the academic life. I strongly believe that without his help, this work would not have been completed, and I continue to count on him as an inspiring scholar.

Special thanks most also be sent to my advisory committee **Prof. Khaleghian** and **Prof. Shah** for their most constructive instructions and guidance throughout this work.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

LIST OF ABBREVIATIONS

Abbreviation	Description
BoC	Bag of Concepts
BoW	Bag of Words
CLP	Capability Language Processing
CM	Concept Model
CMB	Concept Model Builder
CSV	Comma-Separated Value
EC	Entry Concept
EE	Entity Extractor
F	F-Measure
IC	Information Content
IDF	Inverse Document Frequency
KNN	K-Nearest Neighbor
lch	Leacock Chordorow
LCS	Least Common Subsume
LSA	Latent Semantic Analysis
Maas	Manufacturing-as-a-Service
MCT	Manufacturing Capability Thesaurus
MSP	Most Specific Parent

NGD	Normalized Google Distance
NLP	Natural Language Processing
OEM	Original Equipment Manufacturers
P	Precision
R	Recall
RF	Random Forest
SKOS	Simple Knowledge Organization System
SME	Small and Medium Enterprises
SVD	Singular Value Decomposition
SVM	Support Vector Machine
TLP	Technical Language Processing
wup	Wu and Palmer

ABSTRACT

In manufacturing industry, data is available in both structured and unstructured forms. Although the unstructured data represented in natural language text contains valuable information and knowledge, its effective processing for the sake of information retrieval and knowledge extraction is a challenge. Manufacturing Capability data is an example of unstructured data widely used for describing the technological capabilities of manufacturing companies. The objective of this research is to use a set of text analytics techniques to enable automated classification and ranking of manufacturing companies based on their capability narratives available on their websites. For this purpose, a supervised classification method is used in conjunction with semantic similarity measurement method. A formal thesaurus that uses Simple Knowledge Organization System (SKOS) format provides structural and lexical semantics to support classification and ranking. To conduct semantic similarity measurement, edge-based method is combined with Normalized Google Distance (NGD) technique to create a weighted edge-based method for measuring the similarities of manufacturers' capabilities with the queried capabilities provided by customers. The proposed framework is validated experimentally using a hypothetical search scenario. The results indicate that the generated ranked list is highly correlated with human judgment, especially if the query model and supplier capability model belong to the same class. However, the correlation decreases when multiple overlapping classes of suppliers are mixed. The findings of this research can be used to improve the precision and reliability of Capability Language

Processing (CLP) tools and methods and improve the intelligence of supplier discovery and capability mapping platforms.

1. INTRODUCTION

1.1. Background and Motivation

Due to rapid digitalization of manufacturing industry, the volume and diversity of data, in both structured and unstructured forms, is growing exponentially. The focus of this thesis is on manufacturing capability data that describe the production process, material, quality capability, and engineering capabilities of manufacturers. Manufacturers use various forms of data, including structured and unstructured natural language, to describe their manufacturing capabilities. Particularly, companies' websites are often used as the primary venue for advertising manufacturing capabilities. Most of the information found on manufacturing suppliers' websites is human generated which presents data in the form of unstructured natural language text. The unstructured data is a valuable source of highly important capability data. While querying and searching structured data is a relatively mature and efficient process, unstructured data in the form of natural language presents several challenges with respect to search, information retrieval, and knowledge extraction.

Keyword search is the de facto method for retrieving information from unstructured text. However, keyword search often results in lower precisions since it often ignores the semantics of data. In this thesis, the focus is on semantic search methods. As opposed to keyword search, semantic search considers meaning and context instead of only exact matches of a word.

This research is motivated by the need for improving the intelligence of supplier discovery process. A supplier evaluation system that is supported by text analytics and machine learning algorithms can provide manufacturers with a better visibility into the

strengths and weaknesses of the other manufacturing suppliers and positively impacts their decisions about their prospective business partners. The methods and models developed in this research can support supply chain decisions during the early stages of supply chain formation process. The goal is to help supply chain managers select appropriate manufacturing partners based on their capabilities as described on their public websites. Traditional keyword-based techniques for search have several limitations and typically the returned results have very low precision. More sophisticated techniques need to be used to ensure the search process returns a set of relevant manufacturing suppliers that can fulfill the required manufacturing services. Various supervised and unsupervised text analytics techniques can be used for this purpose. One of the techniques that can effectively organize the data into various categories is classification . Classification (Bhavsar & Panchal, n.d.) is a machine learning technique which is widely used in different industries to classify data in various classes. Classification techniques identify group membership for data instances.

Typically, the supplier groups that are formed after classification steps ate still very large in size and manual sorting and analysis of the members of each group is very time consuming. Therefore, suppliers within each group need to be ranked according to their relevance to the search criteria. Keyword matching can be used as a technique for ranking the results. However, since keyword matching disregards the semantics of the search terms, the outcome often suffers from low precision and recall. This research is addressing this issue by implementing a semantic approach to measuring similarity and relatedness.

1.2. Problem Statement

A typical supplier search and assessment process often entails the evaluation of a large number of small-to- medium sized manufacturers (SMMs) which are potentially capable of providing the requested services. However, the diversity of this supply pool negatively affects the efficiency of evaluating and selecting manufacturers while searching for qualified suppliers in this pool. A large volume of unstructured data can be found on manufacturers' website, which contain valuable information about suppliers' capabilities. If manufacturing capabilities available on suppliers' website are analyzed and evaluated, more accurate decision will be made when selecting supply chain partners. The objective of this research is to use text analytics techniques to compare, evaluate, and analyze the capabilities of manufacturing companies based on the unstructured data available on their website.

1.3. Research questions

This research work is intended to answer the following questions:

- Which classifier will have the best precision/recall/ F-measure in this work?
- What method of similarity measurement should be used for the ranking?
- How can classification affect similarity scores?
- How to compare automatically generated rankings with human expert's rankings?

1.4. Assumptions, Limitations, Delimitations

1.4.1. Assumptions

- It is assumed that manufacturers might use informal terminology and jargons for describing their capabilities. The reason for this assumption is that

practitioners in the manufacturing domain do not always follow the scientific terminology often found in manufacturing reference books and technical documents.

- It is assumed that all information that suppliers have provided on their website is true and indicates their real capability.
- It is assumed that the sourcing web-portal (Thomas Net) that is used for collecting supplier data has categorized suppliers correctly. Thomas Net includes multiple capability categories and there exist hundreds of contract manufacturers under each category.

1.4.2. Limitations

- Data can only be collected from suppliers' websites, and it is limited to what they have published as their website content. It is not possible to have access their internal and proprietary information that is not published publicly.
- Limited and various number of suppliers providing services in different manufacturing classes, is a limitation to collect adequate data.
- Lots of suppliers' website URLs are protected against being crawled, which makes it impossible to used Entity Extractor feature on SKOS Tool.

1.4.3. Delimitations

- The research only uses suppliers' websites which are in the USA and have their content in English for data collection, as the source portal only provides categorized suppliers in the USA.
- In this work numeric data such as tolerances and dimensions are excluded, as the manufacturing capability thesaurus primarily addresses qualitative

manufacturing capability characteristics and excludes quantitative characteristics.

- This research only studies manufacturing suppliers and excludes logistics service providers since this research mainly focuses on manufacturing capabilities.

1.5. Research Methodology

To unlock the value of the unstructured capability data, there is a need for developing advanced quantitative techniques supported by semantic modeling, machine learning, Natural Language Processing (NLP), and statistical inference methods. We refer to the pipeline of capability text analysis tools and methods as Capability Language Processing (CLP) which is a branch of Technical Language Processing (TLP). TLP is a human-in-the-loop iterative approach to tailor NLP tools to engineering data (Brundage et al., 2021) .TLP seeks to re-imagine the out-of-the-box NLP pipeline (including tokenization, stop-word removal, cleaning, and stemming) since they are historically designed with non-technical language in mind. Capability Language (CL) is a highly technical language containing specialized vocabulary, jargons, and tribal knowledge, and the data processing and analysis techniques that use capability language as the input need to be supported by specialized resources for data preparation, annotation, and representation.

This research uses a hybrid approach that involves classification and ranking of manufacturing suppliers based on the textual data available on company websites.

Bag of Concepts (BoC) method is used for supervised classification of manufacturing suppliers in which a dictionary of manufacturing concepts is required. In

this research Manufacturing Capability Thesaurus (MCT) which contains manufacturing capability concepts is used. MCT is created using a web-based tool called SKOS Tool that is developed in the Engineering Informatics lab. The created thesaurus uses semantic relationships and links and organizes capability concepts in this regard. It includes 8 concepts schemes and manufacturing capability concepts are added under these concept schemes. Once target classes are defined, for each target class, a set of features have to be chosen which demonstrate relative features to each target class. In addition, as not all the concepts are at the same level of importance in describing the features of a target class, they need to be weighted. These weighed sets are called Concepts Models which SKOS Tool is able to create using Concept Model Builder feature.

To collect data, manufacturing suppliers' websites contents are extracted to generate the raw data. These data are used as the input to Entity Extractor feature in SKOS Tool that can create a concept vector for each supplier. A concept vector includes the concepts and their frequencies for each supplier in a comma-separated value (CSV) format. The generated concept vectors for individual supplier can be unified in a CSV file to form a matrix. Manufacturing suppliers' websites in two target classes are reached out using Thomas Net sourcing portal[1] .

Once concepts model and unified suppliers file are available, using known classifiers, the experiment can be performed. Decision Tree, Support Vector Machine (SVM), K-Nearest Neighbor (KNN), and Random Forest (RF) are the classifiers which will be used in this experiment to classify suppliers into two pre-defined target classes.

[1] Thomasnet.com

Precision, recall, and F-measure are the three metrics that will be used to evaluate each classifier's accuracy.

According to the motivating use case for the proposed manufacturer recommendation framework, the user submits a query that is composed of a vector of thesaurus concepts that collectively represent the required manufacturing services and the desired capabilities of supplier of those services. Since the manufacturers are also represented as concept vectors extracted from the same thesaurus, the similarity measurement between the requested capabilities and provided capabilities by each manufacturer is reduced to a set of pairwise similarity measurements between requested and provided capability concepts.

A hybrid approach using both corpus-based and knowledge-based methods is adopted in this work for semantic similarity measurement between concept pairs. While the edge-based method takes into account the inheritance relationships between the concepts as an indication of their semantic similarity, the Normalized Google Distance (NGD) approximates the semantic relatedness of concepts based on their co-occurrences in a large corpus. In the proposed hybrid method, the NGD between two concepts will be used as the weight of the edges on the path between these concepts.

An example query will be formulated for this experiment that contains a set of concepts referring to different aspects of manufacturing capabilities that can be provided by qualified suppliers. The query will be formulated such that it targets suppliers from one of the target classes. The overall similarity score for each category is the average of the pairwise similarity scores obtained for that category. In this way, the length of the concept vector for each supplier will not inflate the overall similarity score of the

category. The overall similarity between the query and the supplier can be calculated as the sum of similarity values calculated for each capability category. Using the existing equations for proposed methods, the similarity score associated with each supplier will be computed in Python environment.

When all suppliers' similarity scores are available, they will be ranked based on their score. In addition, to evaluate the accuracy of the ranking step, its output will be compared with the ranking provided by a human expert. The human expert will be provided with suppliers' website URL and will be asked to rank suppliers based on their similarity to the formulated given query. Once both rankings of similarity measurement and human expert are done, Spearman's ranking correlation will be used to compute the correlation between similarity measurements ranking with rankings generated by the expert.

Figure 1 shows the main steps of the proposed framework for supplier classification and ranking. The raw data, i.e., capability narratives, are extracted directly from the website of suppliers and typical pre-processing and data cleaning steps are conducted. The next step is tokenization or separating the text into meaningful terms and phrases for each supplier. Then the suppliers are classified into several predetermined capability classes such as suppliers with precisions machining capabilities or suppliers with heavy and large part machining capabilities. Similarity measurement step entails measuring the similarity between the capabilities advertised by a supplier and the capabilities requested by a customer. Since the text is decomposed into tokens, the similarity measurement is boiled down to measuring pair-wise similarities between relevant terms and phrases. For example, if a supplier's industry focus is *Automotive*

Industry but the queried industry focus is *Aerospace Industry*, then similarity between aerospace industry and automotive industry needs to be measured as one of the components of the overall similarity score. The final outcome is a ranked list of suppliers based on their degree of similarities with the queried capabilities. Both classificational and similarity measurement steps use the Manufacturing Capability Thesaurus (MCT) as one of the inputs. It should be noted that in this work we are mainly concerned about the *semantic similarities* between the terms. In absence of a formal ontology that can provide formal and axiomatic semantics for a term, a formal thesaurus that encodes structural and lexical semantics is used in this work. Figure 1 shows an overview of the classification and ranking framework proposed in this work.

Figure 1: The Overview of the Classification and Ranking Framework

1.6. Research Plan

Work in this research is divided into tasks as described below:

- **Task 1. Literature Review on Text Classification and Semantic Similarity Measurement:**

  Reviewing text classification and semantic similarity measurement papers to discern and recognize problem is the purpose of this task. In this task using Google Scholar search engine, related keywords to the topics have to be searched and by reviewing more than 60 papers, appropriate and most related papers to this research have to be selected.

- **Task 2. Modeling and Implementation:**

  In this task, based on the gathered knowledge from previous task, a proper model which can fill the gaps and solve the problem has to be created. This step includes finding the best similarity measurement method and equation to solve the problem.

- **Task 3. Thesaurus Extension:**

  The thesaurus is built in bottom-up manner through tagging the capability-related term on manufacturing suppliers' websites. In this task, thesaurus will be extended by tagging previously untagged capability-related terms on manufacturing suppliers' websites.

- **Task 4. Data Collection:**

  In this task, using Thomasnet.com suppliers of each of these 2 (or more) classes will be collected using their website content and SKOS Tool entity extractor

feature created by the Engineering Informatics lab at Texas State University and collected data will be unified in a CSV file.

- **Task 5. Experiment:**

  In this task, gathered data and created model will be used to run the experiment. The experiment will include the two phases of classification and ranking. The results obtained through classification and ranking will be compared with the ranking generated by human expert.

- **Task 6. Analysis:**

  When experiment is done, the experiment result will be analyzed in this task. Analysis of the experiment result will provide a roadmap for improving the model. In addition, in this task, suggestions for future work will be provided.

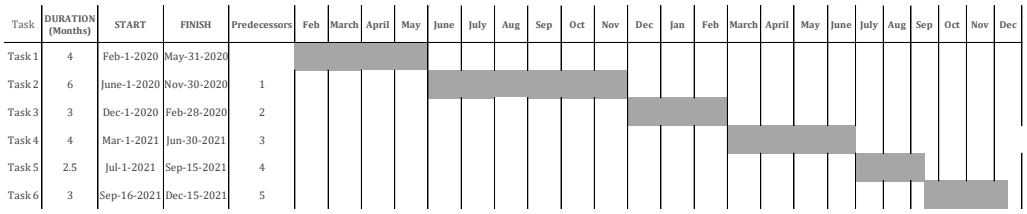

Figure 2: Research plan Gantt chart

2. CAPABILITY TEXT CLASSIFICATION

2.1. Document Classification

Classification is the process of categorizing inputs under pre-defined classes or categories based on shared features of the input elements such as concepts.

Document classification is related to the process of categorizing, labeling, and tagging items and assigning them to defined categories in databases based on their content to ease the procedures of managing, searching, and analyzing the documents.

One of the applications of document classification is text classification which is the concern of this work. Text classification involves the process of classifying textual document using some text analysis techniques in different levels of document, paragraph, sentence, and subsentence.

Three different approaches can be adopted in text classification based on the application text classification is doing for:

- **Supervised Text Classification:** In supervised text classification, a set of training data and classes (labels) are available and controlled by human. Model can learn from the training data and perform classification on other sets of data.
- **Unsupervised Text Classification:** As opposed to the supervised text classification, this approach puts documents into different clusters without any prior training.
- **Semi-Supervised Text Classification:** Semi- supervised text classification falls between supervised and unsupervised text classification. Semi-supervised text classification uses a self-training mechanism to learn from small amount of labeled data to label a vast amount of unlabeled data.

2.2. Capability Text Classification

In the initial stages of supply chain building, capability analysis is required. While the majority of capability analysis methods use structured data, manufacturing suppliers' website present their data in form of unstructured data and natural language form. Figure 3 shows an example of unstructured data that described the capabilities of a manufacturer with specialty in CNC machining of large parts.

As a high volume of information and knowledge is stored as text in manufacturing suppliers' website, text analytics techniques can be used for extracting useful knowledge patterns and insights. Manufacturing capability data can be found through various sources and formats; yet the largest proportion of data is in unstructured and semi-structured form which is the reason why using text mining approaches including text classification can be beneficial (Korde, 2012).

