Analysis of a Data Processing Pipeline for Generating Knowledge Graphs from Unstructured Data

Paras Kumar

Analysis of a Data Processing Pipeline for Generating Knowledge Graphs from Unstructured Data

Master's Thesis in Computer Science
EIT Digital - Cloud Computing & Services

Distributed Systems Group
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

Paras Kumar

21st August 2020

Author
Paras Kumar

Title
Analysis of a Data Processing Pipeline for Generating Knowledge Graphs from Unstructured Data

MSc presentation
28th August 2020

Graduation Committee

Prof. dr. ir. D. H. J. Epema	Delft University of Technology
Prof. dr. M.M. Specht	Delft University of Technology
Dr. Gregor Lämmel	Market Logic Software AG

Abstract

With the rapid growth of unstructured data across different mediums, it exposes new challenges for its analysis. To overcome this, data processing pipelines are designed with the help of different tools and technologies for the analysis of data at different stages. One of the applications which we find useful for our company is the creation of knowledge graphs for better representation and understanding of relations in the data. Knowledge graph is a structure of representing information where nodes represent the entities and edges define the relationships among them. The construction of a knowledge graph is a process of extracting meaningful information of entities and relations from unstructured textual data and storing it in a graph database. In this project, we are using Neo4j as a graph database for the efficient storage of data in the form of nodes and relations.

To achieve this goal, our first research question proposes the architecture and implementation of a data processing pipeline for the construction of knowledge graphs using unstructured textual data. There are three major stages involved in our pipeline and each component is implemented in a microservice architecture. The first stage starts with the parsing of textual documents in two different formats which are PDF and PPT. In the second stage, we are applying natural language processing techniques for the extraction of meaningful information out of this raw text. In the final stage, key pieces of data are stored into a graph database(Neo4j) for the construction of knowledge graphs. We are running our pipeline on a local machine for evaluating the performance and results of each component.

The core aspect of retrieving insights from this unstructured data is achieved with the use of natural language processing. In order to investigate more on this component, our second research question examines the cloud based natural language processing services from three renowned providers which are Amazon, Google and IBM. For choosing a suitable service among them, we evaluate their performance on a common data set of category Marketing from wikipedia. Based on our experimental analysis, IBM stands out among them from the perspective of the quality of output, execution time, features and cost. The adoption of a cloud based service not only leads to a faster development of business solutions but also reduces the engineering effort, its cost and maintenance of our custom implementation only with a little cost per our usage.

Preface

I am writing this thesis for the completion of my master degree under EIT Digital Master School in the track of Cloud Computing and Services. Based on the requirement of industrial thesis, I got an amazing opportunity to work on the industry project with Market Logic Software AG in Berlin, Germany. Market Logic provides a market insights platform which helps the world potential brands to produce and grow based on insights. This work was accomplished together with the collaboration of academia and industry. For this company, it was the first ever experience of doing a master thesis with a student and I hope that I laid the foundation for the future students at this place.

The company works with a huge amount of textual data gathered via many sources as it's the only fuel for generating useful and meaningful insights. As a part of the data science team, I worked on a new research based project which helps to extract meaning from unstructured textual data in the form of knowledge graphs. The entire ecosystem of the data processing pipeline requires a in depth study of many new tools and technologies which I never explored before in academic life. This project was a great mix of theoretical and practical tasks and I can proudly say that over the period of the last 6 months, I was able to gain knowledge of new technologies and frameworks which are widely used in the modern software development practices. I am optimistic that the analysis and result of this project will assist Market Logic in choosing the right tools and cloud based services for this pipeline.

This contribution of work would not be possible without the tremendous support and guidance of my company supervisor Dr. Gregor Lammel at every stage. I am very thankful to my university supervisor at TU Delft, Prof.dr.ir. D.H.J. Epema, for his permission and believing in me to work with this company. With the help of his great feedback and comments, I have improved my thought process and the quality of work.

