B.Sc. in Computer Science and Engineering Thesis

Knowledge Graph-Based Categorization of Newspaper Articles in a Newspaper Corpus

Submitted by

Istiak Bin Mahmod
201705073

Soham Khisa
201705120

Supervised by

Dr. Muhammad Masroor Ali

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology

Dhaka, Bangladesh

May 2023

CANDIDATES' DECLARATION

This is to certify that the work presented in this thesis, titled, "Knowledge Graph-Based Categorization of Newspaper Articles in a Newspaper Corpus", is the outcome of the investigation and research carried out by us under the supervision of Dr. Muhammad Masroor Ali.

It is also declared that neither this thesis nor any part thereof has been submitted anywhere else for the award of any degree, diploma or other qualifications.

Istiak Bin Mahmod
201705073

Soham Khisa
201705120

ACKNOWLEDGEMENT

We would like to express our deep appreciation to our supervisor Prof. Dr. Muhammad Masroor Ali for his guidance, encouragement and for his valuable advices throughout this study. It was really a great chance for us to work with such a thoughtful, friendly and motivating supervisor.

Finally, we sincerely thank our parents for their love and support.

Dhaka
May 2023

Istiak Bin Mahmod

Soham Khisa

Contents

List of Figures

List of Tables

List of Algorithms

ABSTRACT

The process of enriching newspaper articles with structured and semantically rich information is known as newspaper annotation. This technique enhances the accuracy and relevance of information retrieval, which is particularly important given the increasing number of news sources available on the internet. While search engines like Google offer a common method for finding information, they often produce imprecise results. Semantic search, on the other hand, utilizes knowledge graphs to address this problem. A knowledge graph represents real-world entities and their relationships in a graph database and is typically based on ontologies that provide a formal representation of the entities in the graph.

Our thesis proposes a newspaper categorization system that utilizes ontology to extract metadata from unstructured HTML news articles. We use the IPTC Media Topics taxonomy, which categorizes text using over 1200 terms, to categorize the articles based on ontology concepts.

Chapter 1

Introduction

1.1 Motivation

The amount of information on the Internet is growing rapidly as well as the number of new inexperienced users. A general-purpose search engine like Google is designed to search for information across the entire Internet and they use algorithms that are optimized for this purpose. While this makes them very effective at finding information on a wide range of topics, it also means that they can have limitations when searching within a particular domain.

A general-purpose search engine is not designed to acquire a specific piece of information on the web but instead attempts to cover as much as possible on the web in order to answer and handle all possible queries. A domain-specific search engine, on the other hand, can be defined as a specialized search engine that is designed to cover only a certain part of the web, with the same or similar category in the domain of interest. Although a general-purpose search engine has the advantage of broad coverage - as it tries to answer everything just like an encyclopedia would - but its relevancy is usually low. Simple queries with keyword matching often generate too many search results, which are then difficult and sometimes confusing for users to choose from. The coverage of a domain-specific search engine is small and cannot answer queries outside of its domain of interest. But its relevancy is very high, as the entire search results are within the domain.

To address these issues, our thesis proposes the use of an ontology to annotate newspaper articles and extract metadata using the IPTC Media Topics taxonomy [1]. This will create semantically-rich documents that can be used to categorize news articles using widely-accepted concepts in the News Industry. By categorizing articles in this way, they can be utilized in domain-specific search engines, providing users with more relevant and accurate results. The motivation for our study is to improve the accuracy and efficiency of searching for information within a specific domain.

1.2 Objectives of the Thesis

The objectives of our thesis are:

i) Developing an ontology based on the news industry's concepts and terminology.

ii) Automating the extraction of metadata from newspaper articles using the ontology

iii) Automatically categorizing articles based on widely accepted concepts in the news industry

1.3 Outline of Methodology

The following steps outline the sequential approach used in our study:

i) We constructed an ontology based on widely accepted concepts and terminology of the news industry. Our ontology should have two levels of classes. We want to categorize the articles based on the level-1 classes.

ii) We used a proper data source and filtered out some of the data that were inappropriate for our study. We primarily focused on two columns, the content of an article and the category it belongs to, which is aligned with our level-1 class. We removed irrelevant information from the contents of the input articles and stemmed the contents.

iii) We created a text document for each ontology class, which incorporates common and related words and phrases that are generally used in news articles. The text document is topic-specific, meaning they only include the words and phrases that are related to the corresponding class the document represents.

iv) We used a pipeline of different processing resources for the purpose of finding useful annotations of the input articles. We integrated the ontology with the document lists in the pipeline. The pipeline executes the input articles and emanates annotated versions of the input articles.

v) Finally we devised an algorithm that operates on the annotated articles and gives a result of which input article belongs to which category or ontology class.

A comprehensive explanation of these steps is provided in Chapter 4.

1.4 Contribution

The contributions we make through this thesis are summarized as follows:

i) We developed a novel framework that enables a more extensive classification of news articles, encompassing a broader range of topics than previous research works.

ii) We used a set of 17 categories that encompassed all level-1 topics of IPTC NewsCodes, resulting in a broader coverage of news article topics for enhanced categorization accuracy.

iii) We incorporated explicit domain knowledge via an ontology-driven strategy, ensuring accurate and interpretable categorization outcomes.

1.5 Organization of the Thesis

The rest of the thesis is organized as follows. Chapter 2 gives brief background information about the terminologies and technologies used in this thesis. Chapter 3 presents the existing works related to our problem. Chapter 4 gives the details of the components of our system and explains the overall methodology of the research. Chapter 5 presents the evaluation results. Chapter 6 gives a conclusion to our thesis, discussion, and future work.

Chapter 2

Background Information

In this chapter, we define some basic terms and concepts related to newspaper annotation, semantic web, knowledge graph, and ontology.

2.1 Newspaper Categorization

The fast spread of the Internet has led to a substantial increase in the number of news sources available to consumers. As a result, the average recipient is receiving an overwhelming amount of news items daily. Unfortunately, this increase in the availability of news sources has also created challenges in managing metadata, which is currently quite heterogeneous and often insufficient to capture all the relevant information contained in news documents. Consequently, this presents a significant challenge for news classification, as inaccurate or incomplete metadata can impede the dissemination of important news stories to their intended audience. While manual annotation is impractical and prone to errors, automatic annotation tools remain largely underdeveloped.

Effective metadata management is crucial for newspaper classification and journalism, as news articles must be appropriately tagged and categorized to ensure proper organization and archiving for future use. However, the manual labor involved in categorizing news articles is susceptible to various errors and omissions, given the varying perspectives and expertise levels of staff members. Thus, there is a pressing need for specialized knowledge service tools that can search and extract specific knowledge directly from unstructured text on the Web, and automatically categorize news articles under pre-defined sets of codes.

2.2 Semantic Web

The Semantic Web is a concept that aims to expand the usefulness of the current World Wide Web by providing machine-readable metadata to software programs by enhancing existing information and data. While the majority of web material is intended for human consumption, the Semantic Web attempts to organize content in such a manner that computers can read and process information in the same way that humans do, enabling automated tasks [2]. According to Tim Berners-Lee, the Semantic Web's creator, the ultimate goal is to improve computers' ability to manipulate information. Structured collections of information and sets of inference rules must be accessible to computers for automated reasoning in order to achieve this. The objective is to create a language that conveys both data and rules, allowing knowledge representation system rules to be exported to the Web. Berners-Lee proposed the Semantic Web Stack model to show the tools and actions required for Semantic Web enablement [3]. This paradigm goes beyond conventional hyperlinking by providing a framework for linking data and information across several web sources.

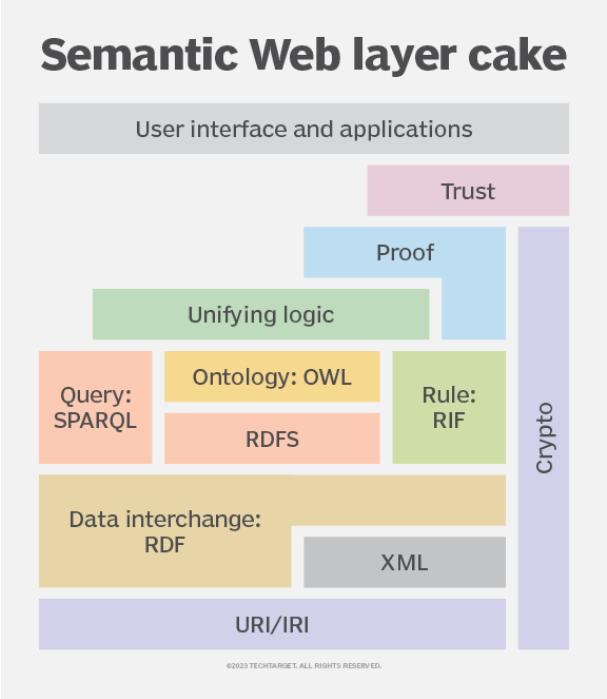


Figure 2.1: Illustration of Semantic Web Stack.

The Semantic Web Stack is made up of several layers, each of which builds on the preceding one to provide a full semantic processing infrastructure. Raw data is represented at the bottom of the stack using Unicode text characters and connected using Uniform Resource Identifiers (URIs). The Resource Description Framework (RDF) provides a standardized mechanism to define entities, characteristics, and their relationships for data interchange, and is often used to organize data in a machine-readable format. The Web Ontology Language (OWL) is used to express entity and relationship information, whereas the Rule Interchange Format (RIF) rep-

resents more sophisticated and difficult connections. The SPARQL query language is used in combination with RDF and OWL to search for and retrieve information saved across several sources. Other technologies are also integrated with these basic semantic processing services to assure data security, trust enforcement, and a consistent user experience.

The original vision of the Semantic Web revolved around three concepts: automation of information retrieval, the Internet of Things, and personal assistants. Over time, these concepts have evolved into two crucial data types that currently realize the vision of the Semantic Web: Linked Open Data and Semantic Metadata [4].

Linked Open Data (LOD) represents structured data modeled as a graph and published in a manner that enables cross-server interlinking. In 2006, Tim Berners-Lee formalized this concept with the Four Rules of Linked Data. These rules emphasize the use of URIs as entity identifiers, HTTP URIs for simple retrieval, the provision of useful information using RDF and SPARQL standards, and the inclusion of links to other URIs for additional exploration. LOD enables both humans and machines to access and interpret data from various servers, expanding the Semantic Web beyond linked documents to linked information. This creates a network of connected, machine-processable meaning, facilitating novel automation and information retrieval possibilities.

Linked Open Data includes factual information about particular entities and concepts, such as the human genome, the Eiffel Tower, and alternative energy sources. It also includes ontologies, which are semantic structures that define object classes (such as Product, Service, and Sport), relationship categories (such as owned by, located in, and studied by), and attributes (such as price, weight, and temperature). This data organization facilitates the interconnection of information across servers and the development of a network of machine-processable knowledge.

Semantic metadata refers to the incorporation of semantic elements into standard web pages in order to enrich their meaning. A web page about a film can be semantically annotated by adding references to relevant entities and concepts, such as the film's title, director, actors, and genre. This metadata plays a vital role in augmenting the precision of search results based on semantic criteria, thereby reducing the likelihood of confusion or ambiguity. Using semantic metadata, a search for the term "Jaguar" can retrieve pages pertaining to either the automobile brand or the animal, depending on the intended meaning.

2.3 Knowledge Graph

In recent years, the concept of a *Knowledge Graph* has garnered considerable attention, particularly in the fields of research and business, frequently in conjunction with Semantic Web technologies, linked data, large-scale data analytics, and the cloud. The increase in prominence

can be partially attributed to the 2012 introduction of Google's Knowledge Graph. Despite its ubiquitous use, however, there is no universally accepted definition of the term [5].

A knowledge graph, also referred to as a semantic network, is a graphical representation of real-world entities such as objects, events, situations, and concepts, as well as the relationships between them. This information is typically recorded in a graph database and represented as a graph structure, hence the name "knowledge graph." Nodes, edges, and labels are the fundamental building blocks of a knowledge graph. Nodes represent various objects, locations, or people, while edges define their connections. For instance, a node could represent a client such as IBM, and an edge could depict a customer relationship linking IBM to an agency such as Ogilvy [6].

A directed labeled graph can be represented as a 4-tuple $G = (N, E, L, f)$, where N is a set of nodes, $E \subseteq N \times N$ is a set of edges, L is a set of labels and $f : E \to L$ is a mapping function from edges to labels. An edge with a label B connecting nodes A and C can be represented as a triple (A, B, C) and visualized as shown in Figure 2.2.

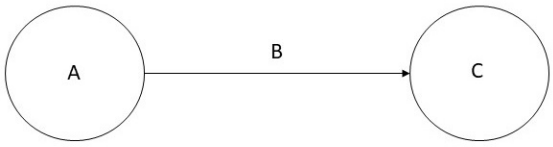


Figure 2.2: Example of a directed labeled graph.

A knowledge graph is a directed, labeled graph whose nodes and edges incorporate domain-specific semantics. Edge labels capture the relationships between nodes, such as alliances between individuals, customer relationships between companies and individuals, and network connections between computers.

Depending on the specifications of an application, the representation of a directed labeled graph can be utilized in numerous ways. For example, a data graph can be constructed with nodes representing individuals and edges representing parent-child relationships. Likewise, a taxonomy can be constructed with nodes representing object classes and edges representing subclass relationships. In some data structures, the subject, predicate, and object of a triple (A, B, C) are referred to as the subject, predicate, and object, respectively.

A knowledge graph is a data structure for preserving information that can be contributed manually, automatically, or through a combination of the two. The recorded information is expected to be readily understandable and verifiable by humans, regardless of the method used.

It is essential to navigate the graph in order to conduct meaningful computations on it. Determining the friends of friends of a person A in a friendship knowledge graph, for instance, entails traversing the graph from A to all nodes B connected to it through a friend relation, and

then recursively to all nodes C connected to each B through a friend relation.

A fact is the most foundational element of stored information in a Knowledge Graph (KG). These facts can be represented as triplets in either HRT (<head, relation, tail>) or SPO (<subject, predicate, object>) formats. A fact consists of three components: the head or tail, which represents real-world entities as nodes, and the relation, which represents the connection between entities as edges in a graph [7].