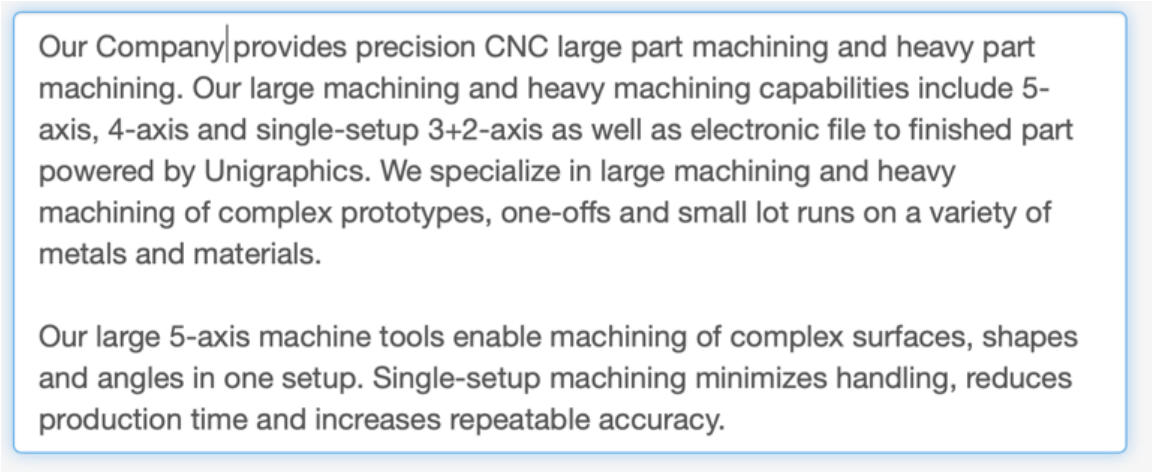

Figure 3: An example of unstructured manufacturing data

This work uses text classification approaches for manufacturing suppliers' capability classification. Text classification techniques are used to classify suppliers under pre-defined capability classes. One step in the text classification is the feature extraction. As text and documents are in the form of unstructured data, this data needs to

be converted into structured forms of data by using mathematical modeling. In the first stage of feature extraction, data needs to be polished to be free of nuisances. Feature extraction can be applied after data cleaning (Kowsari et al., 2019).

A thesaurus guided method is adopted in this work. Manufacturing Capability Thesaurus is built in a bottom-up manner through tagging relevant terms which are semantically and lexically interconnected. The thesaurus contains capability related terminologies used in the manufacturing industry. The thesaurus provides the concepts (features) associated with each manufacturing capability class. The documents (capability narratives collected from websites) are also transformed into vector of terms to provide the input to the classifier. The classifiers that are often used for text classification are: Decision Tree, Support Vector Machine (SVM), K-Nearest Neighbor (KNN), and Random Forest (RF).

2.3. Related Works in Text Classification

One of the first steps in unstructured data processing is tokenization or separating the text into meaningful terms and phrases.

Wang et al. addressed the words with different meanings (polysemy) problem using bag of concepts instead of traditional bag of terms technique (F. Wang et al., 2014). Their proposed mechanism has two stages. In the first stage, a conceptual model for each target category is generated using a large knowledge-based thesaurus. Then phrases and short texts are classified based on their similarity.

Dong and Liu studied classification of enterprise websites using a technique based on support vector machine and topic feature modeling (Dong et al., 2010). They exploited a multi-feature topic vector generated by the website's textual content and content

structure to determine the category of the website. Their method was validated by conducting an experiment on manufacturing enterprise website search.

(Shotorbani et al., 2016) offered a method using clustering and topic modeling to enhance searching and organizing textual documents and extract valuable patterns from manufacturing websites. The method illustrated topic modeling along with document clustering, boost annotation, and classification of manufacturing supplier's webpages, which helped users to extract valuable patterns from supplier's websites (Shotorbani et al., 2016).

Assessing factories' readiness for implementation of technologies was proposed by Jung et al. The result of this research can support creation of smart manufacturing systems. Evaluating the companies and providing users with the status of the current target company's readiness level in comparison to the reference model is the basis of this method. Knowing the current state helps companies improve their readiness level that can have a positive correlation with companies' operational functions (Jung et al., 2017).

2.4. Capability Text Classification

Text classification is the process of categorizing text under pre-defined classes according to its content (Korde, 2012). An automated text classification for manufacturing suppliers provides the opportunity of using the untouched data as published by the manufacturing suppliers. Automatic text classification can be conducted using both rule-based and machine learning methods. This work uses supervised machine learning in which pre-classified text is used to train the classifier. Once trained, the classifier can make a prediction about the class of an unlabeled text. One of the first steps in supervised classification is the feature extraction step, where the text is transformed

into a vector representation (F. Wang et al., 2014). This step can be done either manually or automatically through machine learning. Domain experts will precisely extract features associated with each class of interest in a timely manner. On the other hand, automated feature extraction consumes less time and effort by satisfying the consideration of semantic relationships of the features. Feature extraction strategies do not change the variables' original representation, but rather choose a subset of them. As a result, they preserve the original semantics of the variables, allowing a domain expert to interpret them (Saeys et al., 2007). A semi-automatic feature extraction is used in this research supported by a formal thesaurus that uses SKOS (Simple Knowledge Organization System) for its syntax and semantics.

2.5. Manufacturing Capability Thesaurus (MCT)

The Manufacturing Capability Thesaurus (MCT) is the core of the proposed classification method. MCT provides a concept schema for manufacturing capability terms. The MC Thesaurus comprises the common terminologies that are frequently used to describe a supplier's manufacturing capabilities. One complication is that every industry has a limited set of vocabulary that is only meaningful to those who work in that field. A thesaurus can be used to reduce terminological obscurity by semantically integrating seemingly diverse terms in different subcategories of the manufacturing industry. Terms as Articulating Arm, Coordinate Measuring Machine, and CMM may be used in MCT to describe suppliers' capability with respect to production resources. A thorough thesaurus of manufacturing capability enables each website to be translated to a vector model (Ameri & Bernstein, 2017). MCT is built in a bottom-up manner (by selecting terms from raw text and classifying them under appropriate broader concepts)

through tagging the capability-related terms found on manufacturing suppliers' websites in a corpus and connecting them using semantic and lexical relationships. The thesaurus is built with 13 Concept Schemes, namely, Accreditation and Certification, Capacity and Variety, Human Resource, Industry, Input/Output Artifact, Manufacturing Process, Material, Organization, Product and Process Attribute, Production Capability, Production Resource, Service, and System. Each concept scheme forms a taxonomy of concepts structured through parent-child relationships.

2.5.1. SKOS Tool

MCT uses Simple Knowledge Organization System (SKOS[1]) for its syntax and semantics. SKOS is a W3C standard that is recommended for creating thesauri, concept schemes, and taxonomies by providing an interoperable framework and formal annotation vocabulary. SKOS thesauri are concept-based, as opposed to term-based, in nature. In a term-based thesaurus, terms are directly connected by lexical relationships whereas, in a concept-based thesaurus, semantic connection is at a concept level and terms are the lexical labels for the concepts.

Concept is defined as a unit of thoughts which is based on the common characteristic of the objects or experiences which belong to a category and is able to fragment a vast amount of information (Gray & F. Bjorklund, n.d.).

Each concept in SKOS has exactly one *preferred label* (skos:prefLabel) and can have several *alternative labels* (skos:altLabel) which are the synonym terms frequently used pointing to the same concept.

SKOS thesaurus has a three-level structure:

a) Conceptual level, which identifies concepts and establishes interrelationships of concepts.
b) Terminological correspondence level, which allocates terms to their respective concepts (preferred or alternative labels)
c) Lexical level, which defines concepts interrelationships (i.e., broader, narrower, related).

Narrower labels (skos:narrower) indicate a more specific form of their broader labels (skos:broader) having a hierarchical link, and the associative relationships are defined through related labels (skos:related) (Ameri et al., 2020). A SKOS thesaurus forms a knowledge graph which can be enriched continuously to support various data-driven and knowledge-intensive applications. Figure 5 shows the concept diagram of the molding sand based on the SKOS terminology.

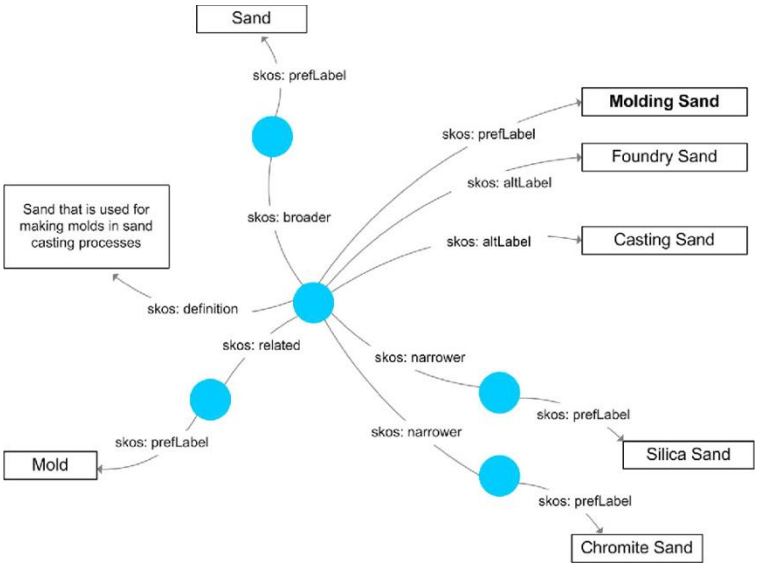

Figure 4:The concept diagram of the Molding Sand based on SKOS terminology (Ameri et al., 2020)

2.5.2. Thesaurus Development and Extension

The MC Thesaurus is built in a bottom-up manner by tagging capability-related terms and phrases on manufacturing suppliers' websites. Thesaurus terms are only added

when they are frequently used. Using the bottom-up technique enables the thesaurus to capture well understood informal vocabularies in the manufacturing industry. It is worth noting that the generated thesaurus focuses primarily on qualitative characteristics of industrial capability. The numeric and quantitative characteristics, same as tolerances and dimensions, are excluded in building the thesaurus, as there is not a fixed list available for them and they vary from supplier to supplier and product to product. The MC Thesaurus currently contains more than 990 concepts designated by about 1,300 labels that are categorized under 13 concepts schemas.

Tagging relevant words in a bottom-up manner can be done through SKOS Tool by Tagging the candidate terms to the most relevant concept schemes manually by identifying the most appropriate broader concept as shown in Figure 5.

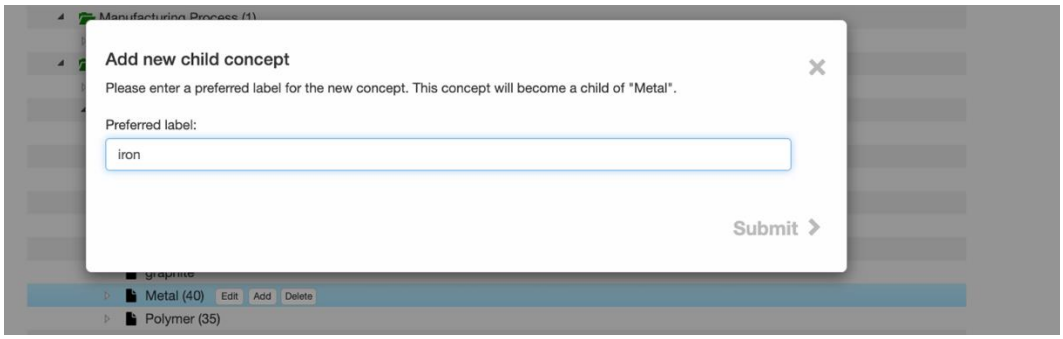

Figure 5: Tagging candidate terms manually to develop the thesaurus

Term selector feature provided in the SKOS Tool is another method of tagging relevant concepts and extending the thesaurus. This feature identifies tagged terms in a plain text, website URL, or in a CSV file which enables the human expert to identify and tag new relevant text by simply choosing the most related parent. Figure 7 shows how term selector feature adds new relevant terms to the thesaurus. Terms highlighted in red and green already exist in the thesaurus.

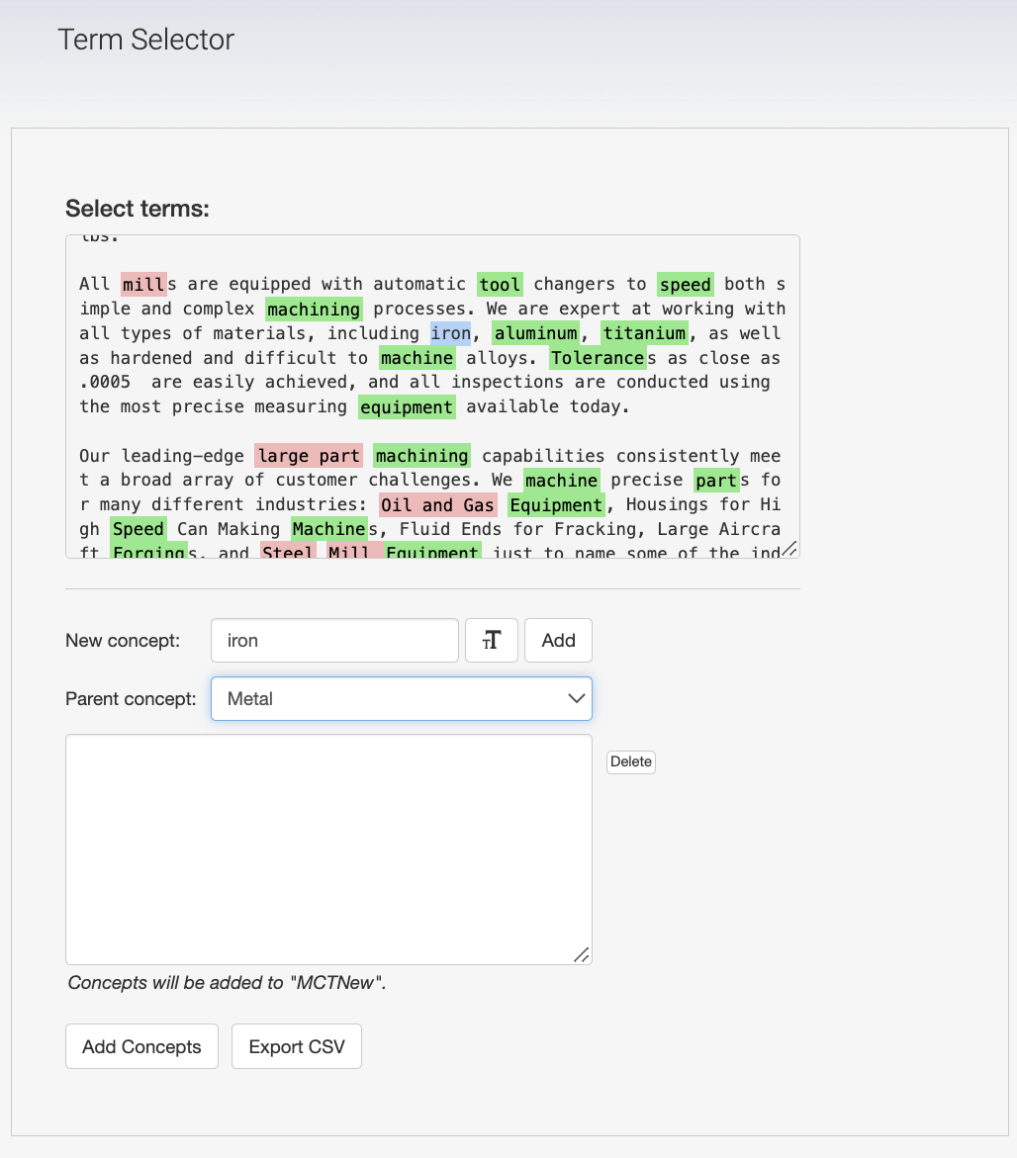

Figure 6: Term selector feature and relevant terms to the thesaurus

The following stages might be used to further explain and define the integrated concepts:

1-Providing a textual definition of the concept

2-Providing alternative labels for the concept, if possible

3-Linking the concept with the other related concepts (both internal and external)

2.5.3. Capability Model in MCT

As a result of the bottom-up term tagging and concept integration described in the previous section, numerous categories of concepts (or concept schemes) were created:

Accreditation and Certification, Capacity and Variety, Human Resource, Industry, Input/Output Artifact, Manufacturing Process, Material, Organization, Product and Process Attribute, Production Capability, Production Resource, Service, and System. Figure 7 shows the distribution of concepts in the thesaurus related to each concept scheme.

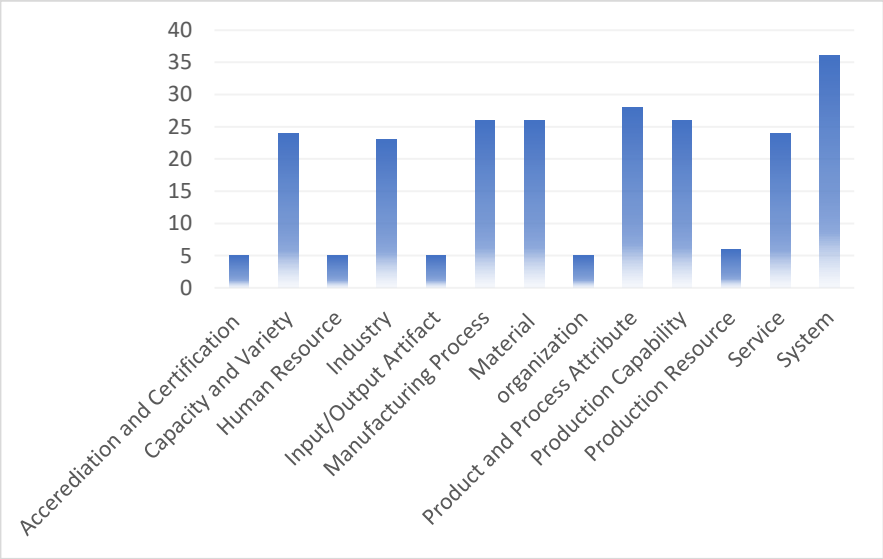

Figure 7: Distribution of concepts included in each concept scheme

Accreditation and Certification: Quality certifications, quality awards, quality control, industry-wide accreditations and inspection methods and tools, and other terms relating to quality, accreditations, and inspection are covered by this schema.

Capacity and Variety: This concept scheme includes production capacity, production scope, and production variety that a manufacturer can provide.

Human Resource: The capabilities relating to human resources and occupation are described in this concept scheme.

Industry: This concept scheme incorporates concepts that explain the market categories and industries supplied by the company, such as defense, automotive, and aerospace.

Input/Output Artifact: This concept scheme contains both digital and physical artifacts that a manufacturer can process or manufacture.

Manufacturing Process: This concept scheme includes manufacturing processes a manufacturer may be capable of.

Material: Material concepts scheme includes material related concepts of material family and named material.

Organization: The capabilities relating to organization type are brought under this concept scheme.