Paras Kumar

Delft, The Netherlands
21st August 2020

Contents

Chapter 1

Introduction

In the early days, companies were struggling with the collection of data for analysis and decision making but now everyone is concerned with the drawing out of meaningful information from this enormous data. The emergence of the internet has provoked access to a large number of content creators to generate information. Users are producing content in different formats such as text, images and videos using various mediums such as pdf, blogs, YouTube etc. There is a massive amount of information available and accessible to everyone on the web. In order to generate useful bits of knowledge, we need a productive method to speak to this information. One such technique is to represent information via knowledge graphs. In brief, a knowledge graph is a graphical representation of a network of interconnected data defining key relations between different types of entities. However, the representation of the knowledge graph from this raw data requires critical measures and it exposes many new challenges from storage, processing, computation to representation.

In the construction of a knowledge graph, different actions are needed to be performed on each step. This process starts from the extraction of data from various sources such as Wikipedia articles, pdf's stored on a cloud storage or physical disk to the generation of semantics using natural language processing libraries and in the end signify it to the knowledge graph. By integrating existing tools at various steps the robust application can be developed and deployed to help businesses understand their values in a better way.

There are several challenges encountered in the execution and generation of the efficient knowledge graph by the Market Logic Software, the company where I'm writing this thesis. Market Logic Software offers an online portal full of data insights and intelligence which facilitates businesses to understand about the ongoing marketing trends and learn from historical data to enrich their customer base in the respective potential market. As part of my thesis internship, I'm currently doing research on the analysis of the data processing pipeline for the generation of know-

ledge graphs. A notable part of this research encompasses the study of existing tools and libraries to deal with each step of the processing pipeline. There are several tools, libraries and services which enabled us to interpolate this information in an effective manner. At Market Logic, we leverage the power of historical data to gather deeper insights about future trends.

1.1 Problem Statement

As per the title of the thesis project, there are three major steps in the pipeline i.e. extraction and processing of unstructured data from various data sources, transformation of raw floating textual data into a unified format to be processed by any natural language processing library for recognizing entities and relations which later on will be converted into a knowledge graph using a graph database.

The core facet of this pipeline heavily relies on the natural language processing component and it requires the right choice of library among various open source libraries. With the fast development of cloud technologies, there are a number of cloud based natural language processing services already offered by different cloud providers such as Amazon Comprehend, Google Cloud Natural Language API and IBM Natural Language Understanding service. As a part of this research, we will analyze and compare the ease of integration, cost, performance and features provided by these leading cloud based natural language processing services.

From the design and implementation perspective another challenge will be the execution of microservices and storage required for the data processing pipeline. The interdependence between services should be minimum such that it does not affect the agility of the design. The steps in the data processing pipeline are tightly coupled as output of one process will act as an input to the other process.

The research topic can be summarized into two following research questions:

1. **RQ1:** How to design and implement a data processing pipeline for generating knowledge graphs?

2. **RQ2:** How to choose the most suitable cloud based natural language processing service based on cost, performance analysis along with features and flexibility of individual service with other cloud providers?

This ongoing research project in Market Logic will be based on a microservice architecture, which runs different tasks of the pipeline and will be deployed on google cloud platform for running the system in the production environment. We are working on this project to discover more available options at each step of the data processing pipeline. As part of the study, different tools and technologies will be studied for the design and development of the pipeline. As a part of the first research question we will propose a design, implementation and architecture for the data processing pipeline. We will run experiments on a sample dataset for the evaluation of the services running in the pipeline. One of the second research

questions is to compare existing cloud based natural language processing services and analyze them based on different perspectives of cost, features, ease of use and the performance.

1.2 Research Approach

The key concept behind this research area is to turn the most of unstructured data into useful information in corresponding domains. The preferable method of research is mingled with study of existing tools, libraries and leveraging their features towards the design, implementation and experimentation of the data processing pipeline. For the very first research question, various state of the art techniques of pipeline will be studied and we will design our own custom architecture using existing frameworks and libraries to get the end result of a knowledge graph from unstructured raw data. Furthermore, we will run experiments on a sample data set to test the performance of each component of the data processing pipeline.