Figure 2.3 provides a simple illustration of a KG, in which <BoB, is_interested_in, The_Mona_Lisa> is an example of a fact. Essentially, the KG is a compilation of multiple such truths. The incorporation of persons (Bob, Alice, etc.), paintings (Mona Lisa), and dates, all represented by nodes in the KG, demonstrates that there are no restrictions on the data types of facts recorded in a KG.

As the most fundamental unit of information in a KG, a fact can be expressed as a triplet in two distinct ways:

i. HRT: <head, relation, tail>

ii. SPO: <subject, predicate, object>

Facts consist of three elements, commonly known as triplets, which contribute to the intuitive representation of a KG as a graph.

i) Head or tail: These represent real-world entities or abstract concepts and are visualized as nodes in the graph.

ii) Relation: This signifies the connection between entities and is depicted as edges in the graph.

Figure 2.3 depicts a simplified KG example. In this KG,<BoB, is_interested_in, The_Mona_Lisa> is a potential fact. Consequently, a KG can be viewed as a collection of such truths.

2.4 Ontology

An ontology is a structured and formal representation of knowledge that defines a set of concepts and their relationships within a particular domain. It includes formally specified components such as individuals (instances of objects), classes, attributes, relations, restrictions, rules, and axioms. Ontologies serve as reusable and shareable representations of knowledge and enable the addition of new domain-specific information.

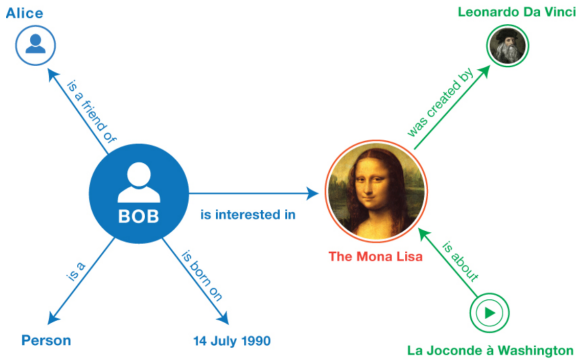


Figure 2.3: Example of a simple knowledge graph.

Despite differences in language, modern ontologies share many structural features, such as describing *individuals*, *classes*, *attributes*, *relations*, *function terms*, *restrictions*, *rules*, *axioms*, and *events*. *Individuals* represent the basic objects, while *classes* correspond to sets or collections of objects. *Attributes* refer to the properties or features of objects, and *relations* describe the ways in which classes and individuals are related to each other. *Function terms* enable complex structures to be utilized in place of individual terms, and *restrictions* provide formal descriptions of the conditions that must be true for an assertion to be accepted as input. *Rules* describe the logical inferences that can be drawn from an assertion in a particular form, while *axioms* consist of assertions (including rules) in a logical form that together form the overall theory that the ontology describes in its domain of application.

An example of an ontology for pets can illustrate these components:

i) **Individuals**: Fluffy (dog), Whiskers (cat), Bubbles (fish)

ii) **Classes**: Dog, Cat, Fish, Pet

iii) **Attributes**: Name, Age, Color, Breed

iv) **Relations**: IsA (Dog is a Pet), Has (Pet has Name), Owning (Person owns Pet)

v) **Function terms**: None

vi) **Restrictions**: A Pet must have exactly one Name attribute

vii) **Rules**: If a Person owns a Pet, then the Pet is dependent on that Person

viii) **Axioms**: A Pet must have a Name attribute, A Pet cannot simultaneously be a Dog and a Cat

ix) **Events**: Change in Age of a Pet, Change in Ownership of a Pet

This ontology defines the concepts of pets, including the subclasses of Dog, Cat, and Fish. It also includes attributes such as Name, Age, Color, and Breed that pets can have, and relations

like Owning, which connects a Pet to the Person who owns it. The ontology also contains restrictions, such as ensuring that a Pet has exactly one Name attribute, and rules like defining the dependency of a Pet on its owner. Finally, the ontology includes axioms such as requiring that a Pet must have a Name attribute and cannot be both a Dog and a Cat simultaneously.

Ontology languages, such as *Web Ontology Language* (OWL), are commonly used to encode ontologies.

2.4.1 Web Ontology Language

In recent years, the use of Web Ontology Language (OWL) and other ontology languages to express ontologies has increased significantly. OWL is a computational logic-based language designed specifically for the semantic web. Its primary purpose is to record intricate and exhaustive information about objects and their relationships. OWL excels at creating precise, consistent, and significant distinctions between object classes, properties, and relationships.

OWL significantly enhances ontology modeling in semantic graph databases, such as RDF triplestores, by delineating object classes, relationship properties, and their hierarchical structures. When combined with an OWL reasoner in triplestores, OWL enables consistency checks to identify logical contradictions and satisfiability tests to determine whether object classes can exist without instances. In addition, OWL provides instruments for establishing the equivalence and distinction of instances, classes, and properties. These relationships facilitate concept matching across multiple data sources, even when they describe concepts differently. In addition, they assure the distinction between instances with identical names or descriptions.

Here is an example of an OWL file:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:owl="http://www.w3.org/2002/07/owl#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema#">

  <!-- Ontology definition -->
  <owl:Ontology rdf:about="http://example.com/ontology"/>

  <!-- Class definitions -->
  <owl:Class rdf:about="http://example.com/ontology/Person"/>

  <!-- Property definitions -->
  <owl:ObjectProperty rdf:about="http://example.com/ontology/hasFriend"/>
  <owl:DatatypeProperty rdf:about="http://example.com/ontology/hasAge"/>

  <!-- Individual instances -->
```

```
  <owl:NamedIndividual rdf:about="http://example.com/ontology/John">
    <rdf:type rdf:resource="http://example.com/ontology/Person"/>
    <hasFriend rdf:resource="http://example.com/ontology/Jane"/>
    <hasAge rdf:datatype="xsd:integer">30</hasAge>
  </owl:NamedIndividual>

</rdf:RDF>
```

This example demonstrates the use of OWL to define classes, properties, and individual instances in an ontology. The ontology represents a simple concept of a ”Person” class with properties like ”hasFriend” and ”hasAge.” An individual instance named ”John” is defined with the ”Person” class and assigned the values of having a friend named ”Jane” and an age of 30.

OWL’s expressiveness and features make it a powerful language for constructing ontologies and leveraging semantic reasoning capabilities in knowledge representation and retrieval systems.

Chapter 3

Related Work

This chapter provides an overview of previous work on ontology-based annotation techniques that have been used for the purpose of this thesis. Each work is described with a focus on the methods used. By surveying this related work, we aim to establish the context for our research and identify the gaps that our study aims to fill.

3.1 Neptuno: Semantic Web Technologies for a Digital Newspaper Archive

This paper proposes the introduction of semantic-based technologies to improve the processes of creation, maintenance, and exploitation of the digital archive of a newspaper. The Neptuno Project [8] has been set up to apply Semantic Web technologies to improve the production and consumption of digital news. The goal of the project is to develop a high-quality semantic archive for Diari SEGRE newspaper where a) reporters and archivists have expressive means to describe and annotate news materials, b) reporters and readers are provided with better search and browsing capabilities than those currently available, c) the archive system is open to integration in potential electronic marketplaces of news products. The main components of the project are

i) An ontology for archive news, based on journalists' and archivists' expertise and practice and integrating current dominant standards from the IPTC consortium.

ii) A knowledge base where archive materials are described using the ontology. A DB-toontology conversion module automatically integrates existing legacy archive materials into the knowledge base.

iii) A semantic search module, where meaningful information needs can be expressed in

terms of the ontology, and more accurate answers are supplied.

iv) A visualization and navigation module to display individual archive items, parts or combinations of items, and lists or groups of items.

The Neptuno project has adopted the IPTC Subject Reference System as a thematic classification system for news archive contents. The integration and adaptation of this standard to our ontology has been carried through by a) converting the IPTC topic hierarchy to an RDF class hierarchy, and b) establishing a mapping between the classification system (thesaurus) previously in use at Diari SEGRE, and the IPTC standard. In order to represent the actual archive contents, they have built their own ontology. The ontology to represent the archives of Diari SEGRE has been built by using the Protégé ontology editor. Three ways of classifying contents have been included. In one of them, contents are classified by subject following the IPTC subject classification hierarchy. An alternative classification can be made according to contents genre, which has to do with the nature (breaking news, summary, interview, opinion, survey, forecast, etc.) of a news or photograph rather than its specific contents. Finally, a content can be classified according to some keywords that describe it.

Figure 3.1: Navigation and search in the Neptuno platform

A semantic search engine has knowledge of the domain at hand. The availability of a domain ontology allows a search system where users can specify search criteria in terms of modeled concepts and attributes. The results are presented in a structured form including only the requested information.

3.2 Artequekt project

The aim of the Artequakt project is to automatically extract knowledge about artists from the Web to populate a knowledge base and use it to generate personalized narrative biographies [9]. Figure 3.2 illustrates Artequakt's architecture. The Artequakt project links a knowledge extraction tool with the Artequakt Ontology to guide the extraction mechanism. The extraction tool searches online documents and extracts knowledge about artists based on the Artequakt ontology and WordNet lexicons. Artequakt ontology represents the domain of artists and artifacts. The Artequakt ontology is a modified version of the CIDOC Conceptual Reference Model (CRM2) ontology. This information is stored in a knowledge base that inspects for duplication. The project also developed narrative construction tools to query the knowledge base through an ontology server to search and retrieve relevant information to generate a specific biography. The Artequakt server takes requests from a reader via a Web interface. The request usually consists of the artists' names, user interests, etc. The server uses story templates to render a narrative from the information stored in the knowledge base.

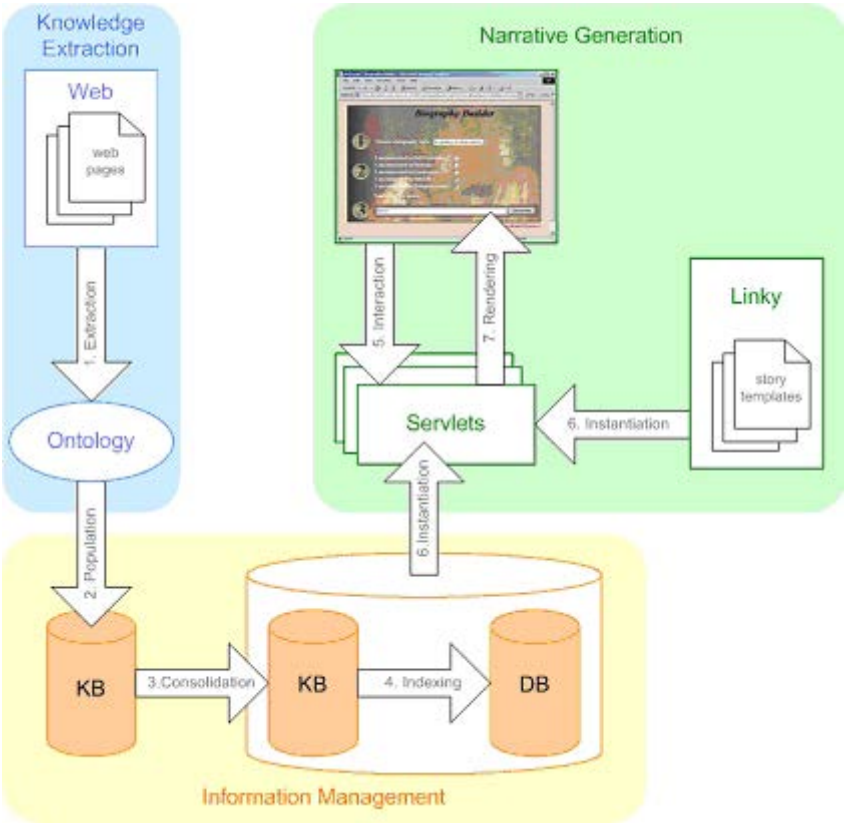


Figure 3.2: The Artequakt architecture

3.3 World News Finder

The World News Finder (WNF) [10] is a semantic search system designed for the news domain. It performs a search using automatically extracted metadata files instead of traditional keyword-based searching. The system collects daily news articles using RSS news feeds. These feeds are provided by news agencies. These articles are in HTML format which are parsed using HTML parser [11] resulting in plain text format. The content of the article is stripped of all the other irrelevant information present in the HTML page (advertisements, media content, presentation elements, etc.) before being passed to ANNIE (A Nearly-New Information Extraction System) pipeline. ANNIE is a GATE (General Architecture for Text Engineering) component that performs syntactical parsing, named entity recognition, and pattern matching using various sub-components, including an English tokenizer, an Onto gazetteer, a sentence splitter, a minipar parser, a pos tagger, a named entity (NE) transducer, an orthomatcher, and a JAPE transducer. Each plain-text article is processed by ANNIE, and the annotated articles are taken as the output. The annotated articles are then enriched with information from World News Ontology (WNO) and given to News Meta Tagger, which is another GATE component. The News Meta Tagger extracts useful metadata. This extraction is fully automatic and the results are stored in a file called a ”hat”. The metadata for each page consists of a set of topics. Three categories of annotations are produced in the metadata files which are

i) Topics: Annotations containing terms of the WNO

ii) Persons: Annotations describing persons found in the text

iii) Locations: Annotations with location information (eg. countries, cities, etc)

An example hat file is given below :

```
<?xml version=''1.0'' encoding=''UTF-8'' standalone=''no''?>
<EVENT>
<!DOCTYPE EVENT SYSTEM ``http://localhost/
    world_news.dtd''>
<URL>http://www.reuters.com/article/worldNews/
    idUSTRE5422D320090503?sp=true
</URL>
    <LOCATION>
        <CONTINENT>Middle East</CONTINENT>
        <COUNTRY>IRAQ</COUNTRY>
        <REGION-CITY>BAGHDAD</REGION-CITY>
    </LOCATION>
    <LOCATION_ABOUT>
        <CONTINENT>Americas</CONTINENT>
        <COUNTRY>U.S.A.</COUNTRY>
```