Product and Process Attribute: This concept scheme includes geometric features, min-max limits, part attribute, and process attribute.

Production Capability: Capability concepts related to complex part machining, large part machining, precision machining, and work holding machining are the concepts under this concept scheme.

Production Resource: Equipment, facility, machine, and tool are covered under production resource concept scheme.

Service: Engineering service, logistic service, test and inspection service, and tool making service are included in this concept scheme.

System: This concept scheme contains concepts related to the production support system, production system, quality system, and software system.

2.6. Manufacturing Text Classification

In this section, the proposed method of classification is described.

2.6.1. BoW vs. BoC

Most conventional text classifiers use an approach known as Bag of Words (BoW) where a document is represented as a vector of words with their frequency of occurrence in the document. The words in BoW come from a dictionary or vocabulary generated automatically from the collection of provided text. One major drawback of BoW approach is that it does not retain the semantics of the original phrases (or N-grams) when they are decomposed into single terms (1-grams). For example, the phrase *Swiss Turning* designates a specialized turning process for small and intricate parts is decomposed into *Swiss* and *Turning*, none of which convey the precise meaning of the Swiss Turning process (Ameri & Bernstein, 2017). To remedy the semantic degradation issue in the BoW approach, Manufacturing Capability Thesaurus (MCT), as a hand-crafted controlled vocabulary, is adopted in this work to substitute the dictionary of terms used in conventional methods of text classification (Sabbagh et al., 2018). MCT contains relevant concepts pointing to different aspects of manufacturing capability. The feature or concept vectors generated during the feature extraction process directly use the concepts from MCT without decomposing them into atomic terms. We refer to this alternative method as the Bag of Concepts (BoC) technique. It has been demonstrated that BoC results have better accuracy compared to the BoW technique when classifying manufacturing capability text (Sabbagh & Ameri, 2018).

2.6.2. Classification

Manufacturing suppliers are classified indirectly by classifying the text that describes their capabilities. Text classification is conducted in two phases: (1) Concept Model Generation (offline phase); (2) Test Document Classification (online phase) as shown in Figure 8.

The first phase results in generating a *concept model* associated with each class. The concept model is essentially a weighted vector of concepts representing the class of interest. During the second phase, the classifier is trained to identify manufacturer capability class for each manufacturer based on its capability narrative. These two phases are discussed in more detail in the following sections.

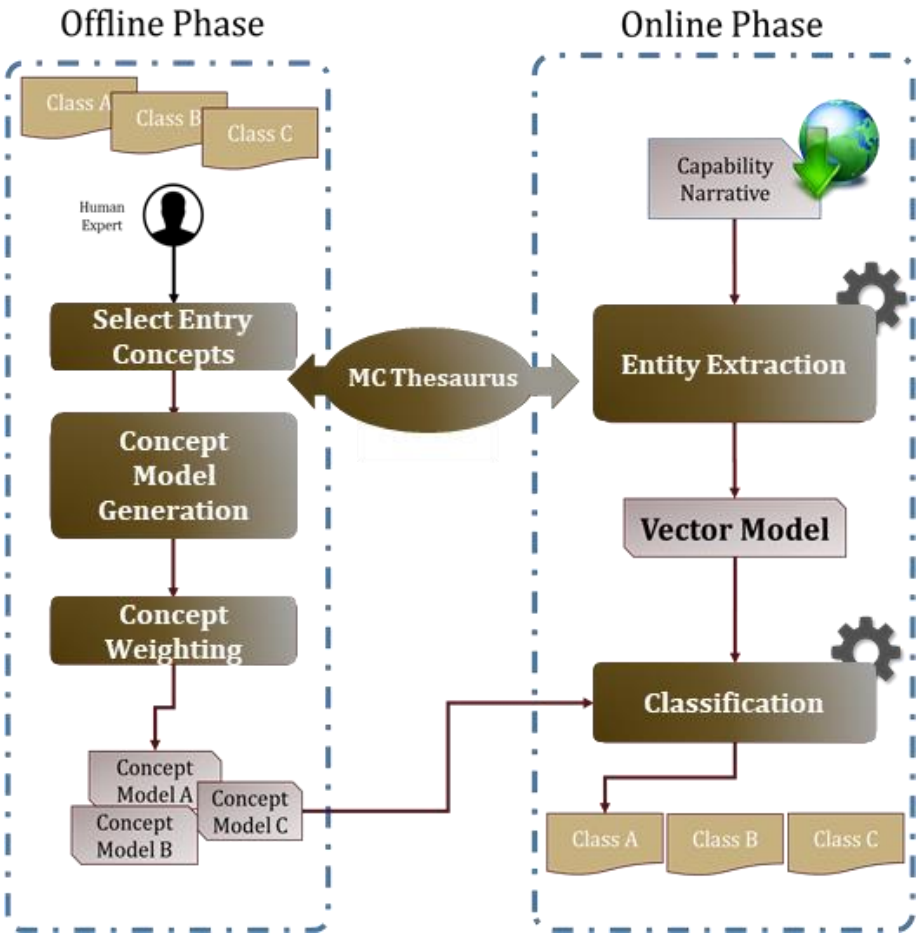

Figure 8: Proposed Manufacturer Classification Framework

2.6.3. Concept Model

Entry Concepts (ec_i): Concept Models are formed by submitting relevant SPARQL queries to the thesaurus. To build a concept model, the first step is to identify a few entry concepts (ec_i) that are closely related to the target class. Entry concepts are selected by a human expert since background knowledge is needed for identifying the distinguishing concepts for each class of interest. For example, in Heavy and Large Machining capability class, a possible entry concept set can be EC = {ec_1= *Heavy Machining, ec_2= Large CNC Machining, ec_3= deep hole machining, ec4 = Large Part, ec4= Vertical Milling Machine*}. The Human expert will identify entry concepts from mid-level concepts which are interrelated with other concepts existing in the thesaurus. The SPARQL query is then submitted to the thesaurus to retrieve the broader, narrower, and related concepts for each entry concept with adjustable depths. By identifying the entry concepts by the human expert, concepts models associated to each capability class will be built around the entry concepts.

Weighting System for Concepts: As not all the concepts have the same level of importance for each particular capability class, the level of significance of each concept is specified using a weighting schema as shown in Figure 9.

All returned concepts do not have the same level of importance for the target class. For example, according to the concept weighting schema used in this work, for an entry concept, the weight assigned to preferred labels of entry concepts is "9" while the alternative labels receive "5" as their weights for entry concepts as the preferred label has more importance than the alternative label(s) based on this schema. In this regard, any related concept, narrower, or boarder receives a lower weight. Given training data as $D = \{d_1, d_2, \dots, d_n\}$ for the class CL_i, the weighted concept model will be represented

as: $CM_i = (c_1, w_1], c_2, w_2], \ldots, c_m, w_m])$, in which wi is the weight associated with each concept.

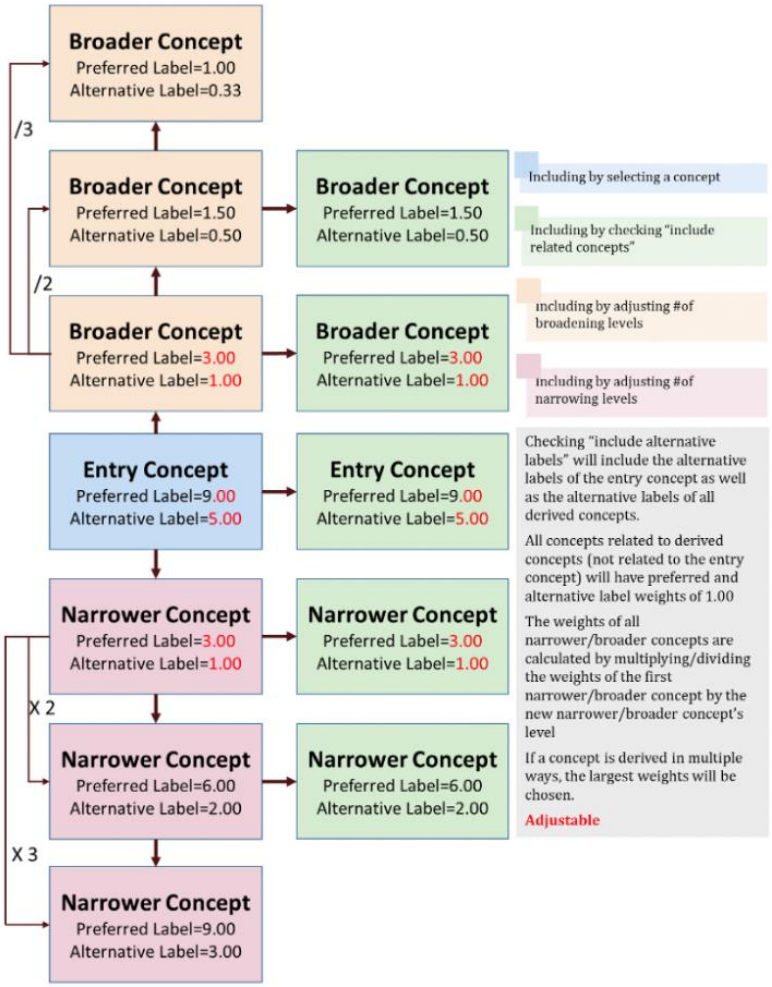

Figure 9: Concept Model Weighting Schema

Concept Model Generation: A Concept Model is a set of weighted concepts that collectively designate a target capability class. In essence, the target capability class describes a desirable supplier through a set of concepts contained in a concept model.

2.6.4. Test Document Classification

In this section, the documents (suppliers' website content) are converted into concept vectors to be classified under the target classes of capability via four common

classification techniques, namely, Decision Tree, Support Vector Machine (SVM), K-Nearest Neighbor (KNN), and Random Forest (RF).

Classification Data: the document's content is directly extracted from the relevant pages of suppliers' websites, and then pre-processed by removing numbers, stop words, and generic words.

Data Conceptualization: In this phase, data related to each supplier's website will be converted to a vector of concepts which represents the supplier's webpage. For this purpose, a custom-made tool, named Entity Extractor Tool, has been built through SKOS Tool. Entity Extractor Tool identifies the thesaurus concepts existing in a document using their preferred or alternative labels. Entity Extractor Tool also calculates frequency of each concept in the given document. Receiving a text document, the Entity Extractor Tool, generate the vector model of a document exported as a Comma-Separated Value (CSV) file, as shown in Figure 10. Entity Extractor tags each term in the text with its matching concept in the thesaurus using their preferred or alternative label. Entity Extractor Tool can also accept supplier's website URL as the input instead of the plain text which makes the process of extracting concepts vector less time consuming.

The level of specificity of the target class, defines the level of abstraction in this process. Three levels of abstraction can be done through Entity Extractor Tool. To be more specific, the highest level of specification, Entity Extractor Tool, considers low-level concepts along with higher- level concepts and top concepts in converting the text to a concept vector.

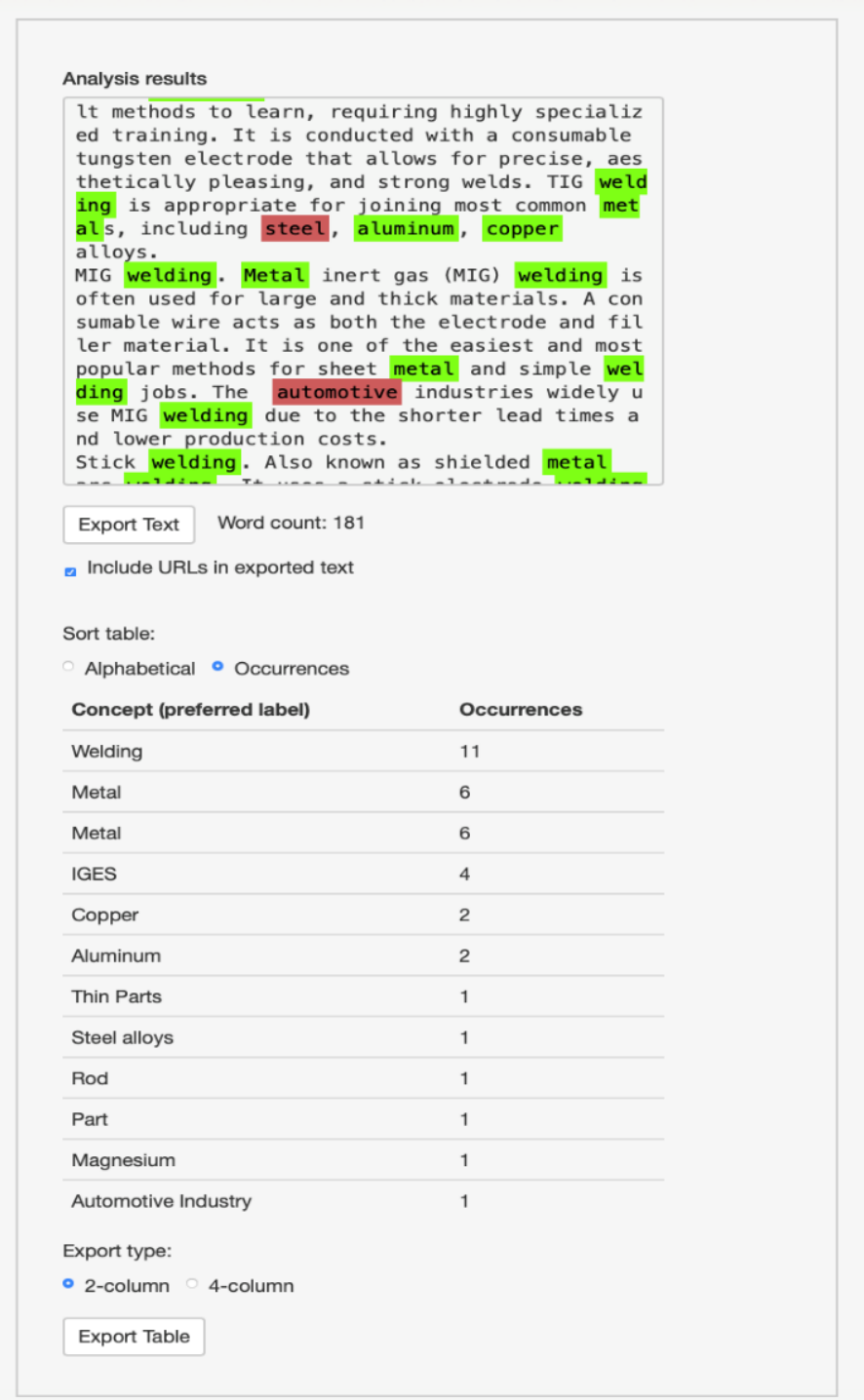

Figure 10: Entity Extractor Tool: Preferred Labels are highlighted in Green and Alternative Labels in Red

Classification: The training and test data in CSV format are provided as the input to the classifier and each supplier is pre-assigned to a capability class. The training data

(concept model for each capability class) and the test data (concept vector for each supplier) that are already converted into CSV format, are the inputs to the classifiers. A classification algorithm is then used to categorize suppliers under capability classes. The process is conducted in Python environment using four classification algorithms of: Decision Tree, Random Forest (RF), Support Vector Machine (SVM), and K-Nearest Neighbor (KNN). In real-life supplier search scenarios, the classified data may still contain a large number of suppliers and the selection process may require significant effort due to the large number of suppliers within each group. Section 3. Classification in the appendix shows the programmed codes related to the classification.

2.7. Experiment Implementation

This section provides the experimental validation results of the proposed method for supplier classification. Two capability classes are used in this research to evaluate the effectiveness of the proposed model. Three information retrieval metrics of precision, recall, and F-measure are used to evaluate the classifier.

2.7.1. Target Capability Classes

The target capability classes were established in the context of machining process capability since the MC thesaurus is plentiful enough in terms of machining concepts. Heavy Component Machining, Precision Electrochemical Machining, Silicon Micromachining, Precision and Complex, General Contract Machining are examples of capability machining classes. Each capability class has a set of concepts which differentiates it from other capability classes. Concepts presenting the capability classes are not unique to a capability class, but the set of concepts is unique for each class of

capability. Precision and Complex Machining and Heavy Component Machining are used in this work as the target class.

Handling and manufacturing of large components for use in heavy machinery, machine tools, and buildings are all part of heavy component machining. Heavy machining, as a result, necessitates specialized tools capable of withstanding the strains and harshness imposed by large components while sustaining precision and tolerance standards. The capability to process items with complicated and geometrically complex features is referred to as Precision and Complex capabilities. Heavy component machining and precision and complex machining are two district capability class; however, it is expected to see some overlaps as well.

2.7.2. Classification of Capability Classes

As discussed before, the classification process has two phases: the offline phase and the online phase. The concepts models for the target classes are generated in the offline phase. During the online phase, the classifier is trained using a training dataset, and then the unlabeled members of the test dataset are classified. Two capability classes, namely *heavy component machining* and *precision and complex machining*, were selected for this experiment. The suppliers in those categories often have specialized equipment, facilities, and expertise represented by a unique vocabulary.

2.7.3. Concept Model Building

For each capability class, a concept model is generated automatically using the Concept Model Builder (CMB) gadget of the SKOS Tool by submitting appropriate queries that are formulated around a few *entry concepts.* Entry concepts are the key

concepts within MCT that represent the class of interest and they are selected by human expert. Table 1 shows the selected entry concepts for both target classes.

Table 1: Entry Concepts of Target Classes

Heavy Component Machining	Precision and Complex Machining
Heavy component	Complex machining
Large part	Difficult machining
Vertical machining center	Live tooling
Deep hole machining	Complex precision part
Heavy machining	Multi-axis capabilities
Large CNC machining	

The SPARQL queries that are executed behind the scenes on Concept Model Builder (CMB), return the broader, narrower, and related concepts for each entry concept based on a user-specified depth level. The final concept model vector is exported as a CSV file containing concepts and their associated weight based on the weighting scheme shown in Figure 9. The CMB user interface is also shown in Figure 11.