Natural language processing is an essential component of the pipeline, as it will discover the useful and meaningful information from the raw text for the construction of knowledge graphs. Natural language processing reflects the context of the content which will help businesses to understand market values. In order to speed up this natural language processing without major engineering effort, we take the advantage of cloud based natural language processing services. There are several cloud based natural language processing services are available but choosing one will require further research and experimentation. Our second research question will deeply analyze the features, performance, cost etc of such services from famous cloud providers such as Amazon, Google and IBM.

1.3 Thesis Structure and Contributions

The thesis consists mainly of five chapters. The contribution of the following next chapters is summarized as follows.

In Chapter 2, we first introduce the core concepts, terminologies and technologies at different stages of the data processing pipeline used in the research topic. As a part of the background chapter, we will also introduce the basic underlying components involved in the data processing pipeline. In Chapter 3, we will highlight existing approaches related to this domain problem, which will be done with the help of thorough literature review. In Chapter 4, we will propose the architectural design and implementation of a data processing pipeline for construction of knowledge graphs from unstructured data. As a second part of this chapter, we will run an experiment on each service based on sample data and evaluate their outputs. This chapter will describe and evaluate RQ1. In Chapter 5, we will analyze the cloud based natural language processing services and get the measures on RQ2. In addition, the key takeaways about experimental results on each cloud based service

will be discussed in this chapter. Finally, Chapter 6 will conclude the research findings and analysis of our pipeline based on different tools and libraries, summary of the thesis work and possible improvements and extensible features as a future work.

Our main contributions are as follows.

- We describe an overview of the existing scientific literature in the corresponding domain for processing of unstructured data into a knowledge graph.
- We propose the design and architecture of a data processing pipeline for the construction of knowledge graphs using different components and libraries. We implement and evaluate results of each component of the pipeline.
- We study the cloud based natural language processing services and evaluate them based on the cost, performance, feature analysis etc. This will help in selecting the suitable cloud based natural language processing service, which will reduce further engineering effort for the organization.

Chapter 2

Background and Concepts

In this chapter, we introduce the main concepts and elements relevant to this thesis. In Section 2.1, we describe the high-level abstract view of the data processing pipeline. Following Section 2.2, it defines the core operations involved in the data processing pipeline. In Section 2.3, we explain different formats of data available to be processed. In Section 2.4, we investigate the concept of graphs and a graph database. In the last Section 2.5, we introduce the concept of natural language processing which is useful for contextual understanding of data and helping us in construction of feasible input for the knowledge graphs.

2.1 Data Processing Pipeline

In computer science, the notion of pipeline brings up the orderly execution of various tasks and instructions by the processor in an organized manner. Pipelines are designed in order to break a bunch of processing steps into different components. These individual components in the pipeline can be easily developed, managed and monitored independently. A pipeline also known as a data processing pipeline, is a set of data processing components connected in a series, where the output of one component is propagated as the input of the next one in the line. Based on the requirement of the application, the connected components in the pipeline can be executed in parallel or in time-sliced (round robin) fashion [23].

The architecture of data processing pipelines allows us to capture and route data in such a way that it can be used for gaining meaningful information and later transforming it into reporting, analysis and using data easier. One out of many advantages of using a data pipeline is that they reduce the human annotated work and data noise of providing information, which is not relevant to the end goal of business decision making [10].

In our project, the data processing pipeline components for the creation of knowledge graphs are highly dependent on each other. We can integrate different tools

to connect different cloud services, software applications and data sources to create robust and resilient data pipelines. In general, a data pipeline improves the speed of our development by offering an easy to use abstract methodology for working with batch and streaming data in our applications. Mainly, there are three core stages of a data processing pipeline, which are defined in the next section [7].