```
</LOCATION_ABOUT>
<DATE>20090503</DATE>
<TOPIC>
    <NAME>troops_withdrawal</NAME>
    <WEIGHT>5</WEIGHT>
    <ATTR>
        <TYPE>home_country</TYPE>
        <VALUE>U.S.A.</VALUE>
    </ATTR>
</TOPIC>
<TOPIC>
    <NAME>armed_forces</NAME>
    <WEIGHT>5</WEIGHT>
    <ATTR>
        <TYPE>home_country</TYPE>
        <VALUE>U.S.A.</VALUE>
    </ATTR>
</TOPIC>
<TOPIC>
    <NAME>war</NAME>
    <WEIGHT>3</WEIGHT>
</TOPIC>
<TOPIC>
    <NAME>security_measures</NAME>
    <WEIGHT>1</WEIGHT>
</TOPIC>
<TOPIC>
    <NAME>act_of_terror</NAME>
    <WEIGHT>1</WEIGHT>
</TOPIC>
</EVENT>
```

The user then performs a search of World News. The system's search algorithm exploits the metadata of each page and compares it to the user's query, which is expressed using terms from the ontology. Approximately 5000 articles were imported into the system's database for evaluation. To evaluate the system's performance, the authors posted 400 queries to the system, which fell into four categories based on the search terms: (1) articles referring to specific persons, (2) articles about a specified location, (3) articles containing one or more topics from the ontology, and (4) articles containing topics with specified attribute values. The search algorithm achieved 97.56% precision and 92.33% recall. This study highlights the effectiveness of ontology-based annotation techniques in improving search accuracy and the potential of such techniques in the field of information retrieval. Our work is based on a similar approach to this study, where we also utilized custom ontology-based metadata extraction techniques to analyze news articles.

3.4 Machine Learning Approaches

Several machine learning-based techniques are used to classify newspaper articles. Support Vector Machines (SVM) is a popular technique that involves locating a hyperplane that effectively separates data points into distinct categories [12]. Another technique is Naive Bayes, which calculates the probability of an article belonging to a particular category based on the occurrence of particular words or characteristics [13]. CNNs are deep learning models that have demonstrated effectiveness in categorizing newspapers by learning features from input data [14]. Recurrent Neural Networks (RNNs) are especially effective with sequential data, making them suitable for categorization tasks involving text [15]. Ensemble learning techniques such as Random Forests, which incorporate multiple decision trees, have also been utilized with promising results for newspaper categorization [16]. These examples are only a subset of the numerous machine-learning techniques used for categorizing newspapers. Numerous additional techniques and their variants have been developed and applied to this task, demonstrating the breadth and depth of approaches in this discipline.

3.5 Summary

Following a close analysis of the preceding sections, it is clear that the researcher has made significant contributions to our issue domain. However, it is worth mentioning that the majority of existing works center exclusively around a small range of categories. For example, the Neptuno project is based on an ontology that includes only a few categories, limiting the system's ability to handle diverse and dynamic inputs effectively. Similarly, the Artequakt project's own Artequakt ontology concentrates primarily on the domain of artists, limiting its applicability to other domains.

The World News Finder project, on the other hand, corresponds more closely with our research aims. The World News Finder system delivers greater flexibility and responsiveness by incorporating a broader range of categories. Furthermore, it overcomes the limits of classic machine-learning algorithms by allowing for the incorporation of domain-specific information and ontologies, improving the accuracy and effectiveness of the categorization process.

Chapter 4

Methodology

In this chapter, we present the methodology employed in this study to address the research objectives. The methodology encompasses the research design, data collection procedure, and the algorithms used to categorize news articles. The following steps outline our workflow:

i) *Construct an ontology* that will give a proper platform for categorizing news articles.

ii) *Select a data source* and follow a proper procedure of *collecting ontology relevant news articles* for our study. *Employ a pre-processing technique to the collected articles* to make it useful for the next steps. The data pre-processing techniques we utilized are *stemming* and *removing boilerplate* from the data.

iii) *Make a list of words and phrases* for each of the classes of the ontology. Each of these lists becomes part of our *gazetteer list*.

iv) *Load the pre-processed articles* in the *GATE framework* [17] (General Architecture for Text Engineering). Integrate the gazetteer list into our architecture and then feed the pre-processed articles into the pipeline of our architecture. This process gives an annotated article for each input article.

v) Develop an algorithm and apply it to the annotated articles for the categorization of the given input articles.

4.1 Ontology Construction

We created our ontology based on the IPTC'S NewsCodes Media Topic taxonomy [1]. The IPTC's NewsCodes Media Topic Taxonomy is a set of controlled vocabularies that is used to categorize news stories and media content. It was developed by the International Press Telecommunications Council (IPTC), a consortium of major news agencies, publishers, and

news industry vendors. The Media Topic Taxonomy includes more than 1,400 standardized codes, covering a wide range of topics related to news and media. These codes can be used to tag news stories and other media content, making it easier to search and categorize them. The codes in the Media Topic Taxonomy are organized into a hierarchical structure, with broad categories at the top and more specific subcategories below. We simplified the hierarchy to two levels for ease of use. The categories of our simplified taxonomy and their respective number of sub-categories are provided in Table 4.1. Additionally, a Wordcloud [18] of some selected categories is given in Figure 4.2.

The software we used to construct our ontology is called *Protégé* [19]. In the area of knowledge representation and ontology creation, Protégé is a well-known open-source software tool. Ontologies, which are formal representations of knowledge domains and their relationships, can be created, edited, and managed in the user-friendly environment of Protégé. In Protégé, we introduced each of the topics from the IPTC hierarchy level one as a class. Then we added the level-2 topics as subclasses to their corresponding level-1 classes. We call the subclasses which are under the same class sibling classes. In Protégé, the whole ontology falls under the *owl:Thing* class. An illustration of our ontology is given in Figure fig. 4.1.

Table 4.1: Categories of IPTC's NewsCodes Media Topic Taxonomy

Category	**Subcategory Count**
Arts, Culture, Entertainment and Media	3
Conflict, War and Peace	9
Crime, Law and Justice	5
Disaster, Accident and Emergency Incident	3
Economy, Business and Finance	4
Education	12
Environment	5
Health	9
Human Interest	8
Labour	7
Lifestyle and Leisure	3
Politics	9
Religion	8
Science and Technology	8
Society	11
Sport	12
Weather	4

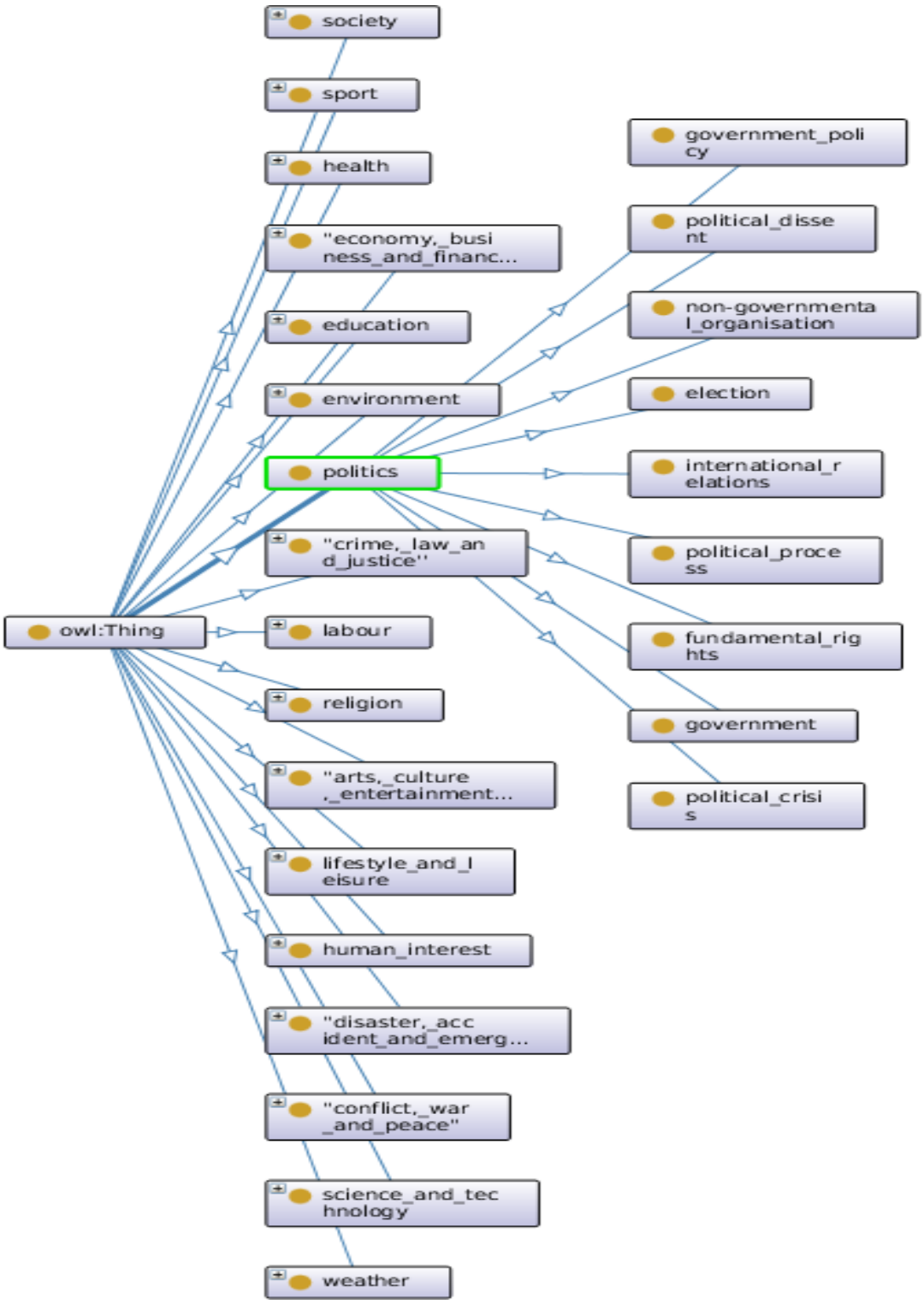


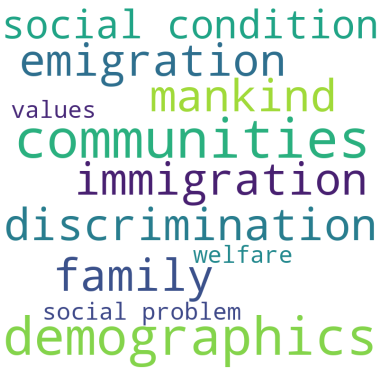
Figure 4.1: Illustration of our ontology in Protégé

In Figure fig. 4.1, we have extended the class politics and showed its subclasses namely, 'government policy', 'political dissent', 'non-governmental organization', 'election', 'international relations', 'political process', 'fundamental rights', 'government' and 'political crisis'. These are the nine subclasses of politics. The number of subclasses of politics is also verified in Table 4.1. Similarly, other classes can also be extended.

Category level 1 : Conflict, war and peace

Category level 1 : Health

Category level 1 : Politics



Category level 1 : Society

Figure 4.2: Word cloud of selected first-level categories

4.2 Data Collection and Processing

We chose a dataset based on the NELA-GT-2019 data source, classified with IPTC'S News-Codes Media Topic taxonomy for our experiment [1, 20].

The dataset originally has 10,917 articles. But some of them are too small for our study. Due to that reason, we selected 5051 news articles with at least 500 words. The overview of the dataset by category is provided in Table 4.2.

Table 4.2: The number of articles under each Level 1 Category

Category	**Articles Count**
Arts, Culture, Entertainment and Media	145
Conflict, War and Peace	325
Crime, Law and Justice	275
Disaster, Accident and Emergency Incident	150
Economy, Business and Finance	180
Education	379
Environment	344
Health	322
Human Interest	170
Labour	280
Lifestyle and Leisure	188
Politics	468
Religion	422
Science and Technology	487
Society	467
Sport	342
Weather	107

The description of the columns in the dataset are as follows:

i) id: Unique identifier of the article

ii) date: Date of article release

iii) source: News source of the article

iv) title: Title of the article

v) content: Content of the article

vi) author: Author of the article

vii) url: Link to the article

viii) published: Date of article publication in local time

ix) published_utc: Date of article publication in UTC time

x) collection_utc: Date of article written in UTC time

xi) category_level1: Level 1 category under IPTC taxonomy

xii) category_level2: Level 2 category under IPTC taxonomy

After acquiring the articles, we proceed with their processing to align with our research objectives. In the subsequent section, we elucidate the employed processing techniques used in our thesis, namely stemming and boilerplate removal.

4.2.1 Stemming

Stemming is a text preprocessing technique used in natural language processing and information retrieval to reduce words to their base or root form, known as a "stem." The goal of stemming is to remove affixes, such as prefixes and suffixes, from words so that related words with the same stem can be treated as a single entity.

For example, the words "running", "runs", and "ran" all have the same stem "run" after stemming. By reducing words to their stems, stemming helps to consolidate variations of words and reduce the dimensionality of the text data. Stemming algorithms employ heuristics and linguistic rules to identify and remove affixes from words. While stemming can be effective in some cases, it may not always produce a linguistically correct stem. It is a simplification technique that prioritizes efficiency over accuracy.

We used stemming because it helps to identify related words of a particular topic regardless of what form the words are in the article. We will discuss it further in section 4.4.

The stemming algorithm we utilized in our thesis is called Snowball stemming [21] also known as Porter2 stemming. We used the Natural language toolkit (NLTK) [22] for stemming. NLTK is an open-source library for natural language processing (NLP) [23] in Python. It provides a comprehensive set of tools and resources for tasks like tokenization, stemming, tagging, parsing, semantic reasoning, and more.

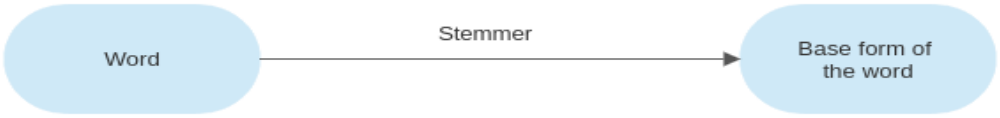


Figure 4.3: Stemming Process

4.2.2 Boilerplate Removal

Online news articles can have extra pieces of information which are irrelevant and unnecessary for our study. They are known as boilerplate contents. These irrelevant pieces of information can hamper the article categorization process. Luckily, almost all of our input articles are already stripped of all the boilerplate contents. We manually removed the irrelevant information from the rest of the few input articles. We identified the following sections as boilerplate and remove them from our input articles.

i) Copyright information

ii) Contact details or company information

iii) Generic introductory paragraphs

iv) Disclaimers or legal statements

v) Subscription prompts or advertisements

vi) Navigation menus or links

vii) Information about the author

4.3 System Architecture and Tooling Methodology

The system architecture serves as a guiding path toward a solution, outlining the sequential stages through which the data traverses. Prior to delving into the intricate details, it is important to address the software framework that encapsulates the entirety of the system architecture. Elaboration on this aspect is presented in the subsequent subsection 4.3.2.

4.3.1 GATE (General Architecture for Text Engineering)

In this part, we elucidate the principal instrument employed for the purpose of annotation. The annotation of textual content within an input article is essential in order to effectively classify the article. Our primary tool for this thesis is GATE (General Architecture for Text Engineering) [17]. *GATE* is an open-source software framework that provides a comprehensive and flexible environment for text processing and analysis.

GATE offers a wide range of tools and functionalities for various tasks in text engineering, including document parsing, information extraction, text annotation, semantic analysis, machine learning, and visualization. It supports multiple languages and is highly customizable, allowing users to adapt and extend its capabilities to suit their specific needs.

At the core of GATE is its architecture, which is based on a pipeline model. Users can create processing pipelines by chaining together different components, each responsible for a specific processing task. These components can be combined in a flexible manner to perform tasks such as tokenization, part-of-speech tagging, named entity recognition, syntactic parsing, and more.

One of the key strengths of GATE is its support for advanced annotation and semantic processing. It provides tools for creating and managing annotations on text, allowing users to define custom annotation schemas and mark up documents with domain-specific information. GATE also supports semantic analysis through the use of ontologies, enabling tasks such as entity linking, relationship extraction, and knowledge-based reasoning.

GATE has a graphical interface, known as the GATE Developer, which allows users to create and configure processing pipelines visually. It also provides a comprehensive set of APIs for programmatic access and integration with other systems.

There is a built-in pipeline application in GATE called *ANNIE (A Nearly-New Information Extraction System))* [24]. ANNIE consists of the following components:

i) Document Reset PR

ii) ANNIE English Tokeniser

iii) ANNIE Gazetteer

iv) ANNIE Sentence Splitter

v) ANNIE Parts of Speech (POS) Tagger

vi) ANNIE Name Entity (NE) Transducer

vii) ANNIE Orthomatcher

We employed a modified form of the ANNIE pipeline, so it is important to discuss the components we incorporated in our pipeline.

Document Reset PR: The Document Reset PR (Processing Resource) in the GATE ANNIE application is a component responsible for resetting the state of the document being processed. It acts as a starting point for subsequent processing stages within the ANNIE pipeline.

The Document Reset PR performs essential tasks such as clearing existing annotations, resetting internal counters, and initializing necessary variables. By doing so, it ensures a clean and consistent state for subsequent processing modules.

This PR is particularly useful when processing multiple documents consecutively, as it ensures that each document is treated independently without any carry-over effects from previous documents. It establishes a clean slate for subsequent analysis and extraction tasks, allowing the ANNIE pipeline to operate reliably and accurately on each document.