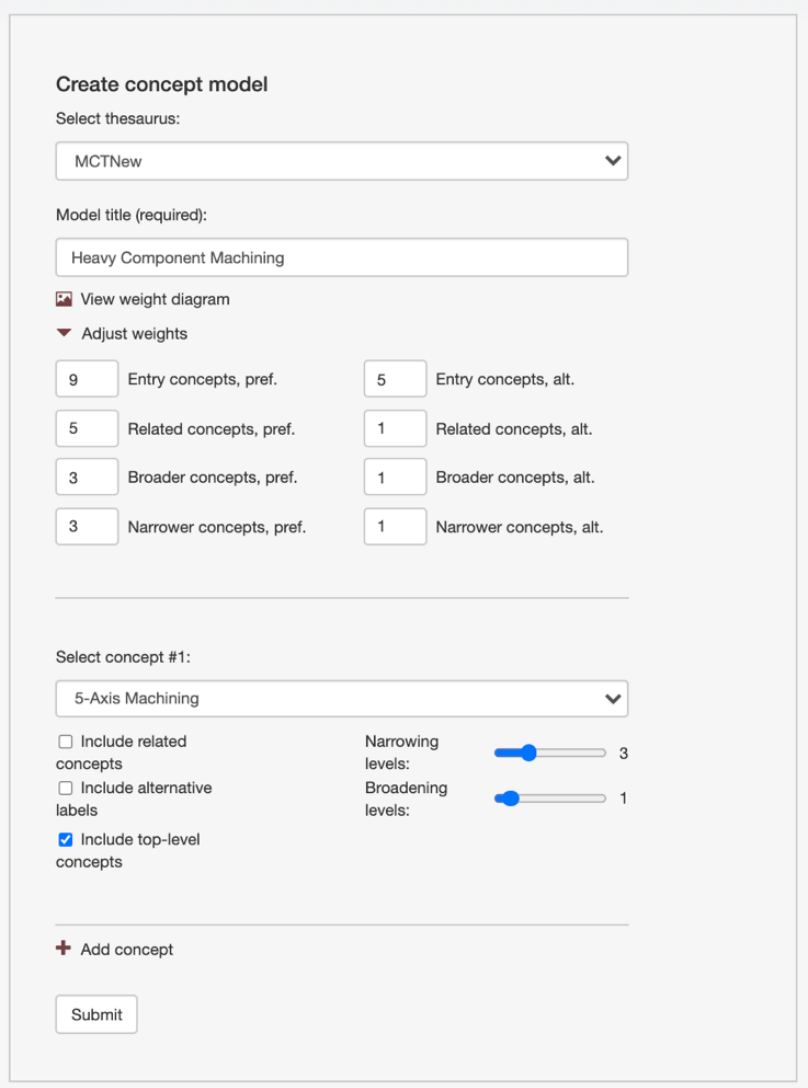

Figure 11: Concept Model Builder Gadget in SKOS Tool

2.7.4. Data Preparation

For each capability class, 130 suppliers were selected from Thomas Net for each class. As a web-based sourcing portal, Thomas Net includes multiple capability classes each containing hundreds of contract manufacturers that have several capabilities related to their respective group. Although every company has a profile on Thomas Net that includes a short textual description of the capabilities and areas of expertise of the

company, the raw text data was collected directly from the company's website rather than their Thomas Net profile for the sake of completeness.

The Entity Extractor feature of the SKOS Tool was used to generate the concept vectors in CSV format for all 260 suppliers participating in this experiment. The Entity Extractor provides the possibility of either entering the URL of the supplier's website or directly inserting the text from the website into a provided textbox before parsing the text and extracting the concepts through their preferred or alternative labels. Figure 12 shows the user interface of Entity Extractor Tool. Examples of generated concept vectors for two representative suppliers from the target classes are shown in Table 2.

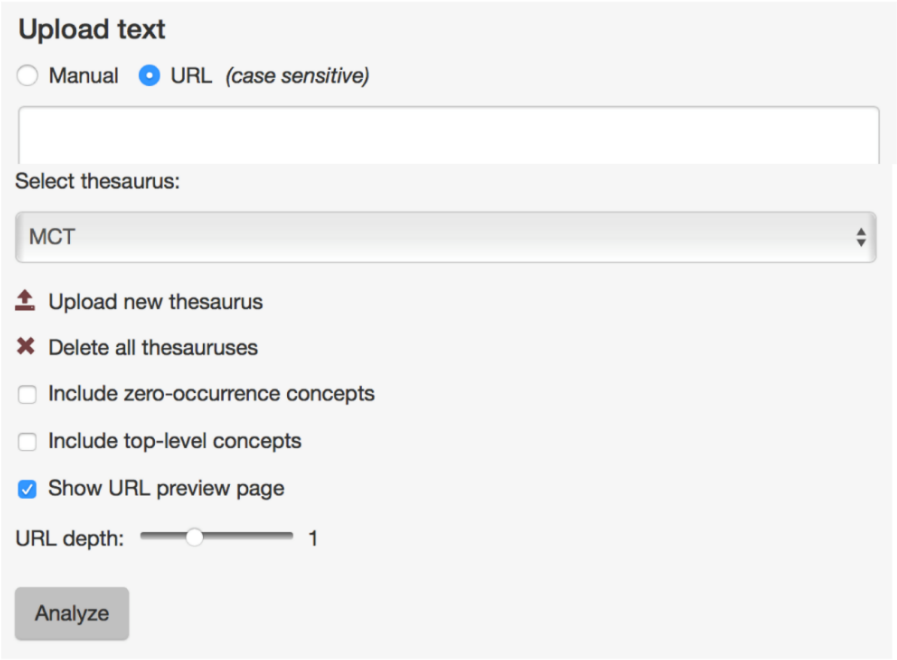

Figure 12: Entity Extractor Tool User Interface

The output of the Entity Extraction step is a vector of observed concepts (along with their frequencies) for each manufacturing supplier. All 260 concept vectors are then combined to form a document-term matrix of $n \times 260$, where n indicates the number of

extracted concepts for all suppliers participating in the experiment. The body of the matrix contains the frequencies of occurrences of the concepts for each supplier.

Table 2: An Example of Two Suppliers' Concept Vectors

Heavy Component Machining for *Supplier XYZ*	**Frequency**	**Precision and Complex Machining for *Supplier ABC***	**Frequency**
Large component	9	Complex machining	6
Heavy component	2	5-axis machining	2
Large part	3	7-axis machining	4
Large machining	6	Difficult machining	5
Large machined part	1	Live tooling	1
Heavy lifting equipment	2	Complex machined part	6
Vertical machining center	5	Complex precision part	2
Heavy CNC machining	7	Machining	7
Large part machining	3	Multi-axis complex machining	1
Vertical boring	4	Complex CNC machining	3
Heavy machining	2	Large-machined part	5

2.8. Performance Evaluation

80% of the data was selected for training the classifier, and the rest was reserved for test and validation. Four classification algorithms, namely, Random Forest, KNN, SVM, and Decision Tree, were used in this experiment. The classification process was implemented and executed in Python environment.

2.8.1. Classification Performance Evaluation Metrics

Precision, recall, and F-measure are the three metrics used to evaluate the classifier's accuracy. Precision is the ratio of the number of correctly classified suppliers and the total number of returned suppliers, while recall is the ratio of correctly classified suppliers to the total number of suppliers belonging to the class of interest in the entire dataset. F-measure is a single score balancing precision and recall, reflecting the model accuracy. To measure model accuracy for all four evaluated classifiers, precision, recall and F-measure equations were used as shown in Table 3.

Table 3: Calculation of Precision, Recall, and F-Measure for Heavy Machining Class

True Positive (TP) The number of heavy machining suppliers which are correctly classified as heavy machining class.	False Negative (FN) The number of heavy machining suppliers which are not classified as heavy machining class.	TP+FN
False Positive (FP) The number of non-heavy machining suppliers which are incorrectly classified as heavy machining class.	True Negative (TN) The number of non-heavy machining suppliers which are not classified as a heavy machining class.	FP+TN
TP + FP	FN + TN	N

$$Precision = \frac{TP}{TP + FR} \quad (1)$$

$$Recall = \frac{TP}{TP + FN} \quad (2)$$

$$F - measure = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (3)$$

2.8.2. Classification Results

20% of the experimental dataset (composed of a mix of 52 suppliers from both groups) were used to test and validate the classifiers. The results obtained for all four classifiers are shown in Table 4.

Table 4: Classification Results

Metric	Decision Tree	SVM	Random Forest	KNN
Precision	81.48%	69.70%	60%	51.16%
Recall	84.62%	88.46%	57.69%	84.62%
F-measure	83.02%	77.97%	58.82%	63.77%

2.9. Conclusion

As shown in Table 4, the *Decision Tree* classifier has a higher F-measure, which indicates higher accuracy in class prediction. After classification, the suppliers in the test group are then transferred to the next stage during which they are ranked according to their semantic similarities with the queried capabilities. To evaluate the impact of the classification step on the accuracy of the final ranking results, the ranking process is carried out separately for classified and unclassified data. The hypothesis is that if the

query vector and the document vectors belong to the same class, the ranking results will be more accurate since the query and the data both share the same context.

3. SIMILARITY MEASUREMENT

3.1. Introduction

Natural Language Processing methods usually implement some type of text similarity measurement technique to support information retrieval, recommendations, automatic question answering, machine translation, dialogue systems, and document matching. Text similarity is defined as the commonness of two piece of text fragments, the more commonness two pieces of text fragment has the more similar, they are (J. Wang & Dong, 2020).

Text similarity approaches can be divided into corpus-based and knowledge-based similarities. With the development of neural network representation, some sematic relationships and graph structures are considered in calculating the text similarity. In addition to semantic similarity, broader perspectives of sematic properties are also considered in text similarity. This work uses text similarity techniques in a graph structure to consider the distance of the words instead of text piece fragments.

According to the motivating use case for the proposed manufacturer recommendation framework, this work aims to provide a quantitative technique for ranking suppliers within certain capability classes based on their similarities to the desirable capabilities. The use case that motivates this work is a supplier search scenario where a customer submits a query to find one or more suppliers that have a set of desirable capabilities related to available manufacturing processes, resources, equipment, processible materials, industry focus, quality certifications, and acceptable part types. We assume that there exists a *supplier recommender system* that receives the query from the user and returns a list of highly relevant suppliers that are ranked according to their

similarity to the query. Since the manufacturers are also represented as concept vectors extracted from the same thesaurus, similarity measurement between the requested capabilities and provided capabilities is reduced to a set of pairwise similarity measurements between requested and provided capability concepts. For example, if the query includes *end milling* and the requested process but a manufacturer only offers *face milling* process, then the similarity between *face milling* and *end milling* needs to be calculated as a component of the overall similarity between query and advertisement with respect to manufacturing processes.

3.2. Related Work

This section provides a brief overview of the existing computational methods for measuring the similarity between terms or phrases. The existing methods can be categorized under two main approaches, namely, *knowledge-based* and *corpus-based approaches* which correspond to the semantic similarity relationship and semantic similarity relatedness, respectively (Sharma et al., 2016).

3.2.1. Knowledge-based Similarity Measurement

Knowledge-based methods calculate the semantic similarity between two terms or concepts based on the information derived from one or more underlying knowledge sources such as ontologies, thesauri, or concept schemas. Depending on how the semantic similarity between words is assessed, knowledge-based semantic similarity methods can be categorized as:

- Edge-based methods
- Feature-based methods
- Information content-based methods

Edge-Based Methods: The general edge-based method is to consider the underlying ontology as a graph connecting words hierarchically and counting the edges between two terms to measure the similarity between them. The terms that are farther apart tend to be the less similar. A measure called $path$ that was proposed by Rada et al. considers the similarity between two terms to be inversely proportional to the length of the shortest path between them (Rada et al., 1989). Edge-based method considers the fact that the words located deeper down in the hierarchy of a graph have a more specific meaning, and that, they may be more similar to each other even though they have the same distance as two words which represent a more generic concept was not taken into consideration.

$$sim_{path}(t_1, t_2) = \frac{1}{1 + \min(length(t_1, t_2))} \quad (4)$$

The proposed method by Leacock Chordorow known as lch technique employs a non-linear function to evaluate the semantic similarity between concepts based on their shortest route length, where D is the maximum depth of a Knowledge Graph's concept taxonomy (KG). Given that KGs comprise concepts that may be structured as a concept taxonomy with hierarchical relations, depth is the path through hierarchical relations between the root concept and a given concept (Leacock & Chodorow, 1998). The lch equation is shown in Equation 5.

$$sim_{lch}(t_1, t_2) = -\log \frac{length(t_1, t_2)}{2 \times D} \quad (5)$$

In this equation, $length_{min}(t_1, t_2)$ represents the minimum distance between the terms t_1 and t_2. More sophisticated edge-counting methods also take depth into account. Terms that are located deeper in the taxonomy have more specific semantics compared to

the shallower terms. Therefore, it is reasonable to include both the depth and the path length as influential variables when calculating semantic similarity in knowledge-based methods.

Wu and Palmer proposed a measure, known as wup measure, that uses the path length from the root node to the Least Common Subsumer (LCS), or Most Specific Parent (MSP), of the two terms (Zhu & Iglesias, 2016). This value is then scaled by the sum of the depth of the individual terms as shown in Equation 6.

$$sim_{wup}(t_1, t_2) = \frac{2depth\ (t_{lcs})}{depth\ (t_1) + \text{depth}(t_2)} \tag{6}$$

In this equation, depth (t_{lcs}) is the depth of LCS node and depth (t_1) and depth (t_2) indicate the depth of the first term and the second term respectively.

The shortest path length is defined as an exponential function of the similarity of two concepts according to (Y. Li et al., 2003). The similarity equation is as follows:

$$sim_{length} = e^{-\alpha.\min len(t_1,t_2)} \tag{7}$$

In addition to the shortest path length, the approach utilizes a nonlinear function. It has been discovered that the strongest correlation is found at $\alpha = 0.25$.

Li et al. proposed a non-linear measure, as shown in Equation 6, which considers both the minimum path distance and depth. In this equation, the optimal values of $\alpha = 0.2$ and $\beta = 0.6$ are derived empirically (Y. Li et al., 2003).

$$sim_{li} = e^{-\alpha.\min len(t_1,t_2)}.\frac{e^{\beta.depth\ (t_{lcs})} - e^{-\beta.depth\ (t_{lcs})}}{e^{\beta.depth\ (t_{lcs})} + e^{-\beta.depth\ (t_{lcs})}} \tag{8}$$

One shortcoming of edge-based methods is that they assume edges between terms or concepts to have equal length.

Feature-Based Methods: In these categories of knowledge-based methods, calculation is based on a function of properties of the words, like gloss, neighboring concepts, etc. Gloss is defined as the meaning of a word in a dictionary; a collection of glosses is called a glossary. Gloss-based semantic similarity measures exploit the knowledge that words with similar meanings have more common words in their gloss (Sánchez et al., 2012). In another word, in feature-based methods, similarity is a function of the properties or features of the words in a lexical or logical model. The semantic similarity is measured based on the extent of overlap between the features of the words in consideration.

The Lesk measure assigns a value of relatedness between two words based on the overlap of words in their gloss and the glosses of the concepts they are related to in an ontology like WordNet (Lastra-Díaz et al., 2019).

Jiang et al. proposed a feature-based method where semantic similarity is measured using the glosses of concepts present in Wikipedia (Y. Jiang et al., 2015). Most feature-based methods take into account common and non-common features between two words/terms. The common features contribute to the increase of the similarity value and the non-common features decrease the similarity value. The major limitation of feature-based methods is its dependency on ontologies with semantic features, and most ontologies rarely incorporate any semantic features other than taxonomic relationships.

Information Content-Based Methods The information content (IC) is the value of information contained in a word in the context. IC is calculated based on the probability of occurrence of a word in a corpus such as WordNet or Wikipedia Category Graph (Martis et al., 2013). A high value of this measure indicates that the word is more

specific and clearly describes a concept with less ambiguity, and conversely for the lower information content values, that is, the words are more abstract in meaning in lower values (Yang et al., 2020). The specificity of the word is determined using Inverse Document Frequency (IDF), which implies on the principle that the more specific a word is, the less it occurs in a document. Information content-based methods measure the similarity between terms using the information content value associated with them.

Resnik proposed a semantic similarity measure called res (Resnik, 1995). The base of this measure is on the idea that if two concepts share a common subsumer they share more information since the IC value of the LCS is higher. Considering IC represents the Information Content of the given term, res is measured as,

$$sim_{res}(t_1, t_2) = IC_{t_{lcs}} \tag{9}$$

By considering the IC value of the terms that dedicate to the individual information of description of the terms and the IC value of their LCS, D. Lin proposed a below measure.

$$sim_{lin}(t_1, t_2) = \frac{2IC_{t_{lcs}}}{IC_{t_1} + IC_{t_2}} \tag{10}$$

A distance measure based on the discrepancy between the sum of the individual IC value of terms and the IC value of their LCS was designed by Jiang and Conrath in (J. J. Jiang & Conrath, n.d.).

$$dis_{jcn}(t_1, t_2) = IC_{t_1} + IC_{t_2} - 2IC_{t_{lcs}} \tag{11}$$

However, knowledge-based systems are highly dependent on the underlying source resulting in the need to update them frequently which requires time and high computational resources. Although strong ontologies like WordNet exist for the English

language, similar resources are not available for other languages that results in the need for the building of strong and structured knowledge bases to implement knowledge-based methods in different languages and across different domains. Various research works were conducted on extending semantic similarity measures in the biomedical domain (Soğancıoğlu et al., 2017) . McInnes et al. built a domain-specific model called UMLS to measure the similarity between words in the biomedical domain. With nearly 6,500 world languages and numerous domains, this becomes a serious drawback for knowledge-based systems (McInnes et al., 2013).