2.2 Data Extraction, Transformation and Loading

The short term ETL for Extract, Transform and Load is widely used in data processing pipelines and also became a popular concept in the research area of data warehousing. It is the generic approach of retrieving data from numerous sources (that are not optimized for analytics) into a target system [14]. With the growing speed of data in every domain, ETL is a very crucial fraction of business intelligence processes. The data in bulk can be analyzed programmatically in one place for the discovery of hidden patterns and useful insights. The task of ETL is scheduled hourly, daily, weekly or monthly for enriching the data in data warehouses.

In practice, the ETL process is agile and it keeps changing with the passage of time as per business requirements. The new cloud based software as a service application offers powerful analytics warehouses like Amazon Redshift and Google BigQuery that have significant power to perform ETL operations in place rather than a need for a special staging area. Today, data is analyzed repeatedly in raw form rather than preloaded transactional summaries in databases. This has granted the development of delicate, adaptable and transparent ETL systems. Usually in the ETL process, data is extracted from databases, online transactions but in our project we are extracting data from PDF documents and wikipedia articles.

The three main operations of ETL are briefly explained below.

2.2.1 Extraction

The first step of the ETL process begins with the extraction of raw data from different homogeneous or heterogeneous sources. The application should be aware of the logical data map before data is extracted and loaded physically into the system. The data map defines the correlation between sources and target data. After extraction, data should be validated before it moves to the next stage of transformation. It is a very crucial aspect of the pipeline as the correct extraction defines the level of success of subsequent processes. Data extraction is commonly done by using one of the three following methods [1].

The method is chosen based on the source system, performance and business needs.

1. **Full Extraction:** In some systems, it is hard to track the changes being made to the existing data. For such system reloading of the entire data is the only possible way to get data out of the system. In this extraction technique,

we need to maintain a copy of the last extraction so that new records can be identified. It is used when data needs to be extracted for the first time and also useful for small tables or a last resort, as it demands a high rate of data volume transfer. Full extraction also reveals the current amount of data present in the source system [12].

2. **Incremental Extraction:** In this method, modifications in the source data can be identified since the last successful extraction. Only these data changes will be retrieved and loaded. We can detect these changes from the source data that has the latest timestamp. Apart from this, a new table can be created to keep track of changes in case data is fetched from any database. One disadvantage of incremental approach is that it may not be possible to detect deleted records in source data. The other alternative for incremental changes is to extract whole data and then perform the difference operation with the current data and last extraction. And this alternative will highly affect the performance of the system [12].

3. **Update Notification:** This approach is the most convenient as the application is notified for extraction of data whenever a change has occurred in the source system. It is useful when the initial data is extracted using the full extraction and then apply update notification for further change of data. The system which has a high frequency of updates will overwhelm the application with too many update notifications [12].

2.2.2 Transformation

Data extracted from various sources can have different variants. In this stage, anomalies in the data are perceived and data is transformed into a valid schema [12]. The generalized pattern of data is defined based on the underlying database storage schema and needs of the business requirements (A process of converting flat files into tables). Transformation also refers to the filtering, cleaning and mapping of data before any analysis can be carried out in next stages. Data transformation techniques enable you to understand more in depth about your business, customers, market trends and competitors and make it more accessible to everyone. This is the crucial step in data processing pipelines which add a certain value and amend the data such that interesting insights can be revealed. Data which does not need any transformation is called direct move [12].

Some basic operations of transformation are discussed below [12].

1. **Cleaning:** Removal of null values or replacing null with zero, conversion of generic terms to defined letters such as size of items (Large to L, Medium to M, Small to S, extra small to XS, extra large to XL etc) and unifying of different date formats into a standard format.

2. **Deduplication:** Removing duplicate tuples in the raw data which creates a hassle of handling redundancy and an overhead of storage.

3. **Format Revision:** Raw data comes from different sources which means that it has measurement of the same article in different units. Such as time in hours can be mapped to days, distance in kilometers can be converted to meters etc.

4. **Key Restructuring:** Relationships across different tables can be defined as well.

Few advance operations of transformation are discussed below.