In summary, the Document Reset PR in the ANNIE application of GATE plays a critical role in maintaining the integrity and independence of document processing by resetting the state and preparing the document for subsequent stages of analysis and information extraction.

ANNIE English Tokeniser: The primary function of the ANNIE English Tokeniser is to perform tokenization, which involves segmenting text into individual tokens or words.

The English Tokeniser utilizes a set of rules and patterns specifically designed for English language tokenization. It takes as input a document or a text corpus and processes it to identify and extract individual words, punctuation marks, numbers, and other linguistic units that constitute the tokens in the text.

The tokenization process involves splitting the text based on spaces, punctuation marks, hyphens, and other delimiters. It also takes into account language-specific considerations, such as handling contractions, abbreviations, and special characters commonly found in English text.

By breaking down the text into tokens, the English Tokeniser lays the foundation for subsequent analysis and processing tasks in the ANNIE pipeline. These tokens serve as the basic units for tasks like part-of-speech tagging, named entity recognition, and syntactic parsing.

OntoGazetteer: OntoGazetteer is a powerful processing resource within the GATE framework. It is a part of the "Ontology Tools (8.6-SNAPSHOT)" plugin. It is designed to enhance text analysis and information extraction tasks by leveraging the knowledge encoded in an ontology.

At its core, OntoGazetteer utilizes a gazetteer, which is a list or dictionary of terms, to identify and annotate entities within the text. However, what sets OntoGazetteer apart is its ability to

go beyond simple term matching. It integrates with an ontology, which provides a structured representation of concepts and their relationships.

By incorporating an ontology, OntoGazetteer enables more sophisticated entity recognition and annotation. It leverages the ontology's hierarchical structure, taxonomies, and relationships to enhance the identification process. This means that not only exact matches are considered, but also related concepts and terms within the ontology.

The usage of an ontology allows OntoGazetteer to capture a broader range of entities and improve the accuracy and coverage of entity recognition. It can recognize variations, synonyms, and related terms associated with the concepts in the ontology. This makes it particularly valuable when dealing with complex domains or when the coverage of a traditional gazetteer may be limited.

OntoGazetteer can be customized and configured to work with specific ontologies, allowing users to adapt it to their specific domain or application needs. It provides a flexible and extensible solution for entity recognition and annotation, making it a valuable tool in various text analysis and information extraction tasks.

Now we will also briefly go through the excluded ANNIE components and talk about their functionalities.

ANNIE Gazetteer: The ANNIE Gazetteer is a component in the GATE text processing framework used to recognize and annotate named entities or specific terms in the text. It uses predefined lists of terms, called gazetteer lists, to identify matches in the input text and generate annotations for recognized entities. It offers flexibility for creating custom gazetteer lists and is effective for entity recognition tasks.

ANNIE Sentence Splitter: The ANNIE Sentence Splitter is a module in the GATE text processing framework used to segment input text into individual sentences. It employs various heuristics and language-specific rules to identify sentence boundaries based on punctuation, capitalization, and contextual cues. The Sentence Splitter is useful for tasks that require sentence-level analysis or processing, such as information extraction, sentiment analysis, and machine translation. It helps in organizing and structuring text data for further analysis and understanding.

ANNIE POS Tagger: ANNIE POS Tagger assigns grammatical labels or tags to each word in a given sentence, indicating its syntactic category, such as noun, verb, adjective, etc. The POS Tagger utilizes machine learning algorithms and statistical models trained on annotated corpora to predict the most likely POS tags for words based on their context and linguistic patterns. This information is valuable for various natural language processing tasks, including text classification, information extraction, and sentiment analysis. The POS Tagger helps in understanding the grammatical structure of sentences and enables more accurate analysis of text data.

ANNIE NE Transducer: The ANNIE NE Transducer identifies and extracts named entities, such as person names, organization names, locations, and other named entities, from text documents. The NE Transducer utilizes pattern-matching rules, gazetteers, and machine learning algorithms to recognize and classify named entities based on their context and linguistic patterns. It helps in extracting important information from the text and is useful in various applications like information extraction, entity linking, and knowledge graph construction. The NE Transducer enhances text understanding by identifying and categorizing named entities, enabling more advanced analysis and information retrieval from text data.

ANNIE Orthomatcher: The ANNIE Orthomatcher is used for orthographic matching in text. It helps identify and match orthographic variants of words, which are different spellings or representations of the same underlying concept. The Orthomatcher is particularly useful in tasks like entity normalization and disambiguation, where it can match different surface forms of an entity to a single canonical representation. By recognizing orthographic variations, the Orthomatcher improves the accuracy and consistency of text analysis and enhances the performance of downstream natural language processing tasks. It employs techniques like string similarity measures, pattern matching, and rule-based approaches to effectively match and link orthographically similar words or phrases in text data.

4.3.2 System Architecture and Workflow

The system architecture of our thesis serves as the blueprint for designing and implementing a robust and efficient solution to the categorization of news articles. The system architecture in our study is demonstrated in Figure 4.4

In Figure 4.4, Initially, the input news articles undergo stemming to extract their essential content, followed by the elimination of extraneous information such as boilerplate.

Subsequently, the processed articles are inputted into GATE, where a comprehensive corpus is constructed to encompass the entirety of the processed articles. This step is crucial in GATE's ANNIE pipeline for processing the articles. Since ANNIE, as part of the GATE framework, operates on a corpus rather than individual documents. So even if we have a single news article as input we still have to create a corpus and put the article in the corpus.

Figure 4.5 shows an article named "10728.txt_0001D" is within "corpus1".

Next, we loaded the ANNIE pipeline application on GATE. Among the ANNIE components we discussed earlier, We used the "Document Reset PR", "ANNIE English Tokeniser", and our Gazetteer List. We will delve into the specifics of this Gazetteer List in Section 4.4, providing a comprehensive analysis and in-depth exploration. Among the other ANNIE components, "ANNIE Sentence Splitter" is only necessary for the latter components. "ANNIE POS Tagger", "ANNIE NE Transducer", and "ANNIE Orthomatcher" are pre-built Machine Learning models.

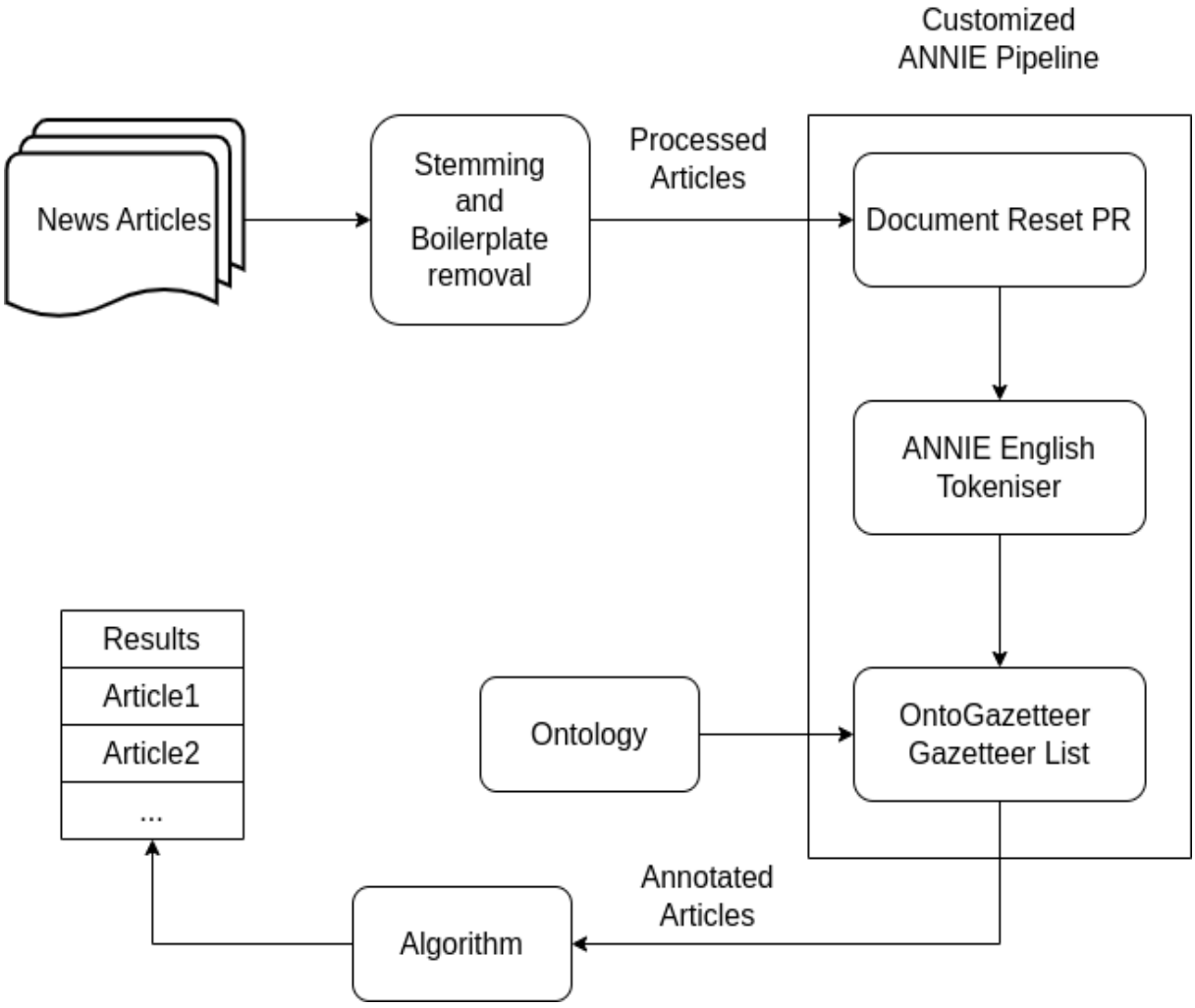


Figure 4.4: System Architecture Overview

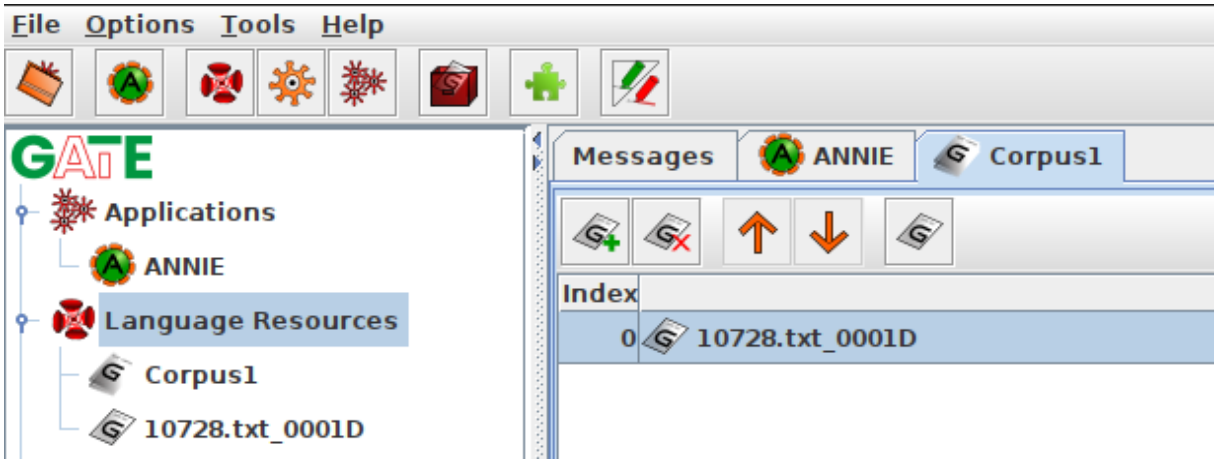


Figure 4.5: An article within a corpus

The reason for not using the other ANNIE components is that we wanted to solve the news article categorization problem solely depending on an ontology-based approach. We also did not include the "ANNIE Gazetteer" in our pipeline as we created our own Gazetteer List.

In order to incorporate our Gazetteer List into the GATE framework, we installed the Ontology Tools (8.6-SNAPSHOT) plugin. This comprehensive plugin serves as a valuable resource for working with ontologies within GATE Developer, offering a range of CREOLE resources including OntoGazetteer, GATE Ontology Editor, OAT, RAT-C, RAT-I, and GAZE. Among these resources, our focus lies on the OntoGazetteer processing resource, which serves as the container for our Gazetteer List.

To integrate the OntoGazetteer into our pipeline, we initiated the loading process by accessing the "lists.def" file and the "mapping.def" file. The "lists.def" file acts as a repository for all the file names contained within our Gazetteer List, providing a centralized reference for efficient management. Furthermore, it is essential to establish a seamless connection between our constructed ontology and the Gazetteer List within the OntoGazetteer. To achieve this, we leverage the "mapping.def" file to map each ontology class to its corresponding file, which contains the pertinent words and phrases associated with that particular class. Figure 4.6 demonstrates the introduction of OntoGazetter in the pipeline and the loading process of "lists.def" and "mapping.def".

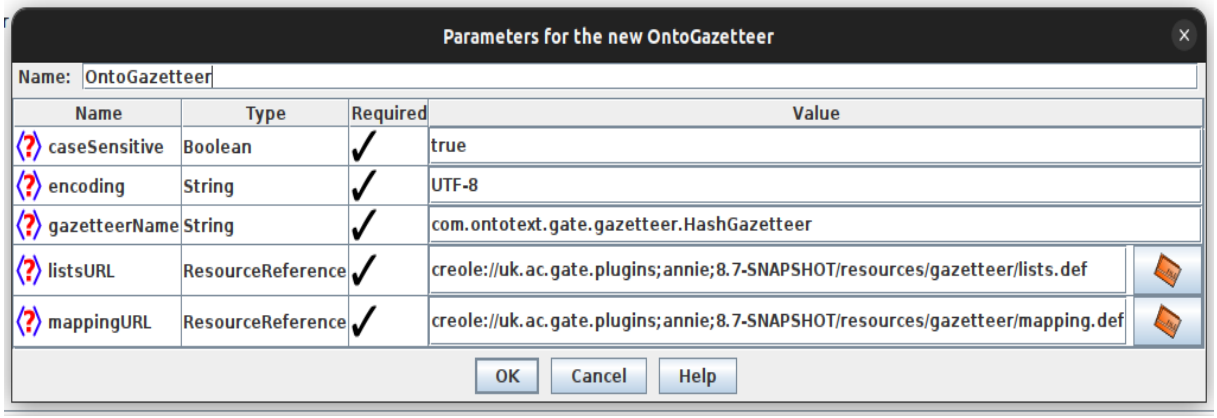


Figure 4.6: Loading the Ontogazetteer

As we introduced the OntoGazetteer we eliminated the ANNIE Gazetteer from the pipeline and placed it in the same position as the ANNIE Gazetteer. Figure 4.7 shows the customized ANNIE pipeline.

To ensure the successful functioning of OntoGazetteer, it is important to adhere to a prescribed format when listing file names in "lists.def". Similarly, proper formatting is also necessary for "mapping.def" to be recognized by the system. Merely including the file names without following the specified format will result in undesired outcomes. Below, we provide the respective formats for "lists.def" and "mapping.def".

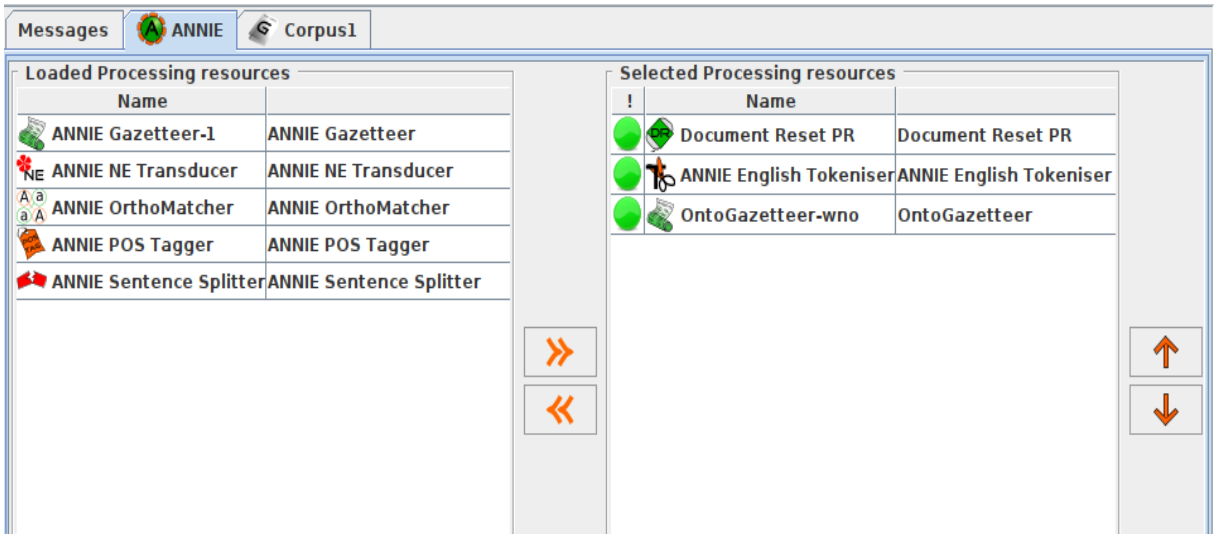


Figure 4.7: Customized ANNIE Pipeline

lists.def: ⟨path_to_the_file⟩/⟨filename⟩.lst:majortype:minortype

mapping.def: ⟨path_to_the_file⟩/⟨filename⟩.lst:file:////⟨path_to_the_ontology⟩/⟨ontology_name⟩.owl:⟨class_name⟩

Within the "lists.def" file, the inclusion of a majortype is obligatory, while the presence of a minortype is discretionary. The majortype signifies the broader category to which an annotated word belongs, whereas the minortype denotes a more specific subgroup to which the word pertains. For instance, in the case of an annotated city name, the corresponding minortype would indicate the country to which it belongs, while the majortype would signify the encompassing continent. In our scenario, we assigned the appropriate filenames to the majortype, as we did not have any further usage for it in our process.

Once the corpus and pipeline have been prepared, the next step involves executing the pipeline on the corpus. The articles within the corpus undergo a number of processing steps facilitated by each component of the pipeline. Firstly, the *Document Reset PR* eliminates any existing annotations from the articles, ensuring a clean starting point for subsequent processing. Following this, the *ANNIE English Tokeniser* partitions the entire article into individual words, facilitating further analysis. Subsequently, the OntoGazetteer component generates annotations for words that correspond to entries within the Gazetteer list, while also associating pertinent class information with each annotated word. The annotations that are created in the OntoGazetteer step fall under the "Lookup" markup. Here, a markup is a set of annotations that incorporates the annotations which follow a common rule. With the successful execution of the OntoGazetteer, the processing in the pipeline finishes. The annotated article showing the Lookup annotation type in GATE is depicted in Figure 4.8. It shows that a stemmed word called "academ" is part of the "school" class, and its annotated type is Lookup.

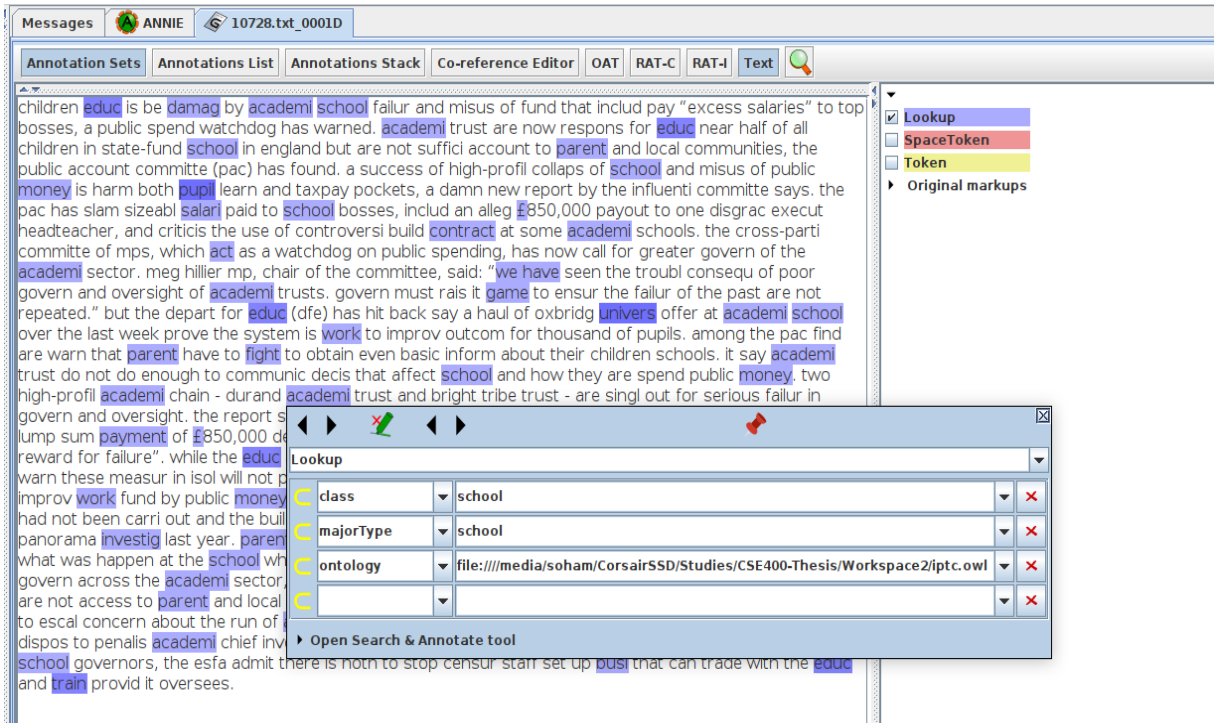


Figure 4.8: An annotated article for Lookup type in GATE

Upon the completion of the ANNIE execution, the resultant output comprises our meticulously annotated articles. It becomes imperative to preserve each annotated article either on a local storage device or any other designated repository. We save the annotated articles in *inline.xml* format. Subsequently, these annotated articles can be individually retrieved and processed through a specialized algorithm. This algorithm leverages the information encapsulated within the annotations to perform computations, ultimately outputting the category each article belongs to.

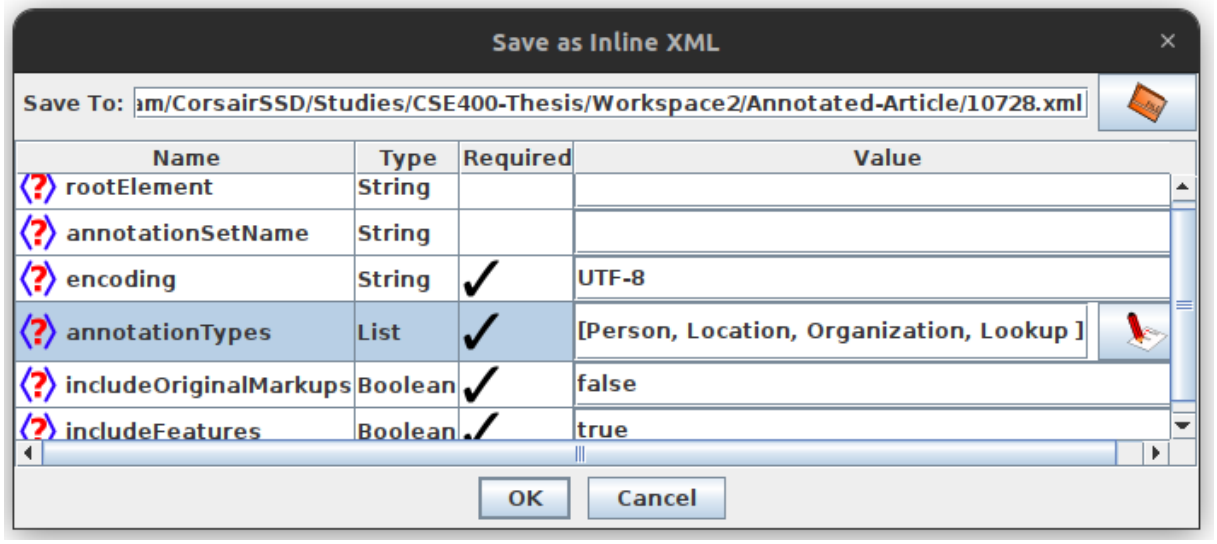


Figure 4.9: Saving an Annotated Article

Figure 4.9 shows the GATE window for saving an annotated article named “10728.xml”. Note

that the file saving window by default keeps the following type of annotations - “Person”, “Location”, and “Organization”. Since our sole focus was on the “Lookup” annotations, we had to manually add the “Lookup” type.

4.4 Gazetteer List Development

The Gazetteer List serves the purpose of detecting and annotating distinct named entities or predetermined terms within textual documents. To construct our Gazetteer List, we employed the assistance of WordNet [25] and word2vec-google-news-300 [26], we also searched for common words and phrases on the internet and manually selected the ones which are relevant. This comprehensive list encompasses text documents, denoted by the file extension ”lst.” For building the Gazetteer List we used both the original form of the words and phrases as well as the stemmed form since stemming is not always accurate. Below, we delve into the mechanisms and functionalities of these resources, shedding light on their operational principles and contributions to the construction of our Gazetteer List.

WordNet: *WordNet* is a lexical database and semantic network that groups words into sets of synonyms called ”synsets.” It provides a comprehensive and structured resource for exploring the meanings, relationships, and associations between words. WordNet captures the lexical and semantic relationships of words and offers information such as definitions, examples, and word usage.

The core concept in WordNet is the *synset*, which represents a collection of synonymous words that share a common meaning or sense. These synsets are interconnected through various relationships, such as *hypernyms* (superordinate terms), *hyponyms* (subordinate terms), *meronyms* (part-whole relationships), and antonyms (opposite terms). These relationships allow for the exploration of semantic connections and the identification of word similarities and differences.

In our study, we utilized *hypnonyms* or the subordinate terms and the *lemmas*. A *lemma*, in this context, refers to the base form or canonical representation of a word. It represents the core lexical form of a word without any inflections or grammatical variations. On the other hand, we found it more reasonable to use *hypnonyms* as they represent words or phrases that are more specific than the given words. We did not use *hypernyms* as we do not want to generalize the words. Because in that case, we would get overlapping words in multiple topics, which is something that we want to minimize. In Algorithm 1, we show the algorithm we implemented to find words from WordNet.

word2vec-google-news-300: Word2Vec is a popular word embedding technique used in natural language processing tasks. “word2vec-google-news-300” refers to a specific pre-trained Word2Vec model that has been trained on a large corpus of Google News articles. The “300” in

Algorithm 1 Generate words from WordNet