3.2.2. Corpus-Based Semantic-Similarity Methods

This group of similarity measurement methods measure the semantic similarity between terms using the information retrieved from large corpora. The underlying principle which is referred to as ‘distributional hypothesis’ exploits the idea that "similar words occur together, frequently"; however, the actual meaning of the words is not taken into consideration (Gorman & Curran, 2006). While various techniques were used to construct the vector representation of the text data, several semantic distance measures based on the distributional hypothesis were proposed to estimate the similarity between the vectors. Among all corpora measures, the cosine similarity gained significance and has been widely used among NLP researchers to date (Mohammad & Hirst, 2012). Based on the underlying methods using which the word-vectors are constructed there are a wide variety of corpus-based methods some of which are discussed in this section.

Latent Semantic Analysis (LSA) :LSA is one of the most popular and widely used corpus-based techniques used for measuring semantic similarity (Landauer & Dumais, 1997). A word co-occurrence matrix is formed where the rows represent the

words and columns represent the paragraphs, and the cells are populated with word counts. This matrix is formed with a large underlying corpus, and dimensionality reduction is achieved by a mathematical technique called Singular Value Decomposition (SVD). SVD represents a given matrix as a product of three matrices, where two matrices represent the rows and columns as vectors derived from their eigenvalues and the third matrix is a diagonal matrix that has values that would reproduce the original matrix when multiplied with the other two matrices (Landauer et al., 1998). SVD reduces the number of columns while retaining the number of rows thereby preserving the similarity structure among the words. Then each word is represented as a vector using the values in its corresponding rows and semantic similarity is calculated as the cosine value between these vectors. LSA models are generalized by replacing words with texts and columns with different samples and are used to calculate the similarity between sentences, paragraphs, and documents.

Normalized Google Distance: Normalized Google Distance (NGD) is a relative semantic metrics that reflects the similarity of two words or phrases in Web documents or other large databases. NGD measurement is based on the co-occurrence of the words in webpages, the more frequent they come together the more similar they are. Given two terms (t_1, t_2), NGD uses Equation 12 to calculate their similarity (Cilibrasi & Vitanyi, 2007).

$$NGD(t_1, t_2) = \frac{\max\{\log f(t_1), \log f(t_2)\} - \log f(t_1, t_2)}{\log G - \min\{\log f(t_1), \log f(t_2)\}} \quad (12)$$

$f\ (t_1)$ and $f\ (t_2)$ represent the number of hits in Google search for terms x and y, and $f\ (t_1, t_2)$, returns the number of hits when both terms appear together in a page and G indicates the total number of pages participating in the Google search. NGD is often

used to measure semantic relatedness rather than semantic similarity because related terms occur together more frequently in web pages though they may have opposite meaning. Another shortcoming of NGD is that it is highly sensitive to the context in which the terms appear and if the domain of the search includes the pages that come from a variety of heterogeneous contexts, the NGD won't return accurate results.

In this research, a weighted edge-based method is used for calculating the semantic similarity between two terms. The weight for each edge is calculated using the NGD method. Five different equations of edge-based method including: *path, lch, wup, length,* and *Li* methods are examined in this work to identify the most suitable method for the particular use case that motivates this work. There are two defining factors in all five edge-based methods, namely, *length* and *depth*. Length is the number of edges counted between two concepts in the shortest path connecting them in the thesaurus, while Depth is the number of nodes existing between the *"root node"* and the *"least common subsumer"* of the two concepts (J. J. Jiang & Conrath, 1997)(Y. Li et al., 2003). The more the least common subsumer is to the root node, the more general meaning it conveys comparing to least common subsumer in deeper levels of the thesaurus. However, it cannot be claimed that if two sets of concepts share the least common subsumer, they definably are equal in case of similarity measurement, because the distance or the length of two concepts also matters (Dong et al., 2010). Path, lch, Wu-Palmer, length, and Li metrics are used for measuring the edge-based similarity of two concepts respectively.

As shown in Equation 1, the *path* metric uses the shortest path between two concepts to measure their similarity, The lch metric uses a non-linear function to consider

the shortest path between two concepts as shown in Equation 2. he depth of terms in the taxonomy defined by "D" which shows the depth of the concepts least common subsumer.

The Wu-Palmer metric uses depth as the only criterion for assessing the similarity measurement as shown in Equation 3.

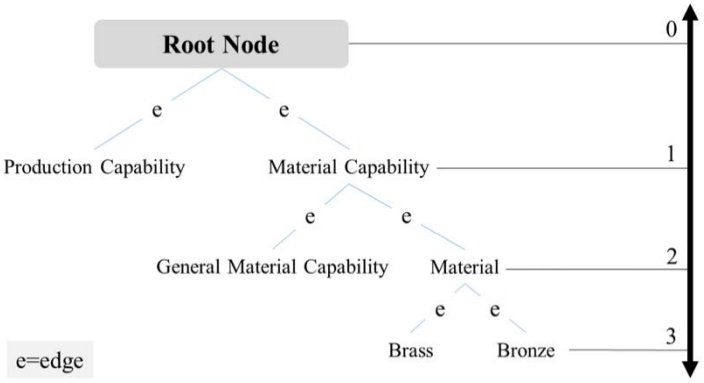

Figure 13: The depth of a concept is the shortest path between the concept and the root

Where depth (C_{lcs}) is the distance of the Least Common Subsumer (or the most shared ancestor) of concepts to the taxonomy root, which represents the specification level and depth (c_i) is equal to $length\ (c_{\mathrm{i}}, c_{\mathrm{root}})$, which defines the shortest length from ci to root. For example, to measure the similarity for pair of (Brass, Bronze) as shown in Figure 13, since Material is their LCS, depth (Brass) will be equal to 3 as the shortest path between Brass and the root (MCT) is defined with three edges, the same is true for depth (Bronze). Material is the second level of the tree, so depth (C_{lcs}) will be depth (Material)=2.

The length method, like the Path metric, investigates the shortest path between two concepts but evaluates similarity using an exponential function as shown in Equation 4. The best correlation was found to be at $\alpha = 0.25$.

Li method introduces two influencing factors of graph distance and specification level on the semantic similarity measurement. Both components of length and depth, which are indicators of graph distance and specification level, are utilized to determine semantic similarity of concepts, as shown in equation 5. To achieve the best outcome in this equation, α and β must be equal to 0.2 and 0.6, respectively, according on the results of Li's research. As can be seen from the equations, *path* and *length* methods only consider length as a defining component when measuring similarity, but *wup* only uses depth. *Li* and *lch* techniques, on the other hand, employ both parameters to determine similarity. One of the goals of this study is to see if different similarity measurement methods produce noticeably different outcomes. To harmonize the context for similarity measurement between word pairs, the webpages are classified first under predefined classes. In this way, word similarity measurement is only conducted within similar groups capability narratives related to similar manufacturers.

3.3. Similarity Measurement

According to the motivating use case for the proposed manufacturer recommendation framework, the user submits a query composed of a vector of thesaurus concepts that collectively represent the required manufacturing services and the desired capabilities of the supplier of those services. Since the manufacturers are also represented as concept vectors extracted from the same thesaurus, similarity measurement between

the requested capabilities and provided capabilities is reduced to a set of pairwise similarity measurements between requested and provided capability concepts.

This study uses a hybrid strategy to evaluate the semantic similarity between concept pairings, combining corpus-based and knowledge-based methodologies. The Normalized Google Distance (NGD) represents the semantic relatedness of concepts based on their co-occurrences in a large corpus, whereas the edge-based method captures the inheritance between the concepts to indicate the semantic similarity. The NGD score of two concepts is employed as the weight of the edges on the path between those concepts in the suggested hybrid technique. For the concept pair (c_1, c_2), the NGD similarity of the concepts is determined according to Equation 13.

$$sim_{NGD}(c_1, c_2) = Exp\big(-NGD(c_1, c_2)\big) \tag{13}$$

For each pair of concepts, the overall similarity is then measured using Equation 14.

$$sim_{pair}(c_1, c_2) = sim(c_1, c_2) \times sim_{NGD}(c_1, c_2) \tag{14}$$

The pairwise similarity for all concept pairs under each capability category (such as material, equipment, process, and industry) is calculated separately.

Figure 14 shows an example of a capability query in which the manufacturing processes demanded are 5-axis machining and horizontal boring. Suppose a supplier can perform three different manufacturing processes, such as 5-axis machining, vertical milling, and injection molding. In that case, there are six different pairwise similarity measurements to examine for all pairwise similarity combinations of required and provided processes (combination of choosing two concepts from six concepts). The

weighted edge-based technique will be calculated for each pair of manufacturing process concepts (Zandbiglari et al., 2021).

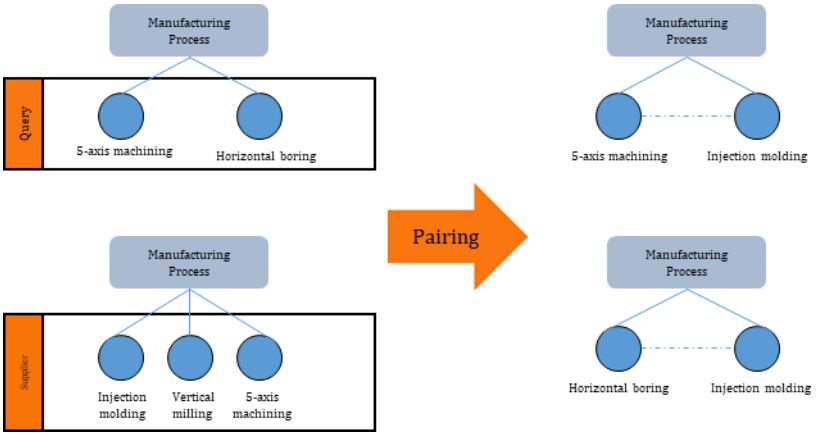

Figure 14: All Given Concepts in a Query Will Be Paired with Concepts in Thesaurus to Calculate Supplier's Similarity Score

The average pairwise similarity scores acquired for each category make up the overall similarity score for that category. As a result, the length of a supplier's concept vector will not inflate the category's total similarity score. Equation 13 shows how to calculate the overall similarity between the query and the supplier by adding the similarity values calculated for each capability category.

$$sim_{wup}(Query, Supplier) = \sum_{j=1}^{m} sim_{Category_j} \quad (15)$$

Where m is the total number of existing categories in the query.

4. RANKING EXPERIMENT

4.1. Introduction

For the ranking experiment, two queries were designed to target manufacturers in the precision and complex machining class and the heavy component machining class. The precision and complex machining query contains material capabilities, available manufacturing processes, engineering design services, supported part types, quality certifications, and target industries. In the heavy component machining query, the required production capabilities include process, equipment, item type, and industry. The desired concepts for the two queries are listed in Table 5 and Table 6. A basic Google search using the search parameters for the complex machining class yielded 755 results, but only 5 manufacturing suppliers. To increase the precision and recall of this complex search, a more advanced search technique with some semantic support is required.

Table 5: Requested Manufacturing Capabilities (Query) - Complex Machining Class

Material	Titanium, Waspaloy, Zircon, Inconel, Tool Steel
Industry	Aerospace Industry, Oil and Gas Industry, Medical Industry, Automotive Industry
Manufacturing Process	5-axis machining, Electrical discharge machining, Screw machining, Thin wall machining
Production Resource	Articulating Arm Coordinate Measuring Machine, CMM
Engineering Services	Reverse Engineering, Tool Design
Accreditation and Certification	ISO 9000, ISO 9001
Physical Artifact	Complex precision parts, complex machined parts, close tolerance parts

Table 6: Requested Manufacturing Capabilities (Query)- Heavy Component Machining Class

Industry	Aerospace Industry, Oil and Gas Industry, Mining Industry, Construction Industry, Agriculture Industry
Manufacturing Process	Heavy Duty Machining, Large CNC Machining, Large Precision Machining, Large Working Envelope Machining, Vertical Boring
Production Resources	Heavy Lifting Equipment, Gantry Machining Center, Vertical Boring Machine, Large Capacity Lathe
Physical Artifact	Large and Heavy Part, Large Diameter Part, Large and Heavy Part, Large and Heavy Machined Components

4.2. Experiment Validation

As indicated in the methodology section, each supplier's overall similarity to the queried capabilities is calculated using a combination of edge-based measures and Google Normalized Distance.

4.2.1. Normalized Google Distance (NGD)

Despite the fact that the entire Internet can be utilized as a corpus for counting hits and computing NGD, the findings will be unreliable due to context heterogeneity. In the manufacturing context, "tubing" refers to the process of forming tubes and pipes by running a strip of metal through rollers to achieve the appropriate shape. Tubing, on the other hand, can have a variety of connotations in various settings, such as a recreational sport in which a person rides on top of an inflatable tube on water or snow. A pilot manufacturing corpus was developed by crawling more than 650 manufacturing webpages with no overlap with the training and test datasets to harmonize the corpus with its context. The crawling was only limited to manufacturers' websites to impart contextual consistency to the pilot corpus. A series of codes in the Python programming environment were run to convert websites contents thoroughly to text files to create customized corpus for manufacturing capability. Related codes to the websites crawling are shown in the appendix of this work in section 4. Convert HTML to Text. Shown in the appendix, section 6. Search Engine, then searches each two concepts in the created corpus to get number of hits for each individual word in the corpus in addition to the number of times a pair of concepts appear together in the corpus. By gathering the number of hits and NGD equation, the NGD score for each pair of concepts is then

calculated using Python programming as section 5. Normalized Google Distance, in the appendix shows.

The NGD values calculated varied from zero to ∞, with zero indicating that concept pairings always appear together and higher values (including ∞) suggesting that concept pairs hardly appear together. The NGD values were scaled in a range between 0 and 1, with 1 denoting the most comparable couples, using the sim NGD(c_1, c_2) equation (exact match)

Table 7 shows the calculated NGD and Sim $_{NGD}$ values for a few example pairs of terms.

Table 7: Examples of computed NGDs

Concept 1	Concept 2	NGD	Sim_{NDG}
Bronze	Brass	0.48	0.62
Titanium	Inconel	0.08	0.92
5-axis machining	Electrical discharge machining	0.17	0.84
Horizontal Boring	Tubing	0.29	0.75
Oil and gas industry	Oil and gas industry	0.00	1.00

4.2.2. Edge-based Method

Using the given equation, $sim_{wup}(c_1, c_2)$ the similarity of all concepts given in Table 5 with all concepts existing in the manufacturing capability thesaurus is computed in Python environment, which entailed 21,780 pairwise similarity measurements.

As the thesaurus is built on the SKOS Tool provided and developed in the Infoneer Engineering lab, it was not accessible through out of the network internet. To facilitate the accessibility to the thesaurus a series of codes programmed by Python simulate the thesaurus in a HTML file containing all the broader and narrower relationships in the thesaurus as shown in the section 2. Tree Builder the appendix of this work.

As an example, according to Equation 6, $sim_{wup}(c_1, c_2)$, the similarity of (*Bronze*, *Brass*) pair, depicted in Figure 13 is calculated in Table 8.

Table 8: (brass, bronze) similarity measurement ($wup\ equation$)

$depth\ (t_{lcs})$= 2	$depth\ (Brass) = 3\ 3$	$depth\ (Bronze) = 3$
$sim_{wup}(t_1, t_2) = \frac{2depth\ (t_{lcs})}{depth\ (t_1) + \text{depth}(t_2)}$		$sim_{wup}(Bronze\ , Brass) = \frac{2 * 2}{3 + 3} = 1.5$

The same measure is applied to all possible pairwise combinations of query concepts and thesaurus concepts.

4.3. Ranking Results

Using $sim_{NGD}(c_1, c_2)$ as the weight for $sim_{wup}(c_1, c_2)$ for each pair of document and query, averaging the similarity score of each category, and summing categories' averages, the similarity scores for all 52 suppliers in the test dataset were calculated.

4.3.1. Precision and Complex Machining Query

The similarity scores of wup method for the complex query for the top-16 suppliers are shown in Table 9 and Table 10 for heavy component and complex machining classes. Table 11 ranks all top-16 suppliers of complex machining class in which is calculated through the wup equation regardless of their class.

Table 9: Similarity scores of suppliers in complex machining class- wup method- complex machining query

Suppliers' Rank in the Class	Suppliers' ID	Similarity Score
1	212	149.57
2	262	103.91
3	238	102.99
4	118	100.63
5	27	95.59
6	150	93.90
7	162	85.39
8	265	81.96
9	249	81.96
10	136	81.17

Table 10: Similarity scores of suppliers in heavy component machining class- wup method -- complex machining query

Suppliers' Rank in the Class	Suppliers' ID	Similarity Score
1	31	134.17
2	238	102.24
3	198	101.22
4	144	96.58
5	128	88.32
6	101	87.16

Table 11: Similarity score of top-16 suppliers in complex machining suppliers- wup method- complex machining query

Supplier	Suppliers' ID	Similarity Score	Classified by Decision tree classifier
1	212	149.57	Precision and Complex Machining
2	31	134.17	Heavy Component Machining
3	262	103.91	Precision and Complex Machining
4	238	102.99	Precision and Complex Machining
5	109	102.24	Heavy Component Machining
6	101	101.22	Heavy Component Machining
7	118	100.63	Precision and Complex Machining
8	144	96.58	Heavy Component Machining
9	23	95.59	Precision and Complex Machining
10	150	93.90	Precision and Complex Machining
11	128	88.32	Heavy Component Machining
12	101	87.16	Heavy Component Machining
13	162	85.39	Precision and Complex Machining
14	265	81.96	Precision and Complex Machining
15	249	81.96	Precision and Complex Machining
16	136	81.179	Precision and Complex Machining

As mentioned earlier in this work, 5 edge-based method metrics have been used to calculate the similarity of manufacturing suppliers. When listing the top-16 suppliers of each equation, it can be seen that there are a combination of 16 out of 20 mutual suppliers in different orders in all metrics. Table 12 shows similarity scores for the 20 mutual suppliers based on their scores for lch, path, Li, and length metrics.