1. **Derivation:** Derive meaningful information from existing sources of data by means of calculations, statistics etc. For example, create a metric of loss per item from total loss for a particular category of items, deriving revenue without taxes, etc.

2. **Filtering:** Raw data contains too many data points that may not be relevant and will be dropped in this stage. In other words, only select useful data.

3. **Joining:** Same kind of data from different sources can be linked together and it can be easily differentiated by introducing a source tag. such as revenue of ads from google ads, Facebook ads, etc.

4. **Splitting:** It is more feasible to divide a large chunk of data into multiple columns.

5. **Data Validation:** On the basis of some user defined rules, several data items are rejected or accepted for further analysis. Such as any sentence of length greater than 20 words is rejected, a row with only one column of null value is accepted, etc.

6. **Summarization:** Data is summarized to obtain full figures which are dispersed at various levels. It can be occurences of a word in a document, summing all the purchases a customer has made to create a total purchase, etc.

7. **Aggregation:** Different data elements across various data sources can be combined together for the generation of useful insights and the improvement of business recommendations, which couldn't be discovered with one source.

8. **Integration:** Based on the particular chunk of data having standard name and definition. We can perform customized operations to reconcile among different data names and values for the same data element.

2.2.3 Loading

This is the last step of ETL data processing pipelines. This stage relies on your application oriented goals intended to bring in with the help of transformed data into the data warehouse. In our project, we are using transformed textual data as an

input for a natural language processing module for further processing and semantic analysis. This process is highly dependent on the system of what you're loading data. In the world of big data, huge volume is dumped into databases in a couple of iterations (relatively in short shifts - when the system is ideally idle) and it should be optimized for better performance [12]. Failure is the norm in such systems and it is inevitable. Hence fault tolerance mechanisms should be configured to ensure data integrity in case of failures. As per prevailing performance of the system, loads can be cancelled or resumed later [12].

There are two main techniques of loading data.

1. **Full Load:** It takes place when the data is loaded very first time into the data warehouse and populates all tables in case of relational or any other format in case of non relational databases [13].

2. **Incremental Load:** In the data processing pipelines, extraction and transformation of data is an ongoing process which runs after every interval of scheduled time period. Hence it causes load operation to occur at regular intervals. Only a delta of source data is dumped in the warehouse and it is calculated using timestamp. Incremental load is mainly of two forms and it depends on the amount of data to be dumped [13].
   Streaming Incremental Load: It is useful for loading small volumes of data.
   Batch Incremental Load: It is useful for loading huge volumes of data.

In short, ETL (Extract, Transform, Load) takes raw data, extracts the information required for analysis, transforms it into a format that can serve business needs, and loads into a data warehouse. Data sources are diverse and we usually increase it with the period of time in order to learn more. Overall the best practice is not to clean entire data as it takes too long and business might not be willing to pay the cost of waiting. Loading becomes complex when you start loading it incrementally due to various factors [13]. Such as the data type of a column in the warehouse is string and it started receiving integers etc. All these changes can be deadly for data warehouses and eventually data will become in an inconsistent state. Hence ETL typically summarizes data to reduce its size and improve performance for certain types of analysis.

2.3 Structured, Semi Structured and Unstructured Data

There are three categories of data that can be analyzed and stored in the world of big data and analytics.

1. **Structured Data:** This is the most organized format of data, which can be easily programmed for searching and analyzing. It refers to the data stored in the predefined fields in the form of rows and columns in relational databases. The properties of structured data makes it straightforward to store and

analyze. We can manage structured data using structured query language known as "SQL" developed by IBM in the 1970's for relational databases. Relational keys are used to define relations between different types of data and can easily be mapped into existing fields. It is the most processed format of data in the development and simplest way to manage information [11].

2. **Semi Structured Data:** This is a partially structured format of data also considered as a mix of structure and unstructured data. It has some consistent properties but does not ensure a rigid structure as found in relational databases. This type of data requires a processing before stored in the relational database. An example of such a data format is xml, it has some meaningful tags and headers defined in the whole document which makes it recognizable but still a lot of floating text [11].