```
Load WordNet
Load the Ontology
Load Snowball Stemmer
for each class in the ontology do
    Open file class.name().lst
    synsets ← wordnet.synsets(class.name())
    for each synset in synsets do
        hyponyms ← synset.hyponyms()
        for each term in hyponyms do
            Write the term in the file
            stemform ← stemmer(term)
            Write the stemform in the file
        end for
        lemmas ← synset.lemmas()
        for each term in lemmas do
            Write the term in the file
            stemform ← stemmer(term)
            Write the stemform in the file
        end for
    end for
end for
```

the name represents the dimensionality of the word vectors, meaning each word in the model is represented by a vector of 300 numerical values.

This particular Word2Vec model is trained using a technique called continuous bag of words (CBOW) or skip-gram. It aims to capture semantic relationships between words by representing them as dense vectors in a high-dimensional space. The vectors are learned through a process of predicting the context of words or predicting words given their contexts within the training data.

The "word2vec-google-news-300" model is known for its wide coverage of words and phrases, as it has been trained on a large and diverse dataset. It can be utilized in various natural language processing tasks, such as word similarity, word analogy, text classification, and information retrieval, to enhance the understanding and processing of textual data.

While extracting related words and phrases we only considered the most similar 500 hundred words and among them, we ultimately accepted the words which had a semantic similarity value of at least 0.8 or 80%. In Algorithm 2 We demonstrate the algorithm we applied to extract words from word2vec.

Manual Selection: There are some classes in our ontology that are not single words, for example, *"conflict, war, and peace"*, *"crime, law, and justice"* etc. In such cases, the Gazetteer files of these classes or topics remain empty. Also, WordNet and word2vec-google-new-300

Algorithm 2 Generate words from word2vec