Table 12: Precision and Complex Machining query suppliers' scores

Suppliers' ID	lch	Path	Length	Li	Class of Capability
4	1139.81766	798.175204	436.066855	17.3333333	Heavy Component
16	1013.95627	387.128677	708.200435	17.3333333	Heavy Component
23	1673.31041	1171.79459	633.324014	5.33333333	Precision and Complex
31	2089.53995	1459.75163	791.983455	22.6666667	Heavy Component
54	1012.86864	701.256156	383.707888	17.3333333	Precision and Complex

Suppliers' ID	lch	Path	Length	Li	Class of Capability
65	826.601922	577.022962	316.679872	16	Precision and Complex
73	1402.67211	975.492523	530.677777	10.6666667	Precision and Complex
101	1349.01791	934.962832	507.676789	5.33333333	Heavy Component
108	788.922063	546.656752	299.445811	10.6666667	Precision and Complex
109	1729.72965	1729.72965	656.541939	10.6666667	Heavy Component
118	1645.56535	1147.53909	622.149408	10.6666667	Precision and Complex
125	1123.93751	778.861416	419.521721	0	Heavy Component
128	1396.22966	974.627759	529.904722	10.6666667	Heavy Component
136	1109.98101	769.574996	417.312155	5.33333333	Heavy Component
140	1151.6368	803.183096	419.521721	5.33333333	Heavy Component
144	1429.95099	995.023685	543.018781	41.3333333	Heavy Component
150	1421.86926	991.255224	537.59996	0	Precision and Complex
162	1347.64229	935.265072	505.678672	0	Precision and Complex
198	1744.47588	1210.48348	652.867332	0	Heavy Component
212	2259.31168	1578.41623	858.798566	41.3333333	Precision and Complex
238	1575.59018	1093.76363	596.047027	37.3333333	Precision and Complex
249	1108.36254	768.92537	421.592778	36	Precision and Complex
262	1648.013	1150.77804	626.666589	25.3333333	Precision and Complex
26	1108.36254	768.92537	421.592778	22.666666	Precision and Complex

4.3.2. Heavy Component Machining Query

Considering the requested query of heavy component machining the similarity scores of wup, lch, path, Li, and length metrics for the 20 suppliers are shown. When listing the top-16 suppliers of each equation, it can be seen that there are a combination of 16 out of 20 mutual suppliers in different orders in all metrics as shown in Table 13.

Table 13: Heavy Component Machining query suppliers' scores

Suppliers' ID	wup	lch	Path	Length	Li	Capability Class
4	199.6996	42.88937	2745.333	1935.113	17.33333	Heavy Component
23	291.6134	60.16066	4055.488	2865.586	22.66666	Heavy Component
31	319.7800	49.9569	4898.9860	3448.5845	5.3333	Heavy Component
73	241.3889	49.3891	3341.5315	2344.8755	17.3333	Precision and Complex
101	42.3503	42.3503	3197.4987	2235.8260	10.6666	Heavy Component
109	4102.8070	2890.7725	4102.8070	2890.7725	0	Precision and Complex
118	239.5820	39.32273	3834.9113	2698.1399	5.3333	Precision and Complex
125	173.7256	29.8628	2403.7273	1900.0548	5.3333	Precision and Complex
128	238.2205	42.9778	3351.5741	2355.0235	10.6666	Precision and Complex
136	188.1868	31.27930	2649.8523	1855.0832	5.3333	Heavy Component
140	219.4237	52.3864	2861.3838	2022.0372	25.333333	Heavy Component
144	282.40605	75.6279	3510.0701	2467.9091	41.3333	Precision and Complex
150	3460.8376	2442.1190	2442.1190	2442.1190	36	Heavy Component
162	243.63816	43.41636	3279.3036	2298.7800	10.6666	Heavy Component
177	198.91486	43.439893	2575.1616	1803.449	17.3333	Precision and Complex
187	188.39467	58.297013	1962.5736	1383.7712	37.3333	Heavy Component
198	337.24080	83.27851	4294.4393	3019.4411	41.3333	Heavy Component
212	344.87651	58.547379	5332.9151	3752.6579	10.666666	Precision and Complex
238	270.76537	58.44421	3724.3552	2609.9528	22.6666	Precision and Complex
249	185.63896	36.093753	2621.3687	1831.5625	10.6666	Precision and Complex
262	278.0614	53.2941	3925.0979	2760.3920	16	Precision and Complex

4.3.3. Spearman’s Correlation

The degree of correlation independent variables is determined using Spearman rank correlation coefficients. It is comparable to Pearson's product-moment correlation coefficient; only it works with data ranks instead of raw data. Spearman's rank correlation

coefficient has advantages over the typical product-moment correlation coefficient (*Detecting Trends Using Spearman′s Rank Correlation Coefficient | Elsevier Enhanced Reader*, n.d.). In this section, Spearman's Rank-Order Correlation is used to compute the correlation between the human expert generated rankings with computationally generated rankings using Equation 15.

$$\rho = 1 - \frac{\sum 6\, d_i^2}{n\,(n^2 - 1)} \tag{15}$$

In Equation 16, ρ shows Spearman's rank correlation coefficient, d_i is the difference between two ranks of each observation and n is the total number of observations. Table 14 shows human expert rankings of suppliers for both target classes.

Table 14. Human expert ranking

Precision and Complex Machining		Heavy Component Machining	
Suppliers' ID	Human Expert Ranking	Suppliers' ID	Human Expert Ranking
4	9	4	1
16	16	23	3
23	3	31	13
31	5	73	14
54	10	101	19
65	6	109	6
73	8	118	21
101	17	125	7
108	19	128	9
109	20	136	15
118	23	140	4
125	12	144	11
128	13	150	12
136	24	162	2
140	4	177	16
144	2	187	17
150	18	198	18
162	11	212	8
198	22	238	10
212	1	249	20
238	21	262	5

Precision and Complex Machining		Heavy Component Machining	
Suppliers' ID	Human Expert Ranking	Suppliers' ID	Human Expert Ranking
249	15		
262	14		
265	7		

Correlation analysis results of precision and complex machining and heavy component machining class for all examined similarity metrics are shown in Table 15 and Table 16.

Table 15. Spearman's rank-order correlation result for complex machining query

METRIC	**Iteration 1** (all suppliers)	**Iteration 2** (only complex machining)	**Iteration 3** (only heavy machining)
Path	0.788235	0.925000	0.678571
lch	0.976470	0.619047	0.619047
wup	0.726471	0.941667	0.357143
Length	0.788235	0.964285	0.366666
Li	0.947058	0.988095	1

Table 16. Spearman's rank-order correlation result for heavy component machining query

METRIC	**Iteration 1** (all suppliers)	**Iteration 2** (only complex machining)	**Iteration 3** (only heavy machining)
Path	0.523529	0.683333	0.5
lch	0.998529	0.751470	0.982142
wup	0.788235	0.607142	0.35
Length	0.822058	0.261904	0.988095
Li	0.751470	0.303571	0.925

According to the equations, each metric considers different variables to measure similarity of the formulated capability query with the manufacturing supplier. The length and the depth are the main variables of these equations. As previously indicated in chapter 4, some of these metrics consider either the length or depth variable while some consider both. The average Spearman's rank-order correlation result of the metrics is based on the main variables involved in the measurement determined in Table 17 and Table 18 respectively for complex machining and heavy component machining.

Table 17. Spearman's correlation average for complex machining query

Complex Machining Query			
Considered factor in the equation	**Metric**	**Spearman's correlation**	**Average Spearman's correlation**
Length	Path	0.788235	0.788235
	Length	0.788235	
Depth	wup	0.726471	0.726471
Length and Depth	lch	0.97647	0.8823525
	Li	0.788235	

Table 18. Spearman's correlation average for heavy component machining query

Heavy Component Machining Query			
Considered factor in the equation	**Metric**	**Spearman's correlation**	**Average Spearman's correlation**
Length	Path	0.523529	0.6727935
	Length	0.822058	
Depth	wup	0.788235	0.788235
Length and Depth	lch	0.998529	0.8749995
	Li	0.75147	

It can be seen that the average Spearman's correlation is higher for the lch and Li metrics which consider both length and depth.

4.4. Conclusion

Table 12 shows that among the top-five suppliers, two are classified as heavy machining suppliers. Because the complex machining and heavy machining capability groups overlap significantly, this was an expected result. Manufacturers who can supply machining services in one class are quite likely to be able to give the services required in the second. Those highly ranked suppliers in the heavy component machining class would be classified under both classes if a multi-label classification approach was applied. When the capability classes are significantly diverse, there is a sharper division between the groups. Rank-order correlation result indicates that there is a significant correlation between human and machine generated rankings when suppliers are first classified under pre-defined capability classes. The rank order correlation result supports the original hypothesis that NGD measure yields better results when the concepts pairs come from the same semantic context. This result justifies the need for conducting classification prior to similarity measurement.

Table 15 and Table 16 represent low correlation between human and machine ranking for the suppliers in the opposite class of the formulated class. One explanation for this low correlation is that an individual is able to analyze the capabilities more accurately though browsing different pages of the website of a supplier, and for instance, identifying the primary and secondary expertise of a supplier, thus giving a higher rank to those companies that use complex machining as their primary expertise in the experiment when the formulated query belongs to complex machining. However, due to using a light-weight ontology, the thesaurus, the expressivity needed for representing primary and secondary capabilities is not supported in the proposed approach. Overall, the proposed method provides a reasonable tradeoff between the search accuracy and search time.

5. CONCLUSION AND FUTURE WORK

5.1. Introduction

This study proposes a mechanism for categorizing and rating manufacturing suppliers based on their capability narratives. A formal manufacturing capability thesaurus supports the proposed procedure. A combination of edge-based and corpus-based methodologies is used in the suggested similarity measurement method. When the search area is populated by a vast number of manufacturers, it is necessary to narrow down the search space to a smaller group of highly relevant suppliers. This objective is achieved through sequential classification and ranking of suppliers based on the similarities between their capability narratives with the queried capabilities. The findings of this research can support supplier screening process in the early stages of suppler discovery.

One of the goals of this study is to assess the efficacy of evaluating manufacturing capabilities using lightweight and low-fidelity knowledge models (such as a thesaurus) and public manufacturing datasets (unstructured data from manufacturers' websites). Although heavy-weight and axiomatic ontologies and gold standard corpora can improve search precision, their development time and cost can be prohibitive. As a result, the proposed method builds a reasonable balance between the overall performance and the development cost. Because the Manufacturing Capability Thesaurus uses SKOS as a standard representation language, it can be shared and extended in a decentralized manner by various groups. This will significantly reduce the effort required to develop a comprehensive formal vocabulary of manufacturing capabilities for various manufacturing industry sectors.

5.2. Findings

This study utilized different methods and algorithms to address the research questions identified in Chapter 1. This section provides the responses to the initial research questions that motivated this work.

5.2.1. Which classifier will have the best precision/recall/ F-measure?

In this work, four classifiers, namely, Decision Tree, Random Forest, K-Nearest Neighbor and Support Vector Machine were used to classify manufacturing suppliers into two capability classes of *precision and complex machining* and *heavy component machining*. As shown in Table 4, decision tree classifiers with the highest precision, recall, and F-measure provided the most precise classification result in this work.

Decision Tree classifiers are widely used in data mining to create classification algorithms based on various covariates or to forecast target variables. The method generates an inverted tree using a population of individuals classified into branch-like segments containing the root, internal nodes, and leaf nodes (SONG & LU, 2015).

5.2.2. How to compare automatically generated rankings with human expert's rankings?

Spearman's Rank-Order Correlation is used to compute the correlation between the human expert generated rankings with computationally generated rankings. Spearman's rank-order correlation compared the results attributed by the machine and the human expert and provides a correlation between 0 to 1 with 1 indicating the highest correlation.

5.2.3. How can classification affect similarity scores?

Knowing that decision tree provides the best highest precision, recall, and F-measure, ranking experiment uses the results obtained from decision tree classification to study the effect of classification on the similarity measurement ranking. The comparison of Spearman's rank-order classification for three iterations of non-classified suppliers, suppliers belonging to the same capability class that the query belongs to, and the opposite classes of query and suppliers showed significantly higher correlation in the second iteration which confirms the effect of classification on ranking results.

Classification as the first step of this hybrid approach, screens suppliers' websites content and categories them based on the content they have provided. This initial screening narrows down the list of qualified suppliers which are expected to have more similarity with the formulated query of capability. Therefore, it was expected to see significantly higher correlation in ranking when suppliers have been classified at the first step.

5.2.4. What method of similarity measurement should be used for the ranking?

This research used a weighted edge-based method to consider both inheritance relationships and relatedness of concepts in the thesaurus by having both corpus-based methods and knowledge-based methods included. Five metrics of edge-based methods for two different queries of capabilities were studied and the average Spearman's correlation showed higher correlation higher for the *lch* and *Li* metrics which consider both length and depth as the elements of the similarity measurement, while other metrics considered either length or depth.

5.3. Future Work

In the validation stage, only a single human expert was used to evaluate the outcome. In the future, multiple experts will participate in the validation phase to improve the credibility of the findings.

As discussed in the limitation section, there are different and limited number of suppliers in each class, collecting adequate data for training and test set difficult. The lack of having adequate and appropriate data is a barrier to using some more precise approaches such as deep learning. In future, this limitation can be resolved by changing the source of data, gathering data not only from US based company in the field, choosing different capability classes which provide a more extensive list of suppliers.

The corpus in this work was established by crawling 650 unique manufacturing suppliers' websites. Although increasing the number of websites will result in increasing the run time of the codes, it improves the accuracy and precision of the results. Creating a sufficiently comprehensive and thorough corpus may provide the possibility of using Information Content-based (IC) methods in future works.

In addition to focusing on manufacturing and processes related capabilities, extending, and developing the thesaurus concepts can provide future research with the opportunity to explore other aspects of manufacturing suppliers' capabilities, such as logistic services.

Additionally, future work can be focused on improving the data acquisition into more autonomous improving the entity extracotr feature on the SKOS Tool.

Traditional deep learning-based models based on convolutional neural networks (CNN) and recurrent neural networks (RNN) mostly use dynamic character- level of word level embedding as the input which makes text feature extraction not to be comprehensive.

BERT (Bidirectional Encoder Representations from Transformers) method represented by Google in October 2018 is an automatic text classification method and an open machine framework for natural language processing. While other methods can only read characters from either left to right or right to left, BERT considers two sides of each character at a time to implement sematic information (W. Li et al., 2019). BERT provides a significant improvement as it does not require data in any specific sequence which enables it to process and pre-train larger amount of data. BERT method can be used in future works to pre-train massive amount of data.