3. **Unstructured Data:** This is the most widely available format of data in the world. Unstructured data cannot be stored directly in the relational database and does not fit into any data model. It is hard to analyze and parse this data as it comes in several variants [11]. The absence of proper structure in this data makes it more hard to analyze, search and manage. Small to medium sized companies cannot afford the cost of processing this complex nature of data and hence it is often discarded. The modern techniques of machine learning and artificial intelligence algorithms made it easier to process this data. In our project, we are retrieving and processing unstructured text from wikipedia articles and PDF's for the generation of knowledge graphs.

2.4 Graphs

A graph is a better representation of real world objects and technology works well with the representation of reality. In the case of graphs the reality is about connections between different things. Formally, graph is a collection of vertices and edges or a set of nodes which defines the discrete object and each of which has some set of relationships that connect them together [34]. Graph represents entities in a way that these are related to the world as relationships. Data in the graph is usually expressed in the form of entities and relations that are centered in a particular domain.

There is a huge value in linking information that can be derived from graphs and it allows us easy traversal of links to discover distant parts of a particular domain model relating to each other. A graph is one of the most flexible formal data structures, so you can easily map other data formats to graphs using generic tools and pipelines.

2.4.1 Knowledge Graph

The embedding of data as a knowledge graph was introduced by Google in 2012 for enhancing the search results of a user's query and helping users in discovering the information quickly and easily. It's a mechanism of converting a huge amount of information into knowledge which is easily accessible. One of the research studies has defined a knowledge graph in a way that "A knowledge graph acquires and integrates information into an ontology and applies a reasoner to derive new knowledge." [37]. Knowledge graph is also considered as a large network of entities, their types, properties and relationship among them. Relations are first class citizens of the graph data model. From the perspective of programming, it's a method to model a knowledge domain with the help of subject matter experts and data interlinking using different algorithms.

The semantical meaning of the data can be found in the graph data in the form of ontology, which is the primary basis of the knowledge graph. We can run queries on the graph data in a similar way to natural language but using certain vocabulary constraints and rules. Precisely, we can show data in the form of entity and relation names related in a particular context. The numerous connections between data in a knowledge graph assists in improving the search efficiency, discovery of new trends and bridging the communication gap between businesses and customers.

The graph computation techniques can also be applied on the knowledge graphs for various applications such as shortest path, network analysis, etc. This makes the knowledge graph more intelligent over the stored data. The nature of schema is quite different from SQL as in contrast it is flexible in structure and easy to extend over the time. Knowledge graphs have the ability to retrieve miscellaneous metadata such as origin or versioning information which make it powerful for dealing with dynamic dataset. There are many potential applications of knowledge graphs which includes semantic search, finding fraud rings, risk and impact analysis, social network, logistic and routing, content based recommendation and knowledge management system. At market logic software, we are trying to create a rich knowledge graph for better understanding about behaviour, desire and need of the customers for our clients in different domains.

Roughly we can define four steps for the creation of a knowledge graph.

- Identify use cases for knowledge graphs in a particular domain.
- Collecting and organizing relevant data.
- Mapping of relationships across the data is defined.
- Keep adding knowledge to the data using any graph database.

The below figure 2.1 represents an example of a knowledge graph with several nodes and relationships. The nodes of graph are representing title of the movies,

actors participated in the movie with different roles in each movie which is further represented as a relation between node of movie and actor in the form of edge. From this graph it is evident that few actors played multiple roles in few movies and each movie has more than one actor.