```
1: Load word2vec-google-news-300
2: Load stemmer
3: Load the ontology
4: for each class in ontology do
5:    Open file with the name class.name().lst
6:    similarWords ← word2vec.most_similar(class.name(), top=500)
7:    for word and similarity_value in similarWords do
8:       if similarity_value ≥ 0.8 then
9:          Write the word in the file
10:         stemmedword ← stemmer.stem(word)
11:         Write stemmedword in the file
12:      end if
13:   end for
14: end for
```

are not always accurate in detecting the relevant words and phrases of a given class name or topic. To address the first issue, we manually looked out for related words and phrases on the internet. We examined these related terms and selected the ones that are most frequently used in new articles. Then, to solve the second problem we had to manually remove some less closely related terms from the Gazetteer List.

4.5 Analysis of Annotated Output Files

In Section 4.3, we talked through the whole working process of our system. The outputs of our system are a list of annotated files corresponding to the input news articles. As we have seen in Figure 4.9 that GATE suggests us a number of annotation types, namely, "Person", "Organization", and "Location". Even though we do not get these annotations by processing through our pipeline. The annotation type we are only interested in is "Lookup".

We recall from Subsection 4.3.2 that we saved our annotated files in *inline.xml* format. In this format, we have the annotated words inside the ⟨Annotation_type⟩ and ⟨/Annotation_type⟩ tags. In our case, These are ⟨Lookup⟩ and ⟨/Lookup⟩. Below, we demonstrate the full annotation format of the annotated files.

⟨Lookup gate:gateId = "⟨ID⟩" majorType = "⟨majorType⟩" class = "⟨class_name⟩" onotology = "⟨path_to_ontology⟩/⟨ontology_name⟩"⟩

4.6 Algorithm Analysis

In the final section of this chapter, we will delve into our algorithm and illustrate its functionality when applied to the input articles. Algorithm 3 demonstrates the pseudo-code.

In our algorithm, we first load our ontology. We declare a mapping variable that will count the number of annotated words or phrases and map them with their corresponding classes. Next, we read an input annotated file. Lines 4 to 17 of Algorithm 3 extract and process the annotated article. Since we want to focus on the "$\langle Lookup \rangle$" tag, we begin the process by splitting the article using the delimiter "$\langle /Lookup \rangle$". This converts the article into tokens of strings called lines, containing "$\langle Lookup \rangle$". Then we start a loop for the tokens. For each token we determine the starting position of the substring "class="" to extract the class name of "$\langle Lookup \rangle$" annotation. In fact, there can be multiple "$\langle Lookup \rangle$" annotations for each token, hence multiple classes. After finding the class names we increase their value in a mapping variable called *"class_count"*.

Lines 18 to 33 handle the core computation for determining the resulting class or category of a given news article. We iterate through the ontology, and for each subclass or level-2 class of the ontology, we figure out its parent class and add its *class_count* value to its parent class's *class_count* and make the current subclass's *class_count* equal to zero. After the previous steps, we get all the number of occurrences of the level-1 classes.

We then calculate the percentage of the number of occurrences for each class. We determine the class and its percentage value that has the highest percentage of the number of occurrences. We set a threshold that is five percent less than the highest percentage. All classes with a percentage above or equal to the threshold are considered as the solution for categorizing the news article. We allow a threshold less than the maximum value because there are certain articles that place emphasis on topics other than their true category or main focus. So we consider all the categories that have a value close to the maximum value as our answer.

Algorithm 3 Determine the category of a news article

Load the ontology
$class\_count \leftarrow$ mapping$\langle className, count\rangle$
$fileReader \leftarrow$ Read the input annotated file
$lines \leftarrow$ split the content of the $fileReader$ by the delimiter "$</Lookup>$"
for each $line$ in $lines$ **do**
  if the substring "$<Lookup$" exists in $line$ **then**
    $indices \leftarrow$ extract the starting index of all instances of the substring "**class="**" from $line$
    for each $startidx$ in $indices$ **do**
      $tokens \leftarrow$ split $line[startidx + len(class = \text{“})]$ by the delimiter(**"**)
      $className \leftarrow$ tokens[0]
      if className does not exist in $class\_count$ **then**
        $class\_count[className] \leftarrow 0$
      end if
      increase $class\_count[className]$ by 1
    end for
  end if
end for
for each $class$ in ontology **do**
  if class.name() exists in $class\_count$ **then**
    $currentclass \leftarrow$ ontology[class.name()]
    $rootclass \leftarrow$ currentclass
    while rootclass.name() $\neq$ "owl.Thing" **do**
      $currentclass \leftarrow$ rootclass
      $rootclass \leftarrow$ ontology.parent(rootclass)
    end while
    if currentclass.name() does not exist in $class\_count$ **then**
      $class\_count[currentclass.name()] \leftarrow 0$
    end if
    $var \leftarrow$ class_count[currentclass.name()]
    $class\_count[currentclass.name()] \leftarrow$ var + class_count[class.name()]
    $class\_count.remove(class.name())$
  end if
end for
$sumval \leftarrow$ sum(class_count.values())
for each $className$ in $class\_count$ **do**
  $class\_count[className] \leftarrow$ class_count[className]/sumval $\times$ 100
end for
$maxval \leftarrow$ max(class_count.values())
$secondmax \leftarrow$ maxval - 5
$ans \leftarrow$ empty set
for each $className$ in $class\_count$ **do**
  if $class\_count[className] >= secondmax$ **then**
    $ans.add(className)$
  end if
end for
$print(ans)$

Chapter 5

Experimental Result and Analysis

In this chapter, we give a detailed presentation of our experimental result. In section 4.2 we provided details about our experimental data and preprocessing methods. The experimental outcomes are then analyzed in section 5.2

5.1 Evaluation Metrics

We used accuracy, precision, recall, and F_1-score as evaluation metrics.

A Contingency Table or **Confusion Matrix** consists of an $n \times n$ array where n is the number of classes in the data [27]. It is also known as **Error Matrix**. The reference data, or data that is known to be accurate or true, is always represented by the columns. The mapped classes that were produced from the remote sensing data are represented in the rows. We can construct a number of accuracy measures from our data using the confusion matrix. An example of the Confusion Matrix of our experiment is given below in Figure 5.1.

		Reference Data			
		Water	Forest	Urban	Total
Classified Data	Water	21	6	0	**27**
	Forest	5	31	1	**37**
	Urban	7	2	22	**31**
	Total	**33**	**39**	**23**	***95***

Figure 5.1: Example of Confusion Matrix.

We'll broadly discuss the evaluation metrics we used to measure the performance of our system. But first, consider the following terms described below.

True Positive (TP): The amount of positive examples or cases that a classification model correctly identifies or categorizes as positive is referred to as true positive. It determines how well the model can distinguish the positive class from other classes by reliably identifying positive cases.

True Negative (TN): True negative refers to a model's accurate classification or identification of negative situations. It shows how well the model can identify non-positive cases by accurately predicting and differentiating the negative class from other classes.

False Positive (FP): A false positive is when a model predicts a positive result when it should have predicted a negative outcome. It happens when something is incorrectly classified by the model as belonging to the positive class when it shouldn't.

False Negative (FN): A false negative occurs when a model incorrectly labels a positive case as negative. It indicates that when the model should have, it was unable to correctly recognize an instance as being a member of the positive class.

Now we will explain each of the evaluation metrics we employed to measure our system.

Accuracy: In classification tasks, accuracy is a frequently used metric to evaluate a classification model's overall correctness. It determines the proportion of correctly identified examples to all of the dataset's instances. In other words, it evaluates how accurately the class labels are predicted by the model.

Let's look at a binary classification problem where we need to categorize emails as spam or not spam in order to understand accuracy. Let's say we have 100 emails in our collection. Of these, 15 emails are accurately identified as spam (true positives), whereas 80 emails are correctly identified as not being spam (true negatives). However, the model incorrectly labels 2 spam emails as non-spam and 3 spam emails as spam (false positives and false negatives).

With these figures, we can use the following formula to determine the accuracy:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

In this example, the accuracy would be, $\frac{(80 + 15)}{(80 + 15 + 3 + 2)} = \frac{95}{100} = 0.95$ or 95%.

A greater accuracy score means that the model is correctly classifying its predictions, whereas a lower accuracy value reveals that misclassifications may exist. It's crucial to keep in mind that accuracy might not be sufficient to fully evaluate model performance, particularly when the dataset is unbalanced or the prices of certain errors vary. Therefore, for a more thorough assessment of the model's performance, it is frequently advantageous to take into account additional metrics like precision, recall, and F1-score in addition to accuracy.

Precision: Precision is a metric that assesses a classification model's accuracy, specifically in finding positive examples. It computes the percentage of accurately anticipated positive instances (true positives) out of all positive instances predicted (true positives + false positives). In layman's terms, precision measures how accurate the model is at identifying positive cases.

Let's consider a medical test for a specific ailment to demonstrate precision. Precision would be the percentage of correctly diagnosed positive cases (true positives) out of all positive instances discovered by the test (true positives + false positives). A high precision score indicates a low false positive rate, indicating that the model is dependable and accurate in identifying positive situations.

Precision is calculated as $\text{Precision} = \frac{\text{TP}}{\text{TP + FP}}$

Precision is especially crucial when the cost or impact of false positives is large. It focuses on the accuracy of positive predictions, specifically the fraction of accurately predicted positive cases among all positive instances projected.

Recall: Recall, also known as "sensitivity" or "true positive rate," measures a classification model's ability to capture all relevant instances of a certain class. It is concerned with the proportion of true positive forecasts versus the overall number of positive events.

To further grasp recall, let's consider a binary classification scenario in which we must decide whether or not an email is spam. Assume we have a dataset of 100 spam emails and our model correctly detects 80 of them as spam. The model, however, incorrectly identifies ten spam emails as non-spam.

The true positive (TP) in this instance refers to the 80 spam emails accurately recognized as spam by the model. The false negative (FN) relates to the ten spam emails that were incorrectly categorized as non-spam. Recall is computed by dividing TP by the total of TP and FN, in this case, 80 divided by (80 + 10), yielding a recall of 0.8889, or 88.89%.

$$\text{Recall} = \frac{\text{TP}}{\text{TP + FN}}$$

Recall is an important measure, particularly in cases where detecting all positive instances is critical, such as disease identification, fraud detection, or security concerns. A greater recall score indicates that the model is better able to properly capture positive events while minimizing false negatives.

F_1-score: The F_1-score is a metric that combines precision and recall to provide a balanced

estimate of the effectiveness of a classification model. It is especially useful when working with skewed datasets, where the number of positive and negative cases varies greatly.

To further comprehend the F_1-score, let us return to the example of email spam classification. Assume our model has an accuracy of 0.85 (85%) and a recall of 0.80 (80%). Precision is the proportion of correctly categorized positive occurrences among all positive instances predicted, whereas recall is the proportion of correctly classified positive instances among all actual positive instances.

The harmonic mean of precision and recall is used to get the F_1-score:

$$F_1\text{-score} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}.$$

In our case, by entering the precision and recall values into the calculation, we get:

$$\text{The } F_1\text{-score is equal to } 2 \times \frac{(0.85 * 0.80)}{(0.85 + 0.80)} = 0.827$$

The F1-score is a single statistic that weighs precision and recall. It is scaled from 0 to 1, with 1 representing great precision and recall and 0 representing poor performance.

The F_1-score comes in handy when we need to account for both false positives and false negatives. It enables us to evaluate the overall performance of a classification model, which is especially useful when achieving a good balance between precision and recall is critical.

5.2 Experimental Result

For our experiment, we already described in Section 4.2 that we had 5,051 articles from our dataset. But due to some unfortunate reasons, we were only able to test 425 randomly selected articles. Of the 425 articles, we have selected 25 articles for each category.

The first reason for not being able to test all the articles from the dataset is that we failed to automate certain tasks in GATE. For example, loading the preprocessed articles, running the ANNIE pipeline application, and finally saving the annotated file in storage. We had to do these steps manually and it was not that efficient. The second reason is that we found the dataset NELA-GT-2019 [20] only recently.

We created a table 5.1 that represents the testing result of 25 articles. We have also made a confusion matrix for the results we have achieved. The matrix is given in Figure 5.2

Table 5.1 illustrates that the accuracy of our system is 80%, which is not as good as the related works in the field. However, it is still a decent result. The table also shows that some classes perform very well and others are not that impressive. For example, of all the classes, the class "politics" performs very badly, whereas, the class 'sports' performs very well.

Table 5.1: Evaluation Metrics

Category	Precision	Recall	F1-Score	Support
arts, culture, entertainment and media	0.80	0.80	0.80	25
conflict, war and peace	0.81	0.88	0.85	25
crime, law and justice	0.90	0.72	0.80	25
disaster, accident and emergency incident	0.78	0.72	0.75	25
economy, business and finance	0.89	0.68	0.77	25
education	0.83	1.00	0.91	25
environment	0.85	0.92	0.88	25
health	0.78	1.00	0.88	25
human interest	0.74	0.68	0.71	25
labour	1.00	0.80	0.89	25
lifestyle and leisure	0.94	0.64	0.76	25
politics	0.51	0.72	0.60	25
religion	0.77	0.80	0.78	25
science and technology	1.00	0.80	0.89	25
society	0.63	0.88	0.73	25
sport	0.88	0.92	0.90	25
weather	0.90	0.72	0.80	25
Accuracy			0.80	425
Macro Avg	0.83	0.80	0.81	425
Weighted Avg	0.83	0.80	0.81	425

In the case of precision, some classes like “politics” and “society” perform badly. To dig into the reasons for its poor performance on precision, we must refer to the confusion matrix in Figure 5.2. The confusion matrix shows us that low precision is due to the classes having higher false positive (FP) values. That means quite a number of other classes were categorized as the mentioned classes or the classes with low precision. This is because there are some articles that have true labels or categories different from what they largely discuss. These kinds of articles have large discussions about content that should belong to other topics. So when our system sees these extensive discourses on other topics it automatically assumes that the articles must fall under the topics it is discussing.

Now, categories or classes like “politics” and “society” can be very common in other news articles with different categories. For example, it is not very uncommon to talk about politics while discussing “crime, law, and justice” and other topics like “economy, business, and finance”,

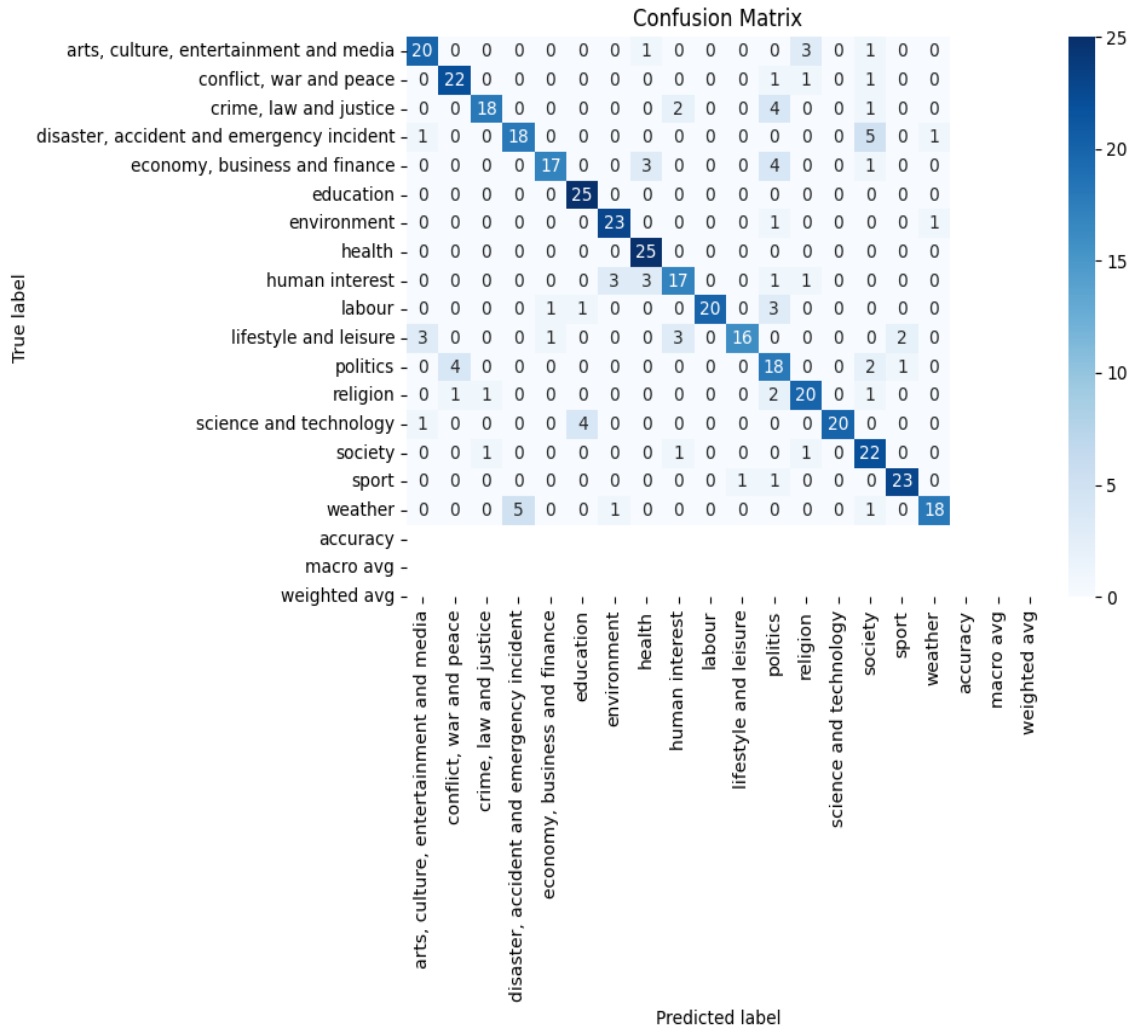


Figure 5.2: Confusion Matrix of our experiment

resulting in our system identifying such articles as “politics” instead of their true categories.

On the other hand, some articles have decent precision that is because their main contents do not extensively talk about other topics. That’s why the precision becomes high.

We again focus on Table 5.1. Similar to precision, some categories have bad recall values and others have average to good recalls. The fundamental cause for having a bad recall value is having too many false negatives, which is classifying a news article into another category instead of its true category. Table 5.1 demonstrates that classes such as “economy, business and finance”, “human interest”, “lifestyle and leisure” etc. have bad recalls. This is due to some articles from these categories or any category with low recall having too much content about other topics. For example, many articles on “economy, business, and finance” were identified as “politics” and “health”. There’s a high chance that an article from “economy, business, and finance” would talk more about political issues and how politics influences economic deals and agreements than its own content. It is highly likely that such an article would be categorized

as “politics” instead of “economy, business, and finance”, resulting in high false positives (FP), which ultimately, reduces the recall value.

Now if we talk about the F_1-scores of our experimental result in Table 5.1, we can undoubtedly say that the categories having good scores on both precision and recall eventually make up a good F_1-score, as F_1-score combines both precision and recall.

In summary, we can say that the result of our system on the performance metrics is overall decent, neither remarkable nor dreadful. However, it is difficult to declare anything about our system since we only tested 425 randomly selected articles out of 5,051 articles. This experiment serves only as a demonstration of our system’s performance, nothing more. So until we are able to test all the articles of the dataset it is difficult to come to any conclusion about our system.

Chapter 6

Conclusion and Future Work

In the final chapter of our thesis, we summarize our research and contribution, and then briefly go through the research objectives we achieved from this thesis. Subsequently, we discuss the future scope of our research.

6.1 Salient Points

In our thesis, we implemented a system that automatically categorizes input news articles based on industry-accepted IPTC's NewsCodes Media taxonomy. For that, we developed an ontology based on the IPTC taxonomy's level-1 and level-2 topics. Each of the classes of our ontology represents a level-1 or level-2 topic. We populated a Gazetteer List for our ontology which is a list of text documents. each of the text documents captures relevant words and phrases of the topic it represents. We selected 5,051 articles with their respective category from NELA-GT-2019 [20]. The selected articles consist of at least 500 words. We then removed all the unnecessary content from the articles and stemmed them afterward. We utilized a text processing software called GATE for further processing the input articles. In GATE, we used a customized ANNIE pipeline application that contains the following components Document Reset PR, ANNIE English tokenizer, and a processing resource called OntoGazetteer which combines both the Gazetteer List and the ontology. We executed the articles through the pipeline and finally get an annotated version of the articles. We ran an algorithm on the annotated articles that counts the occurrences of the annotated terms in the articles and that value to the corresponding class the terms belong to. Next, for each class, the algorithm adds the class values to the respective parent class. For each input article, the algorithm then finds the maximum value of all the classes and sets a threshold that is 5% than that value. The classes that surpass that threshold are the categories that an input article belongs to. In this way, our system automatically categorizes the topic of a news article.

We tested 425 articles in total and 25 articles from each class and got 80% accuracy overall. However, the result of the other metrics was somewhat average. With this, we achieved all of our thesis objectives, they are following: we developed an ontology based on the news industry's concepts and terminology, automated the extraction of metadata or the annotations from news articles using the ontology, and automatically categorized articles based on accepted concepts in the news industry.

In summary, our contribution lies in the development of a new system that encompasses a broader range of topics than the existing research works. We applied explicit domain knowledge through our ontology. Through our system, we will be able to help new industries, journalists, and editors to categorize news articles accurately and more specifically.

6.2 Future Research Direction

From our observations, we have figured out two potential directions: one for improving our work and another for expanding it to further enhance its utility.

Improvement: We could use all of the components of GATE's pipeline, this includes the ANNIE Gazetteer, ANNIE Sentence Splitter, ANNIE POS Tagger, ANNIE NE Transducer, and ANNIE Orthomatcher. We have already discussed these components in Subsection 4.3.1. Some of these components such as POS Tagger, ANNIE NE Transducer, and ANNIE Orthomatcher are pre-trained ML models. These components, as their names suggest, will perform parts of speech detection, named-entity recognition, and orthographic matching in text, respectively. This will give us a deeper insight into the input articles. The output of these components will provide additional information, which will then be integrated into new annotation types. Then, by analyzing these annotations, it is possible to discover additional relationships between annotated words. This analysis can offer valuable insights on which categories to avoid and which categories to prioritize, ultimately leading to an improved answer. The ANNIE pipeline for our future work is presented below.

i) Document Reset PR

ii) ANNIE English Tokeniser

iii) OntoGazetteer

iv) ANNIE Gazetteer

v) ANNIE Sentence Splitter

vi) ANNIE POS Tagger

vii) ANNIE NE Transducer

viii) ANNIE Orthomatcher

This model would become a hybrid model composed of ontology and machine learning. It is important to note that in the new system, we should avoid stemming the input articles in the preprocessing phase. Stemmed articles will provide a stemmed version of the words, in such a case components like ANNIE POS Tagger and NE Transducer will not be able to work. It is because stemmed words lose their grammatical significance, also ANNIE POS Tagger and NE Transducer are not trained for stemmed words. It is worth mentioning that we have the flexibility to apply this model to languages other than English. However, we should also mention that utilizing the ANNIE pipeline for a specific language still depends on the availability of ANNIE's support for that particular language. Then, we have to populate the ontology or the Gazetteer list with relevant words and phrases in that target language. In this way, we can provide newspaper categorization for other languages as well.

Search Engine Integration: We draw inspiration from the related research conducted by Kallipolitis et al. (2012) in the field of semantics. Our goal is to integrate our system with a search engine to enable users to retrieve all relevant articles from the world news domain based on their selected topic. Notably, our system offers an improvement by leveraging all the 17 level-1 categories derived from the IPTC's NewsCodes media taxonomy. This integration will result in a more refined and specific search engine, as it encompasses a broader range of topics.

References

[1] International Press Telecommunications Council (IPTC), "NewsCodes." https://iptc.org/standards/newscodes/, 2021. Accessed: May 9, 2023.

[2] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Scientific american*, vol. 284, no. 5, pp. 34–43, 2001.

[3] I. Horrocks, B. Parsia, P. Patel-Schneider, and J. Hendler, "Semantic web architecture: Stack or two towers?," in *PPSWR*, vol. 3703, pp. 37–41, Springer, 2005.

[4] "What is the semantic web?." https://www.ontotext.com/knowledgehub/fundamentals/what-is-the-semantic-web/, Accessed 2023.

[5] L. Ehrlinger and W. Wöß, "Towards a definition of knowledge graphs.," *SEMANTiCS (Posters, Demos, SuCCESS)*, vol. 48, no. 1-4, p. 2, 2016.

[6] IBM, "Knowledge graph." https://www.ibm.com/topics/knowledge-graph, 2021. Accessed: 2023-05-09.

[7] M. Mayank, "A lazy data science guide," 2021.

[8] P. Castells, F. Perdrix, E. Pulido, M. Rico, R. Benjamins, J. Contreras, and J. Lorés, "Neptuno: Semantic web technologies for a digital newspaper archive," in *The Semantic Web: Research and Applications: First European Semantic Web Symposium, ESWS 2004 Heraklion, Crete, Greece, May 10-12, 2004. Proceedings 1*, pp. 445–458, Springer, 2004.

[9] H. Alani, S. Kim, D. E. Millard, M. J. Weal, W. Hall, P. H. Lewis, and N. R. Shadbolt, "Automatic ontology-based knowledge extraction from web documents," *IEEE Intelligent Systems*, vol. 18, no. 1, pp. 14–21, 2003.

[10] L. Kallipolitis, V. Karpis, and I. Karali, "Semantic search in the world news domain using automatically extracted metadata files," *Knowledge-Based Systems*, vol. 27, pp. 38–50, 2012.

[11] "HtmlParser." https://htmlparser.sourceforge.net/. Accessed May 14, 2023.

[12] S. M. H. Dadgar, M. S. Araghi, and M. M. Farahani, "A novel text mining approach based on tf-idf and support vector machine for news classification," in *2016 IEEE International Conference on Engineering and Technology (ICETECH)*, pp. 112–116, IEEE, 2016.

[13] A. N. Chy, M. H. Seddiqui, and S. Das, "Bangla news classification using naive bayes classifier," in *16th Int'l Conf. Computer and Information Technology*, pp. 366–371, IEEE, 2014.

[14] D. Kim and M.-W. Koo, "Categorization of korean news articles based on convolutional neural network using doc2vec and word2vec," *Journal of KIISE*, vol. 44, no. 7, pp. 742–747, 2017.

[15] M. Ahmed, P. Chakraborty, and T. Choudhury, "Bangla document categorization using deep rnn model with attention mechanism," in *Cyber Intelligence and Information Retrieval: Proceedings of CIIR 2021*, pp. 137–147, Springer, 2022.

[16] Q. Wu, Y. Ye, H. Zhang, M. K. Ng, and S.-S. Ho, "Forestexter: an efficient random forest algorithm for imbalanced text categorization," *Knowledge-Based Systems*, vol. 67, pp. 105–116, 2014.

[17] H. Cunningham, "Gate: A framework and graphical development environment for robust nlp tools and applications," in *Proc. 40th annual meeting of the association for computational linguistics (ACL 2002)*, pp. 168–175, 2002.

[18] L. Oesper, D. Merico, R. Isserlin, and G. D. Bader, "Wordcloud: a cytoscape plugin to create a visual semantic summary of networks," *Source code for biology and medicine*, vol. 6, no. 1, p. 7, 2011.

[19] M. A. Musen, "The protégé project: A look back and a look forward," *AI Matters*, vol. 1, June 2015.

[20] A. Petukhova and N. Fachada, "Mn-ds: A multilabeled news dataset for news article hierarchical classification," *Data*, vol. 8, no. 5, p. 74, 2023.

[21] M. F. Porter, "Snowball: A language for stemming algorithms." Published online, October 2001. Accessed 11.03.2008, 15.00h.

[22] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[23] J. F. Allen, *Natural Language Processing*, p. 1218–1222. GBR: John Wiley and Sons Ltd., 2003.

[24] K. Bontcheva, V. Tablan, D. Maynard, and H. Cunningham, "Evolving gate to meet new challenges in language engineering," *Natural Language Engineering*, vol. 10, no. 3-4, pp. 349–373, 2004.

[25] G. A. Miller, "Wordnet: a lexical database for english," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[26] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[27] T. Olmsted, "Error matrix." Webpage, 2019. Accessed: May 10, 2023.

Appendix A

Ontology

A.1 Ontology used for our purpose