APPENDIX SECTION

Payton Codes of Classification and Ranking

1. Get Relevant Data

```
import argparse
import math

import numpy as np
import pandas as pd
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import tree
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score
# from sklearn.metrics import precision_recall_fscore_support
from sklearn import decomposition
from sklearn.model_selection import GridSearchCV
from sklearn import svm

import tree_builder
import nd


classes = {'complex': 0, 'heavy': 1}


def _generate_upper_case(text):
    str_list = text.split()

    # Define a variable to store the converted string
    new_string = ''

    # Iterate the list
    for val in str_list:
        # Capitalize each list item and merge
        new_string += val.capitalize() + ' '

    return new_string:-1]


def _find_LCS(first_node, second_node):
    first_node_path = ]
    first_node_parent = first_node
    while first_node_parent is not None:
        first_node_path.append(first_node_parent)
        first_node_parent = first_node_parent.parent

    second_node_parent = second_node
    path_length = len(first_node_path)-1
    while second_node_parent is not None:
```

```
        if second_node_parent in first_node_path:
            return second_node_parent, path_length

        second_node_parent = second_node_parent.parent
        path_length += 1

def decision_tree_regressor_predict_proba(X_train, y_train, X_test,
**kwargs):
    """Trains DecisionTreeRegressor model and predicts probabilities of
each y.

    Args:
        X_train: Training features.
        y_train: Training labels.
        X_test: New data to predict on.
        **kwargs: Other arguments passed to DecisionTreeRegressor.

    Returns:
        DataFrame with columns for record_id (row of X_test), y
        (predicted value), and prob (of that y value).
        The sum of prob equals 1 for each record id.
    """
    # Train model.
    m = DecisionTreeRegressor(**kwargs).fit(X_train, y_train)
    # Get y values corresponding to each node.
    node_ys = pd.DataFrame({'node_id': m.apply(X_train), 'y': y_train})
    # Calculate probability as 1 / number of y values per node.
    node_ys'prob'] = 1 /
node_ys.groupby(node_ys.node_id).transform('count')
    # Aggregate per node-y, in case of multiple training records with
the same y.
    node_ys_dedup = node_ys.groupby('node_id',
'y']).prob.sum().to_frame()\
        .reset_index()
    # Extract predicted leaf node for each new observation.
    leaf = pd.DataFrame(m.decision_path(X_test).toarray()).apply(
        lambda x:x.to_numpy().nonzero()0].max(), axis=1).to_frame(
            name='node_id')
    leaf'record_id'] = leaf.index
    # Merge with y values and drop node_id.
    return leaf.merge(node_ys_dedup, on='node_id').drop(
        'node_id', axis=1).sort_values('record_id', 'y'])


def classify(dataset):
    dataset'Hc'] = dataset'Hc'].replace('complex', 'heavy'], 1, 2])
    Y_dataset = dataset"Hc"]]
    X_dataset = dataset.drop('Hc', 1)

    le = preprocessing.LabelEncoder()
    X_dataset = X_dataset.apply(le.fit_transform)
    X_dataset = (X_dataset-X_dataset.mean())/X_dataset.std()

    X_train, X_test, Y_train, Y_test = train_test_split(X_dataset,
Y_dataset, test_size=0.2, random_state=42, stratify=Y_dataset)

    # res = decision_tree_regressor_predict_proba(X_train, Y_train,
```

```
X_test, random_state=0, min_samples_leaf=5)
    # print(res)

    # param_grid = {
    #     'bootstrap': False, True],
    #     'max_depth': 80, 90, 100, 110],
    #     'min_samples_leaf': 1, 3, 4, 5],
    #     'min_samples_split': 8, 10, 12],
    #     'n_estimators': 100, 200, 300, 1000]
    # }
    # classifier = tree.DecisionTreeClassifier()
    # classifier = RandomForestClassifier()
    # classifier = AdaBoostClassifier(n_estimators=2, random_state=42)
    # classifier = KNeighborsClassifier()
    param_grid = {'kernel': 'rbf'], 'gamma': 1e-3, 1e-4, 1e-5],
                      'C': 1, 10, 100, 1000, 5000]},
                     {'kernel': 'linear'], 'gamma': 1e-3, 1e-4], 'C': 1,
10, 100, 1000, 5000]}]
    # classifier = svm.SVC()
    # scores = 'precision', 'recall']
    classifier = GridSearchCV(
        svm.SVC(probability=True), param_grid
    )

    # grid_search = GridSearchCV(estimator=classifier,
param_grid=param_grid, cv=3, n_jobs=-1, verbose=2)

    classifier.fit(X_train, Y_train.values.ravel())
    Y_pred = classifier.predict(X_test)
    y_proba = classifier.predict_proba(X_test)

    # print(Y_test.values.ravel() - Y_pred)
    # print(Y_test)
    # print(classifier.best_params_)
    # print(grid_search.cv_results_)
    # print(Y_pred)
    # prob = classifier.predict_proba(X_test)
    test_acc = accuracy_score(Y_test, Y_pred)
    test_precision = precision_score(Y_test, Y_pred)
    test_recall = recall_score(Y_test, Y_pred)
    test_f1 = f1_score(Y_test, Y_pred)

    # packed = pd.DataFrame(X_test, Y_pred)
    # print(len(X_train))
    # print(sum(Y_pred))
    # print(len(Y_pred))
    # print(sum(Y_test.values))
    # print('test_accuracy =  {:.2f}%, precision =  {:.2f}%, recall =
{:.2f}%, f1 = {:.2f}%'.format(
    #     test_acc*100,
    #     test_precision*100,
    #     test_recall*100,
    #     test_f1*100,
    #     )
    # )
    # print(y_proba)
    # print(Y_pred)
```

```
    # print(Y_test)
    classification_result = {}
    for row, i in zip(X_test.index, Y_pred):
        classification_resultrow] = y_probai]

    return classification_result


def _compute_similarity_between_word_and_all_nodes(desired_node):
    similarity = {}
    for other_node in tree_builder.tree.all_nodes:
        LCS, path_length = _find_LCS(desired_node, other_node)
        nd_value = nd.get_normalized_distance(desired_node.data,
other_node.data)
        if other_node.data == desired_node.data:
            similarityother_node.data] = tree_builder.tree.max_depth

        else:
             similarityother_node.data] = \
                 math.exp(-nd_value)*(2*LCS.depth/float(path_length))
            #1 LCH
            # print(path_length)
            # similarityother_node.data] = \
            #     math.exp(-nd_value) * (math.log((2 * LCS.depth), 10)
/ float(path_length))
            # print(similarityother_node.data])
            # #2 PATH
            # similarityother_node.data] = \
            #     math.exp(-nd_value) * (1 / 1 + float(path_length))
            # #3 LI1
            # similarityother_node.data] = \
            #     math.exp(-nd_value) * (math.exp(-0.2) *
float(path_length))
            # #4 LI2
            # similarityother_node.data] = \
            #     math.exp(-nd_value) * (math.exp(-0.2) *
float(path_length)) \
            # * ((math.exp(0.6) * LCS.depth) - (math.exp(0.6) *
LCS.depth)) \
            # / ((math.exp(0.6) * LCS.depth) + (math.exp(-0.6) *
LCS.depth))



    return similarity


def main():
    parser = argparse.ArgumentParser()
    parser.add_argument(
        '--csv_file',
        help='Path to csv file',
        required=True,
        type=str,
    )
    parser.add_argument(
        '--query',
```

```
        help='Path to the query file',
        required=True,
    )
    parser.add_argument(
        '--dfdict',
        help='Path to Document Frequency Dict',
        required=True,
    )
    parser.add_argument(
        '--weights',
        help='Path to Weights',
        required=False,
    )
    arguments = parser.parse_args()

    tree_builder.generate_tree_from_csv_file(arguments.csv_file)
    nd.load_data(arguments.dfdict)

    weights = {}
    if arguments.weights:
        with open(arguments.weights) as f:
            lines = f.read().splitlines()

        for line in lines:
            key = line:-2]
            key = _generate_upper_case(key)
            weight = line-1]
            weightskey] = int(weight)

    with open(arguments.query) as f:
        lines = f.read().splitlines()

    supplier = pd.read_csv('./word_occurrence2/HC.csv')
    columns = supplier.columns
    for column in columns:
        upper_case = _generate_upper_case(column)
        supplier.rename(columns={column: upper_case}, inplace=True)
        if upper_case in weights.keys():
            supplierupper_case] =
weightsupper_case]*supplierupper_case]

    classification_result = classify(supplier)

    similarity = {}
    categories = {}
    for index, line in enumerate(lines):
        tokens = line.split(",")
        parent_data = _generate_upper_case(tokens0])
        categoriesparent_data] = parent_data]
        wanted_nodes = tokens1:]
        for wanted_node in wanted_nodes:
            wanted_node = _generate_upper_case(wanted_node)
            categoriesparent_data].append(wanted_node)
            related_node = tree_builder.find_node(parent_data,
wanted_node)
            if related_node is None:
                continue
```

```
            similaritywanted_node] =
_compute_similarity_between_word_and_all_nodes(related_node)

    supplier_score = {}
    # max_reward = 2
    for row in range(1, len(supplier)+1):
        if row not in classification_result.keys():
            continue

        supplier_scorestr(row+2)] = 0
        categories_score = {}
        for wanted_node in similarity:
            if wanted_node not in supplier.columns:
                continue

            value = supplierwanted_node].valuesrow-1]
            wanted_node_score = 0

            # if value >= 1:
            #     wanted node score += max reward

            # machining = '5-axis Machining', 'Electrical Discharge
Machining', 'Screw Machining', 'Thin Wall Machining']

            # number_of_similar_node = 0
            # alternative_nodes_score = 0
            for other_node in similaritywanted_node]:
                # print(wanted_node, other_node)
                if other_node in supplier.columns:
                    if supplierother_node]row-1] > 0:
                        wanted_node_score +=
similaritywanted_node]other_node]
                        # number_of_similar_node += 1

            # print('@@@@@@@@@@@@')
            # if number_of_similar_node > 0:
            #     wanted_node_score +=
alternative_nodes_score/number_of_similar_node

            #weightsDict = {} #final dict
            #names = ] #category names
            #wghts = ] #categoru weights
            #for i in range(len(names)):
            #    weightsDictnamesi]] = wghtsi]


            for key in categories.keys():
                if wanted_node in categorieskey]:
                    if key in categories_score:
                        categories_scorekey].append(wanted_node_score)

                    else:
                        categories_scorekey] = wanted_node_score]


        for key in categories_score.keys():
```

```
            categories_scorekey] =
weightskey]*(sum(categories_scorekey])/len(categories_scorekey]))

        print(categories_score)
        # print('=========================')

        # if 'Machining' not in categories_score.keys() \
        #         or categories_score'Machining'] == 0:
        #     supplier_scorestr(row+2)] = 0
        #
        # else:
        supplier_scorestr(row+2)] = sum(categories_score.values())

    d_sorted_by_value = sorted(supplier_score.items(), key=lambda x:
x1], reverse=True)
    final_complex = ]
    final_heavy = ]

    print('====================================')
    for i in range(len(d_sorted_by_value)-1):
        print(d sorted by valuei]0],
classification_resultint(d_sorted_by_valuei]0])-2])
        cls =
np.argmax(classification_resultint(d_sorted_by_valuei]0])-2])
        if cls == classes'complex']:
            final_complex.append(d_sorted_by_valuei])

        else:
            final_heavy.append(d_sorted_by_valuei])

    print('complex ' + str(final_complex))
    print('====================================')
    print('heavy ' + str(final_heavy))


if __name__ == "__main__":
    main()
```

2. Tree Builder

```
import argparse
import json
import time

import pandas as pd

import search_engine
import taxonomy
import tf_idf


global arguments, tree


def parse_arguments():
    parser = argparse.ArgumentParser()
    parser.add_argument(
        '--csv_file',
        help='Path to csv file',
        required=True,
        type=str,
    )
    parser.add_argument(
        '--website_folder',
        help='Path to your websites to generate word counts',
        required=False,
        type=str,
    )
    parser.add_argument(
        '--with_tf_idf',
        help='Generate tf idf vector',
        required=True,
        type=bool,
    )
    global arguments
    arguments = parser.parse_args()


def _generate_upper_case(text):
    str_list = text.split()

    # Define a variable to store the converted string
    new_string = ''

    # Iterate the list
    for val in str_list:
        # Capitalize each list item and merge
        new_string += val.capitalize() + ' '

    return new_string[:-1]


def _find_node(node, parent_data, data):
    for child in node.children:
        if child.data == data:
```

```
            return child

    for child in node.children:
        result = _find_node(child, parent_data, data)
        if result is None:
            continue

        return result


def find_node(parent_data, data):
    global tree
    root = tree.root

    return _find_node(root, parent_data, data)


def generate_tree_from_csv_file(csv_path):
    csv_file = pd.read_csv(csv_path)
    columns = csv_file.columns

    global tree, parent
    tree = taxonomy.Tree()
    depth = 2
    old_parents = ]
    for column_name in columns:
        if 'concept' not in column_name:
            continue

        column = csv_filecolumn_name].notnull()
        # if column is True:



        not_null_indices = columncolumn].index
        new_parents = ]
        for index in range(len(not_null_indices)):
            row_number = not_null_indicesindex]
            concept = csv_filecolumn_name].valuesrow_number]
            if concept-1] == ' ':
                concept = concept:-1]

            concept = _generate_upper_case(concept)
            node = taxonomy.Node(concept, row_number, depth)
            if depth == 2:
                parent = tree.root

            else:
                for p_index in range(len(old_parents) - 1):
                    if old_parentsp_index].row_number < node.row_number
< old_parentsp_index + 1].row_number:
                        parent = old_parentsp_index]
                        break

                    if old_parents-1].row_number < node.row_number:
                        parent = old_parents-1]
                        break
```

```
            parent.add_child(node)
            node.set_parent(parent)
            new_parents.append(node)
            tree.all_nodes.append(node)

        old_parents = new_parents
        depth += 1

    tree.max_depth = depth
    tree.plot_tree()
# python3 get_relevant_document.py --query ./words.txt --dfdict
./word_occurrence2/word_occurrence_dict.json --csv_file
./word_occurrence2/HC.csv
# python3 tree_builder.py --csv_file ./word_occurrence2/HC.csv --
with_tf_id ./word_occurrence2/word_occurrence_dict.json

def main():
    parse_arguments()
    global arguments
    generate tree from csv file(arguments.csv file)

    if arguments.website_folder:
        word_occurrence_dict = build_word_occurrence_vector()
        save_word_occurrence_vector(word_occurrence_dict)

    if arguments.with_tf_idf and arguments.with_tf_idf:
        tf_idf_dict = generate_tf_idf(word_occurrence_dict)
        save_tf_idf_dict(tf_idf_dict)


def save_tf_idf_dict(tf_idf_dict):
    json_value = json.dumps(tf_idf_dict)
    tf_idf_file = open("tf_idf_dict.json", "w")
    tf_idf_file.write(json_value)
    tf_idf_file.close()


def generate_tf_idf(word_occurrence_dict):
    tf_idf_dict = {}
    for word in word_occurrence_dict.keys():
        word_tf_idf_dict =
tf_idf.get_word_tf_idf(word_occurrence_dictword])
        tf_idf_dictword] = word_tf_idf_dict

    return tf_idf_dict


def init_search_engine(website_folder):
    print('Loading the search engine ...')
    search_engine.ROOT_PATH = website_folder
    search_engine.load_data()
    print('Search engine loaded!')


def compute_word_occurrence(node, word_occurrence_dict):
    for child in node.children:
```

```
        start = time.time()
        word_occurrence_dict = compute_word_occurrence(child,
word_occurrence_dict)
        pages_hit, page_hit_info =
search_engine.get_number_of_hits(child.data])
        word_occurrence_dictchild.data] = page_hit_info
        search_time = time.time() - start
        print('Computing occurrence vector for the word {} took {}s'
              .format(child.data, search_time))

    return word_occurrence_dict


def build_word_occurrence_vector():
    global tree
    # init the search engine
    word_occurrence_dict = {}
    init_search_engine(arguments.website_folder)
    word_occurrence_dict = compute_word_occurrence(tree.root,
word_occurrence_dict)
    return word occurrence dict


def save_word_occurrence_vector(word_occurrence_dict):
    json_value = json.dumps(word_occurrence_dict)
    f = open("word_occurrence_dict.json", "w")
    f.write(json_value)
    f.close()


if __name__ == "__main__":
    main()
```

3. Classification

```
# Importing all required libraries
from sklearn import datasets
import pandas as pd
import numpy as np
import requests
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Perceptron
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from sklearn.neural_network import MLPClassifier


def plot_contours(ax, clf, xx, yy, **params):
    Z = clf.predict(np.c_xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    out = ax.contourf(xx, yy, Z, **params)
    return out


# Downloading the dataset, creating dataframe
download_url = "https://archive.ics.uci.edu/ml/machine-learning-
databases/car/car.data"
cardata_path = "car.data"

response = requests.get(download_url)
response.raise_for_status()

# Check that the request was successful
with open(cardata path, "wb") as f:
    f.write(response.content)
print("Download ready.")

# Read the data from the file
cardata = pd.read csv("car.data", names="buying", "safety", "output"])#
Check the number of columns and rows in the file

# convert categorical data to numeric
le = preprocessing.LabelEncoder()
cardata = cardata.apply(le.fit transform)

# Get the output
Ycardatnorm_train=cardata"output"]]

# Normalize the data
normalized_cardata=(cardata-cardata.mean())/cardata.std()

# Get feature vectors X
Xcardatnorm_train=normalized_cardata"buying", "safety"]]

# Neural Network hyperparameteres
splits = 0.1, 0.2, 0.3]
learning_rates = 1, 0.1, 0.01]
n_iterations = 10, 100, 200, 400]
```

```
# splits = 0.1]
# learning_rates = 1]
# n_iterations = 10]
#open the file to report
f= open("Accuracy.txt","w+")


import matplotlib.pyplot as plt

max_test_acc = 0
best_model = None
x_boundary = None
y_boundary = None

for split in splits:
    X_train, X_test, Y_train, Y_test =
train_test_split(Xcardatnorm_train, Ycardatnorm_train, test_size=split,
random_state=1, stratify=Ycardatnorm_train)

    for learning_rate in learning_rates:
        for n_iter in n_iterations:
            ppn = MLPClassifier(activation='logistic',
learning_rate='constant',learning_rate_init=learning_rate,
max_iter=n_iter)
            #ppn = Perceptron(eta0=learning_rate, random_state=1,
max_iter = n_iter)

            #This is training the model
            ppn.fit(X_train, Y_train.values.ravel())
            #Testing the model data
            Y_pred = ppn.predict(X_test)

            train_acc = ppn.score(X_train, Y_train.values.ravel())
            test_acc = accuracy_score(Y_test, Y_pred)
            print('Train Accuracy: %.2f' % train_acc)
            print('Test Accuracy: %.2f' % test_acc)
            print('split = ', split, '  learning_rate = ',
learning_rate, '  n_iter = ', n_iter)

print('#####################################################')
            f.write('split = '+ str(split) + '  learning_rate = ' +
str(learning_rate) + '  n_iter = ' + str(n_iter) + "\n")
            f.write('Train Accuracy: ' + str(train_acc) + "\n")
            f.write('Test Accuracy: '+ str(test_acc)+"\n")
            f.write("\n")

            if test_acc > max_test_acc:
                max_test_acc = test_acc
                best_model = ppn
                x_boundary, y_boundary = X_train, Y_train


fig, ax = plt.subplots()
# Set-up grid for plotting.
X0, X1 = X_train'buying'], X_train'safety']
```

```
x_min, x_max = X0.min(), X0.max()
y_min, y_max = X1.min(), X1.max()
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min,
y_max, 0.01))

plot_contours(ax, best_model, xx, yy, cmap=plt.cm.coolwarm, alpha=0.01)
ax.scatter(X0, X1, c=Y_train.values.ravel(), cmap=plt.cm.coolwarm,
s=20, edgecolors="k")
ax.set_ylabel("{}".format('buying'))
ax.set_xlabel("{}".format('safety'))
ax.set_xticks(())
ax.set_yticks(())
ax.set_title('Decision Boundary')
plt.show()
```

4. Convert HTML to Text

```
import os

import html2text
from bs4 import BeautifulSoup as soup

ROOT_READ_PATH = "./websites"
ROOT_SAVE_PATH = "./websites_converted_to_text"


def main():
    h = html2text.HTML2Text()
    h.ignore_links = True

    if not os.path.exists(ROOT_SAVE_PATH):
        os.mkdir(ROOT_SAVE_PATH)

    websites_paths = os.path.join(ROOT_READ_PATH, website)
                      for website in os.listdir(ROOT_READ_PATH)
                      if os.path.isdir(os.path.join(ROOT_READ_PATH,
website))]

    for website_path in websites_paths:
        website = os.path.basename(website_path)
        print('Converting {} website to text files'.format(website))
        website_save_path = os.path.join(ROOT_SAVE_PATH, website)
        if not os.path.exists(website_save_path):
            os.mkdir(website_save_path)

        website_html_files = os.path.join(website_path, html_file)
                             for html_file in os.listdir(website_path)
                             if os.path.isdir(website path)]

        for html_file_path in website_html_files:
            try:
                html_file = open(html_file_path, 'r', encoding="utf-8")
                html content = str(soup(html file.read(),
'html.parser'))
                text_content = h.handle(html_content)
```

```
                html_file_name = os.path.basename(html_file_path)
                text_file_save_path = os.path.join(website_save_path,
html_file_name)

                destination_file = open(text_file_save_path, 'w')
                destination_file.write(text_content)
                destination_file.close()

            except Exception as e:
                print(website, e)


if __name__ == "__main__":
    main()
```