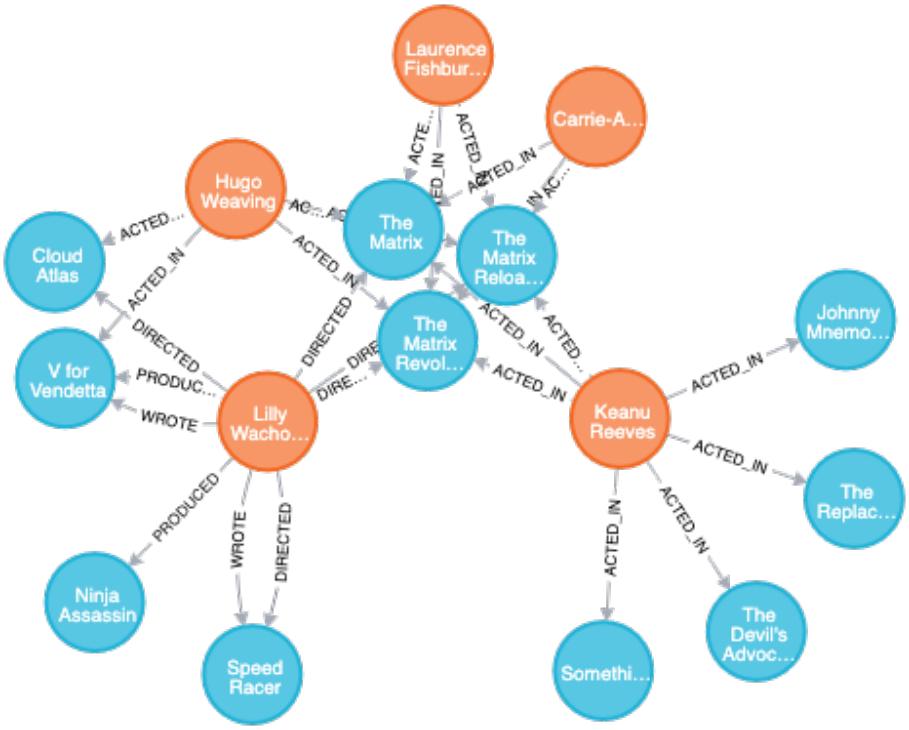

Figure 2.1: A sample knowledge graph of a movie dataset.

2.4.2 Graph Databases

Databases are classical systems used to store information in different formats. Graph database is a database that uses graph structure for semantic queries with nodes, edges and properties to store and represent data. Graph database is designed to treat the relationships between data as equally important to the data itself [29]. The graph database queries data efficiently because it stores, processes and embraces relationships natively. Graph database outperforms at managing connected data and complex queries irrespective of the size of the data set. Traditional SQL databases compute relationships at query time by applying expensive JOIN operations through different tables but on the other hand graph database stores connections alongside the data in the model [29]. Accessing nodes and relationships in a native graph database is an efficient, constant-time operation and allows you to quickly traverse millions of connections per second per core [29]. In our project, we are using Neo4j, one of the most popular graph databases available.

2.4.3 Neo4j

Neo4j [29] is an open source, NoSQL, graph database that provides an ACID (Atomicity, Consistency, Isolation, Durability) compliant transactional support for our application with native graph storage and processing of data. It implements the property graph model down to the level of storage, meaning that data is stored the way we imagine in a graph world and the database uses pointers to navigate and traverse over this graph. In contrast to graph processing or in-memory libraries, Neo4j also provides full database characteristics, including cluster support, and fault tolerance which makes it desirable to use graphs for data in production [29]. Neo4j provides a declarative query language known as Cypher which is similar to SQL but highly optimized for graph databases. Neo4j has constant time traversals along the depth and breadth of graphs due to efficient representation of nodes and relationships. It can be easily scalable to billions of nodes on commodity hardware [29]. The graph schema is quite flexible as it readily adopts changes over time, addition of new relationships and gauge up the business data as per requirement and agility. Neo4j is widely used by many companies and organizations due to its above-mentioned properties, community support and performance.

2.5 Natural Language Processing

Natural language processing(NLP) gives the machine an ability to read, understand and derive information from human languages [20]. It is concerned with the understanding of conversations between computers and human language. There is a huge volume of unstructured data in the form of text that needs to be processed and analyzed by the computer programs just like we do for other types of data. Text analytics is a type of natural language processing that turns text into data for analysis. Many companies are using text analytics to drive better customer experiences, business needs and improve daily life. Unstructured text data comes from almost every domain such as social media newsfeed, public posts, reviews and feedback for various applications, health records, research papers in many areas and much more. All hidden insights can be revealed in word streams with the help of natural language processing. Natural language processing is helpful for reconciling ambiguity in language and adding a structure to the data for many applications, such as speech recognition or text analytics [20].