```
arts, culture, entertainment and media
|--- arts and entertainment
|--- culture
|--- mass media
conflict, war and peace
|--- act of terror
|--- armed conflict
|--- civil unrest
|--- coup d'etat
|--- cyber warfare
|--- massacre
|--- peace process
|--- post-war reconstruction
|--- prisoners of war
crime, law and justice
|--- crime
|--- judiciary
|--- justice
|--- law
|--- law enforcement
disaster, accident and emergency incident
|--- accident and emergency incident
|--- disaster
|--- emergency response
```

```
economy, business and finance
|--- business information
|--- products and services
|--- economy
|--- market and exchange
education
|--- curriculum
|--- educational grading
|--- educational testing and examinations
|--- entrance examination
|--- online and remote learning
|--- parents group
|--- religious education
|--- school
|--- social learning
|--- students
|--- teachers
|--- vocational education
environment
|--- climate change
|--- conservation
|--- environmental pollution
|--- natural resources
|--- nature
health
|--- disease and condition
|--- government health care
|--- health facility
|--- health insurance
|--- health organisation
|--- health treatment and procedure
|--- medical profession
|--- private health care
|--- public health
human interest
|--- anniversary
|--- award and prize
|--- birthday
|--- celebrity
```

|--- ceremony
|--- high society
|--- human mishap
|--- record and achievement
labour
|--- employment
|--- employment legislation
|--- labour market
|--- labour relations
|--- retirement
|--- unemployment
|--- unions
lifestyle and leisure
|--- leisure
|--- lifestyle
|--- wellness
politics
|--- election
|--- fundamental rights
|--- government
|--- government policy
|--- international relations
|--- non-governmental organisation
|--- political crisis
|--- political dissent
|--- political process
religion
|--- belief systems
|--- relations between religion and government
|--- religious conflict
|--- religious facility
|--- religious festival and holiday
|--- religious leader
|--- religious ritual
|--- religious text
science and technology
|--- biomedical science
|--- mathematics
|--- natural science