5. Normalized Google Distance

```
import argparse
import math
import json

import pandas as pd

import search_engine

global word_occurrence_dict


def get_number_of_hits(keyword):
    global word_occurrence_dict

    if len(keyword) == 1:
        if keyword0] in word_occurrence_dict.keys():
            number_of_hits = sum(1 for value in
word_occurrence_dictkeyword0]].values() if value != 0])
            return keyword0], number_of_hits

        edited = ' {}'.format(keyword0])
        if edited in word_occurrence_dict.keys():
            number_of_hits = sum(1 for value in
word_occurrence_dictedited].values() if value != 0])
            return edited, number_of_hits

        else:
            print('{} not in the keywords'.format(keyword0]))
            return keyword0], -1

    else:
        first_word = keyword0]
        second_word = keyword1]
        number_of_hits = sum(
            1 for value1, value2 in
zip(word occurrence dictfirst word].values(),
word_occurrence_dictsecond_word].values())
             if value1 != 0 and value2 != 0]
        )
        return number_of_hits


def get_normalized_distance(w1, w2):
    N = 587.0  # Number of results for "the", proxy for total pages
    N = math.log(N, 2)
    if w1 != w2:
        word1, w1_hits = get_number_of_hits(w1])
        if w1_hits == -1:
            return 1

        word2, w2_hits = get_number_of_hits(w2])
        if w2_hits == -1:
            return 1

        w12_hits = get_number_of_hits(word1, word2])
```

```
        if w1_hits == 0 or w2_hits == 0 or w12_hits == 0:
            return 1

        f_w1 = math.log(w1_hits, 2)
        f_w2 = math.log(w2_hits, 2)
        f_w1_w2 = math.log(w12_hits, 2)

        normalized_distance = \
            (max(f_w1, f_w2) - f_w1_w2) / (N - min(f_w1, f_w2))
        return normalized_distance

    else:
        return 0


def init_search_engine(website_folder):
    search_engine.ROOT_PATH = website_folder
    search_engine.load_data()


def load data(dfdict path):
    global word_occurrence_dict
    word_occurrence_dict_file = open(dfdict_path, 'r')
    word_occurrence_dict = json.load(word_occurrence_dict_file)
    return word_occurrence_dict


def main():
    parser = argparse.ArgumentParser()
    parser.add_argument(
        '--words',
        help='Path to words',
        required=True,
    )
    parser.add_argument(
        '--dfdict',
        help='Path to Document Frequency Dict',
        required=True,
    )
    arguments = parser.parse_args()

    load_data(arguments.dfdict)

    csv_file = pd.read_csv('./words.txt', names='first', 'second'])
    nds = ]
    for index, row in csv_file.iterrows():
        first = str(row.iloc0]).strip()
        second = str(row.iloc1]).strip()
        ND = get_normalized_distance(first, second)
        nds.append(ND)

    csv_file'nd'] = nds
    csv_file.to_csv('ngds.csv')
    # for count, line in enumerate(lines):
    #     if (count + 1) % 3 != 0:
    #         words.append(line.strip())
    #
```

```
    #     if len(words) == 2:
    #         print('computing NGD for words {}'.format(words))
    #         result_file.write('words : {}\n'.format(words))
    #         result_file.write(
    #             'NGD for words {}, {}: {}\n\n'.format(
    #                 words0],
    #                 words1],
    #                 get_normalized_distance(words0], words1]),
    #             )
    #         )
    #         words = ]
    #
    # json_value = json.dumps(tf_idf_dict)
    # f = open("dict2.json", "w")
    # f.write(json_value)
    # f.close()


if __name__ == "__main__":
    main()
```

6. Search Engine

```
import os
import itertools


ROOT_PATH = './websites'
global websites
global website_pages


def load_data():
    global websites, website_pages
    website_pages = {}
    websites = website
                for website in os.listdir(ROOT_PATH)
                if os.path.isdir(os.path.join(ROOT_PATH, website))]

    for website in websites:
        website_pageswebsite] = {}
        website_path = os.path.join(ROOT_PATH, website)
        pages = os.listdir(website_path)
        for page in pages:
            f = open(os.path.join(website_path, page), 'r')
            contents = f.read()
            website_pageswebsite]page] = contents


def number_of_documents():
    return len(websites)


def get different cases(word):
    tokens = word.split(' ')
    all_cases = list(map(''.join,
itertools.product(*((token.capitalize()+' ', token.lower()+' ') for
token in tokens))))
    all cases = item:-1] for item in all cases]
    return all_cases


def check_occurrence_by_folder(words, folder):
    global website pages
    included_words = {}
    number_of_hits = 0
    for key in website_pagesfolder].keys():
        contents = website_pagesfolder]key]
        for word in words:
            all_cases = get_different_cases(word)
            for case in all_cases:
                if case in contents:
                    included_wordsword] = 1
                    number_of_hits += contents.count(case)

    if len(included_words) == len(words):
        return number_of_hits
```

```
    return 0


def get_number_of_hits(words):
    global websites
    pages_hit = 0
    page_hit_info = {}
    for website in websites:
        hits_per_page = check_occurrence_by_folder(words, website)
        if hits_per_page != 0:
            pages_hit += 1

        website_base_name = os.path.basename(website)
        page_hit_infowebsite_base_name] = hits_per_page

    return pages_hit, page_hit_info
```

7. Taxonomy

```
import view


class Node(object):
    def __init__(self, data, row_number, depth):
        self.data = data
        self.row_number = row_number
        self.depth = depth
        self.children = ]
        self.parent = None

    def add_child(self, node):
        self.children.append(node)

    def set_parent(self, node):
        self.parent = node


class Tree(object):
    def __init__(self):
        self.root = Node('root', row_number=-1, depth=1)
        self.all_nodes = ]
        self.root.set_parent(None)
        self.max_depth = 0

    def print_tree(self, node):
        for child in node.children:
            print(node.depth, node.data, child.data)

        for child in node.children:
            self.print tree(child)

    def _plot_node(self, node, html_content):
        if len(node.children) == 0:
            html_content += '<li>{}-{}</li>\n'.format(node.depth,
node.data)
            return html_content

        html_content += '<li>{}-{}\n'.format(node.depth, node.data)
        html_content += '<ul>\n'
        for child in node.children:
            html_content = self._plot_node(child, html_content)

        html_content += '</li>\n'
        html_content += '</ul>\n'
        return html_content

    def plot_tree(self):
        f = open('tree.html', 'w')
        html_content = '''
            <!DOCTYPE html>
            <html>
            <head>
            <style>
            {}
```

```
            </style>
            </head>
            <body>\n
        '''.format(view.css_content)
        html_content += ' <ul class="tree">\n'
        html_content = self._plot_node(self.root, html_content) + '\n'
        html_content += '</ul>\n'
        html_content += '</body>\n</html>'
        f.write(html_content)
```

8. TF/IDF

```
import json
import math


def get_word_tf_idf(word_occurrence_dict):
    tf_idf_dict = {}
    N = len(word_occurrence_dict.values())
    number_of_documents = sum(1 for value in
word_occurrence_dict.values() if value != 0])
    idf = math.log(N/min(number_of_documents+1, N))
    for key, value in word_occurrence_dict.items():
        tf = math.log(1+value)
        wtf_idf = tf * idf
        tf_idf_dictkey] = wtf_idf

    return tf_idf_dict


def main():
    word_occurrence_dict_file = open('./word_occurrence_dict.json',
'r')
    tf_idf_dict = {}
    word_occurrence_dict = json.load(word_occurrence_dict_file)
    for word in word_occurrence_dict.keys():
        word_tf_idf_dict = get_word_tf_idf(word_occurrence_dictword])
        tf_idf_dictword] = word_tf_idf_dict

    json_value = json.dumps(tf_idf_dict)
    tf_idf_file = open("tf_idf_dict.json", "w")
    tf_idf_file.write(json_value)
    tf idf file.close()


if __name__ == "__main__":
    main()
```

REFERENCES

Ameri, F., & Bernstein, W. (2017). A Thesaurus-Guided Framework for Visualization of Unstructured Manufacturing Capability Data. In H. Lödding, R. Riedel, K.-D. Thoben, G. von Cieminski, & D. Kiritsis (Eds.), *Advances in Production Management Systems. The Path to Intelligent, Collaborative and Sustainable Manufacturing* (Vol. 513, pp. 202–212). Springer International Publishing. https://doi.org/10.1007/978-3-319-66923-6_24

Ameri, F., Yoder, R., & Zandbiglari, K. (2020). SKOS Tool: A Tool for Creating Knowledge Graphs to Support Semantic Text Classification. In B. Lalic, V. Majstorovic, U. Marjanovic, G. von Cieminski, & D. Romero (Eds.), *Advances in Production Management Systems. Towards Smart and Digital Manufacturing* (pp. 263–271). Springer International Publishing. https://doi.org/10.1007/978-3-030-57997-5_31

Bhavsar, H., & Panchal, M. H. (n.d.). *A Review on Support Vector Machine for Data Classification*. *1*(10), 5.

Brundage, M. P., Sexton, T., Hodkiewicz, M., Dima, A., & Lukens, S. (2021). Technical language processing: Unlocking maintenance knowledge. *Manufacturing Letters*, *27*, 42–46.

Cilibrasi, R., & Vitanyi, P. M. B. (2007). The Google Similarity Distance. *ArXiv:Cs/0412098*. http://arxiv.org/abs/cs/0412098

Detecting Trends Using Spearman's Rank Correlation Coefficient | Elsevier Enhanced Reader. (n.d.). https://doi.org/10.1006/enfo.2001.0061

Dong, L., Srimani, P. K., & Wang, J. Z. (2010). WEST: Weighted-Edge Based Similarity Measurement Tools for Word Semantics. *2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, 216–223. https://doi.org/10.1109/WI-IAT.2010.39

Gorman, J., & Curran, J. R. (2006). Scaling Distributional Similarity to Large Corpora. *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, 361–368. https://doi.org/10.3115/1220175.1220221

Gray, P., & F. Bjorklund, D. (n.d.). *Psychology* (Eighth).

Jiang, J. J., & Conrath, D. W. (n.d.). *Semantic Similarity Based on Corpus Statistics and Lexical Taxonomy*. 15.

Jiang, J. J., & Conrath, D. W. (1997). Semantic Similarity Based on Corpus Statistics and Lexical Taxonomy. *Proceedings of the 10th Research on Computational Linguistics International Conference*, 19–33. https://aclanthology.org/O97-1002

Jiang, Y., Zhang, X., Tang, Y., & Nie, R. (2015). Feature-based approaches to semantic similarity assessment of concepts using Wikipedia. *Inf. Process. Manag.* https://doi.org/10.1016/J.IPM.2015.01.001

Jung, K., Kulvatunyou, B., Choi, S., & Brundage, M. P. (2017). An Overview of a Smart Manufacturing System Readiness Assessment. *IFIP Advances in Information and Communication Technology*, *488*, 705–712. https://doi.org/10.1007/978-3-319-51133-7_83

Korde, V. (2012). Text Classification and Classifiers:A Survey. *International Journal of Artificial Intelligence & Applications*, *3*(2), 85–99. https://doi.org/10.5121/ijaia.2012.3208

Kowsari, K., Jafari Meimandi, K., Heidarysafa, M., Mendu, S., Barnes, L., & Brown, D. (2019). Text Classification Algorithms: A Survey. *Information*, *10*(4), 150. https://doi.org/10.3390/info10040150

Landauer, T. K., & Dumais, S. T. (1997). A solution to Plato′s problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review*, *104*(2), 211.

Landauer, T. K., Foltz, P. W., & Laham, D. (1998). An introduction to latent semantic analysis. *Discourse Processes*, *25*(2–3), 259–284. https://doi.org/10.1080/01638539809545028

Lastra-Díaz, J. J., Goikoetxea, J., Hadj Taieb, M. A., García-Serrano, A., Ben Aouicha, M., & Agirre, E. (2019). A reproducible survey on word embeddings and ontology-based methods for word similarity: Linear combinations outperform the state of the art. *Engineering Applications of Artificial Intelligence*, *85*, 645–665. https://doi.org/10.1016/j.engappai.2019.07.010

Leacock, C., & Chodorow, M. (1998). Combining Local Context and WordNet Similarity for Word Sense Identification. In *WordNet: An Electronic Lexical Database* (Vol. 49, p. 265).

Li, W., Gao, S., Zhou, H., Huang, Z., Zhang, K., & Li, W. (2019). The Automatic Text Classification Method Based on BERT and Feature Union. *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*, 774–777. https://doi.org/10.1109/ICPADS47876.2019.00114

Li, Y., Bandar, Z. A., & Mclean, D. (2003). An approach for measuring semantic similarity between words using multiple information sources. *IEEE Transactions on Knowledge and Data Engineering*, *15*(4), 871–882. https://doi.org/10.1109/TKDE.2003.1209005

Martis, R., Acharya, U. R., Lim, C., Mandana, K., Ray, A., & Chakraborty, C. (2013). Application of higher order cumulant features for cardiac health diagnosis using ECG signals. *International Journal of Neural Systems*, *23*, 1350014. https://doi.org/10.1142/S0129065713500147

McInnes, B., Pedersen, T., Pakhomov, S., Liu, Y., & Melton-Meaux, G. (2013). UMLS::Similarity: Measuring the Relatedness and Similarity of Biomedical Concepts. *Proceedings of the 2013 NAACL HLT Demonstration Session*, 28–31. https://www.aclweb.org/anthology/N13-3007

Mohammad, S. M., & Hirst, G. (2012). Distributional Measures of Semantic Distance: A Survey. *ArXiv:1203.1858 [Cs]*. http://arxiv.org/abs/1203.1858

Rada, R., Mili, H., Bicknell, E., & Blettner, M. (1989). Development and application of a metric on semantic nets. *IEEE Transactions on Systems, Man, and Cybernetics*, *19*(1), 17–30. https://doi.org/10.1109/21.24528

Resnik, P. (1995). Using Information Content to Evaluate Semantic Similarity in a Taxonomy. *ArXiv:Cmp-Lg/9511007*. http://arxiv.org/abs/cmp-lg/9511007

Sabbagh, R., & Ameri, F. (2018). Supplier clustering based on unstructured manufacturing capability data. *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, *51739*, V01BT02A036.

Sabbagh, R., Ameri, F., & Yoder, R. (2018). Thesaurus-guided text analytics technique for capability-based classification of manufacturing suppliers. *Journal of Computing and Information Science in Engineering*, *18*(3).

Saeys, Y., Inza, I., & Larranaga, P. (2007). A review of feature selection techniques in bioinformatics. *Bioinformatics (Oxford, England)*, *23*, 2507–2517. https://doi.org/10.1093/bioinformatics/btm344

Sánchez, D., Batet, M., Isern, D., & Valls, A. (2012). Ontology-based semantic similarity: A new feature-based approach. *Expert Systems with Applications*, *39*, 7718–7728. https://doi.org/10.1016/j.eswa.2012.01.082

Sharma, R., Agarwal, R., & Arora, A. (2016). Evaluation of Ultrasonic Transducer with Divergent Membrane Materials and Geometries. In A. Unal, M. Nayak, D. K. Mishra, D. Singh, & A. Joshi (Eds.), *Smart Trends in Information Technology and Computer Communications* (pp. 779–787). Springer. https://doi.org/10.1007/978-981-10-3433-6_93

Shotorbani, P. Y., Ameri, F., Kulvatunyou, B., & Ivezic, N. (2016). A Hybrid Method for Manufacturing Text Mining Based on Document Clustering and Topic Modeling Techniques. In I. Nääs, O. Vendrametto, J. Mendes Reis, R. F. Gonçalves, M. T. Silva, G. von Cieminski, & D. Kiritsis (Eds.), *Advances in Production Management Systems. Initiatives for a Sustainable World* (Vol. 488, pp. 777–786). Springer International Publishing. https://doi.org/10.1007/978-3-319-51133-7_91

Soğancıoğlu, G., Öztürk, H., & Özgür, A. (2017). BIOSSES: A semantic sentence similarity estimation system for the biomedical domain. *Bioinformatics*, *33*(14), i49–i58. https://doi.org/10.1093/bioinformatics/btx238

SONG, Y., & LU, Y. (2015). Decision tree methods: Applications for classification and prediction. *Shanghai Archives of Psychiatry*, *27*(2), 130–135. https://doi.org/10.11919/j.issn.1002-0829.215044

Wang, F., Wang, Z., Li, Z., & Wen, J.-R. (2014). Concept-based Short Text Classification and Ranking. *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, 1069–1078. https://doi.org/10.1145/2661829.2662067

Wang, J., & Dong, Y. (2020). Measurement of Text Similarity: A Survey. *Information*, *11*(9), 421. https://doi.org/10.3390/info11090421

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., & Le, Q. V. (2020). XLNet: Generalized Autoregressive Pretraining for Language Understanding. *ArXiv:1906.08237 [Cs]*. http://arxiv.org/abs/1906.08237

Zandbiglari, K., Ameri, F., & Javadi, M. (2021, November 17). *Capability Language Processing (CLP): Classification and Ranking of Manufacturing Suppliers Based on Unstructured Capability Data*. ASME 2021 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. https://doi.org/10.1115/DETC2021-71308

Zhu, G., & Iglesias, C. A. (2016). Computing semantic similarity of concepts in knowledge graphs. *IEEE Transactions on Knowledge and Data Engineering*, *29*(1), 72–85.