```
|--- scientific institution
|--- scientific research
|--- scientific standards
|--- social sciences
|--- technology and engineering
society
|--- communities
|--- demographics
|--- discrimination
|--- emigration
|--- family
|--- immigration
|--- mankind
|--- social condition
|--- social problem
|--- values
|--- welfare
sport
|--- competition discipline
|--- disciplinary action in sport
|--- drug use in sport
|--- sport achievement
|--- sport event
|--- sport industry
|--- sport organisation
|--- sport venue
|--- sports coaching
|--- sports management and ownership
|--- sports officiating
|--- sports transaction
weather
|--- weather forecast
|--- weather phenomena
|--- weather statistic
|--- weather warning
```

Appendix B

Codes

B.1 Step 3: Data Pre-processing

B.1.1 Stemming

```
# This file stems the contents of the given input file
import sys
from nltk.stem.snowball import SnowballStemmer

# Create an instance of the PorterStemmer
stemmer = SnowballStemmer("english")

if len(sys.argv) < 2:
    print("Usage: python3 stemming_input.py <input_file>")
    exit(1)

args = sys.argv[1:]

# Convert the arguments to a string
args_string = "".join(args)

with open(args_string, "r") as f:
    basefile = args_string.split("/")
    output_file = "Stemmed-Article/" + basefile[-1]
    text = f.read()

# Read the input file
```

```
23
24 # Split the text into individual words
25 words = text.split()
26
27 # Stem each word using the PorterStemmer
28 stemmed_words = [stemmer.stem(word) for word in words]
29
30 # Convert the stemmed words back to a single string
31 output_text = "␣".join(stemmed_words)
32
33 # Write the stemmed words to the output file
34 with open(output_file, "w") as file:
35     file.write(output_text)
```

B.2 Step 4: Ontology Population

We use this code to construct gazetter lists by compiling words and phrases for each class in the ontology. We utilized WordNet and word2vec-google-news-300 for this purpose.

B.2.1 Populating Gazetter lists using WordNet

```
1
2 import nltk
3 import os
4 from owlready2 import get_ontology
5 from nltk.corpus import wordnet
6 from nltk.stem.snowball import SnowballStemmer
7
8 nltk.download('wordnet')
9
10 onto = get_ontology("iptc.owl").load()
11 empty = False
12 nempty = 0
13 stemmer = SnowballStemmer("english")
14
15 for cls in onto.classes():
16     # Get the synsets of a word
17
```

```
18      className = cls.name
19      className = className.replace("/", "_")
20      file = open("OntoGazList/" + className + ".lst", "w")
21      term = className.replace("(", "")
22      term = term.replace(")", "")
23      term = term.replace(",", "")
24      synsets = wordnet.synsets(term)
25      # Iterate over the synsets
26      for synset in synsets:
27          # Get the hyponyms of a synset
28          hyponyms = synset.hyponyms()
29          for hypo in hyponyms:
30              word = hypo.name()
31              word = word.split(".")
32              word = word[0].replace("_", "␣")
33              # Stemming the word using SnowballStemmer
34              word = stemmer.stem(word)
35              file.write(word + "\n")
36          lemmas = synset.lemmas()
37          for l in lemmas:
38              word = l.name()
39              word = word.replace("_", "␣")
40              # Stemming the word using SnowballStemmer
41              word = stemmer.stem(word)
42              file.write(word + "\n")
43      file.close()
44      with open("OntoGazList/" + className + ".lst", 'r') as fp:
45          lines = len(fp.readlines())
46          if lines == 0:
47              nempty += 1
48  print(nempty)
```

B.2.2 Populating Gazetter lists using word2vec-google-news-300

```
1  import os
2  import gensim
3  import gensim.downloader as api
4  from nltk.stem.snowball import SnowballStemmer
5  from owlready2 import get_ontology
```

```
6
7 # Downloads the word2vec-google-news-300
8 # model = api.load('word2vec-google-news-300')
9 model = gensim.models.KeyedVectors.load_word2vec_format
10         ('''gensim-data/word2vec-google-news-300/GoogleNews-vectors-
11          negative300.bin''', binary=True)
12
13 onto = get_ontology('iptc.owl').load()
14
15 stemmer = SnowballStemmer("english")
16
17 for cls in onto.classes():
18     className = cls.name
19     className = className.replace("/", "_")
20     input_word = className.lower()
21     input_word = input_word.replace("(", "")
22     input_word = input_word.replace(")", "")
23     input_word = input_word.replace(",", "")
24     with open("OntoGazList/" + className + ".lst", 'a') as fp:
25         # Find the top 10 most similar words
26         try:
27             similar_words = model.most_similar(input_word, topn=500)
28             filtered_words = []
29             for word, similarity in similar_words:
30                 if similarity >= 0.8:
31                     filtered_words.append((word, similarity))
32             # Print the results
33             # print(f"Top 10 words related to '{}':")
34             for word, similarity in filtered_words:
35                 # print(f"\t{word} ({similarity:.2f})")
36                 word = word.replace("_", "␣")
37                 fp.write(stemmer.stem(word) + "\n")
38         except KeyError:
39             print(f'''The word "{input_word}"
40                    is not present in the vocabulary.''')
```

B.3 Step 5: Newspaper Annotation using GATE

B.3.1 *lists.def* file generation

```
# This program adds majortype in the "OntoGazList/lists.def" file
# lists.def stores the list of
# - gazetteer files and their corresponding major type
# major type is defined by the first term followed
# by a colon for each of the gazetteer list name in the lists.def file

from owlready2 import get_ontology
import re
import sys
import os

onto = get_ontology("iptc.owl").load()
file = open("OntoGazList/lists.def", 'w')
for cls in onto.classes():
    majortype = cls.name
    lstfile = majortype + ".lst"
    file.write(lstfile + ":" + majortype + "\n")
file.close()
```

B.3.2 *mapping.def* file generation

```
# It generates the mapping between
# - the OntoGazetteer list and the ontolgy classes.
# format:
# lstfile.lst:file:////<Path_to_the_owl_file>/iptc.owl:lstfile

from owlready2 import get_ontology
import re
import sys
import os

path = os.path.abspath("iptc.owl")
onto = get_ontology("iptc.owl").load()
filewrite = open("mapping.def", 'w')
for cls in onto.classes():
```

```
15     my_class = cls.name
16     lstfilename = my_class
17     lstfile = my_class + ".lst"
18     filewrite.write(lstfile + ":file:///" + path + ":"
19                     + lstfilename + "\n")
20 filewrite.close()
```

B.4 Step 6: Categorization Algorithm

We use this code to categorize the annotated articles based on the established ontology, enabling classification of input articles.

```
1  # result.py captures the core algorithm of our thesis.
2  # It runs the annotated files and generates results.
3
4  from owlready2 import get_ontology
5  import re
6  import sys
7
8  if len(sys.argv) < 2:
9      print('Usage: python3 result.py <annotated_file>')
10     exit(1)
11 with open(sys.argv[1], "r") as f:
12     content = f.read()
13
14 class_count = {}
15
16 lines = content.split("</Lookup>")
17 for line in lines:
18     if "<Lookup" in line:
19         indices = [match.start() for match in re.finditer(r'\bclass="\b',
20                                                           line)]
21         for class_start_index in indices:
22             class_name = line[class_start_index + 7:].split('"')[0]
23             if class_name not in class_count:
24                 class_count[class_name] = 0
25             class_count[class_name] += 1
26 with open('class_count.txt', 'w') as f:
27     for class_name, count in class_count.items():
```

```
28         f.write(f'{class_name}: {count}\n')
29 f.close()
30
31 onto = get_ontology("iptc.owl").load()
32
33 for cls in onto.classes():
34     if cls.name in class_count:
35         my_class = onto[cls.name]
36         root_class = my_class
37         while str(root_class) != "owl.Thing":
38             my_class = root_class
39             root_class = root_class.is_a[0]
40         if my_class.name not in class_count:
41             class_count[my_class.name] = 0
42         class_count[my_class.name] += class_count.pop(cls.name)
43
44 sumvalue = sum(class_count.values())
45 # Find the percentage of each class
46 class_count = {k: v / sumvalue * 100 for k, v in class_count.items()}
47 maxvalue = max(class_count.values())
48
49 # We set a threshold of 5% less than the max value
50 # taking all the classes that have a value greater than the threshold
51 secondmax = maxvalue - 5
52 maxvaluekeys = [k for k, v in class_count.items() if v >= secondmax]
53 for item in maxvaluekeys:
54     print(item)
```

Generated using Undegraduate Thesis LaTeX Template, Version 2.2. Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh.

This thesis was generated on Saturday 20th May, 2023 at 5:24pm.