In our project, a natural language processing module is embedded in the pipeline for the recognition of entities (subject, object, predicate) and relationships required for the construction of a knowledge graph. We are not exploring the natural language processing techniques mechanics in depth but we are leveraging this method for analysis purposes. There are many popular open source libraries of natural language processing available, which provides extraction of meaningful information from the raw text. With the rapid development in the field of cloud computing, cloud providers started offering natural language processing services which will

take away the complexity of this task from the user and will assist them to focus on the core aspect of the application. For answering our second research question, we will study cloud based natural language processing services provided by different cloud providers. More advanced research is in progress in the domain of natural language processing that will make machines more smarter in recognizing human language.

There are two main techniques used to fulfil natural language processing tasks, Syntactic analysis and Semantic analysis [25].

1. **Syntax:** It describes the structure and arrangement of words in the sentence. The syntactical analysis outlines how well the grammatical rules are enforced in the sentences. Different algorithms are used to apply these rules and extract meaning from them. It takes the charge of metadata information in the sentence [25]. Various syntactical techniques can be applied such as parsing of a sentence based on the grammar rules, applying boundaries on the sentences, identifying the part of speech among sentences. It requires the process of lemmatization that is reducing various inflected words into one form for easy analysis and cutting the inflected word to their root form which is known as stemming [25].

2. **Semantic:** It is one of the difficult tasks in natural language processing. It provides the meaning of the text in a certain context. The output of a natural language processing module is highly dependent on this method and this cannot be resolved completely. Human language is not easy to understand when it comes to specifically in various contexts. Different algorithms can be used for the understanding of words and structure of sentences in different paradigms. It implies the extraction of the parts of the text that can be categorized and identified as places, persons etc known as named entity recognition. Contextual ambiguity is also detached and specifies the appropriate meaning based on the context. With the help of an external database it improves the semantic intentions of words and converts it to human language, known as natural language generation [25].

Chapter 3

Literature Analysis of Knowledge Graph Systems

In this chapter we introduce the existing approaches and techniques of building the knowledge graph pipelines. In Section 3.1, we introduce the applications of knowledge graphs and related research approaches for handling these challenges. In Section 3.2, we discuss a Dstlr platform of creating a knowledge graph using existing tools and frameworks. In Section 3.3, we highlight an application of a knowledge graph and how it can deal with the question answer based applications using a knowledge graph. It creates a semantic based graph of keywords for relating the existing knowledge with new questions. In Section 3.4, we present an architecture of extracting knowledge and building a knowledge base of a specific domain. In Section 3.5, we explain another approach of creating a knowledge graph based on enrichment at each step of the pipeline. In the last section, we summarized our analysis based on this literature.

3.1 Introduction of Knowledge Graph Applications

Knowledge graphs are widely used in many applications and creating a rich knowledge graph from unstructured data is a very challenging task. In the past researchers have made many contributions to this emerging field with the growing amount of data. The Semantic Knowledge Graph has numerous applications, including automatically building ontologies, identification of trending topics over time, concepts related to failure scenarios from free text, data cleansing, document summarization, semantic search interpretation and expansion of queries, recommendation systems, etc [38]. A research defined that construction of a knowledge graph is a three step process which includes knowledge extraction, Entity mapping and data integration [39]. The area of knowledge extraction from unstructured text is a never ending process that learns to read the web [39]. One of the approaches is to extract only triples from sentences by using syntactic and lexical patterns [39]. This technique works well for extraction of triples from unstructured text but still misses the

mapping of entities. This leads to the result of ambiguous entity extractions [39].

There is also a lot of work done on relation extraction, linking of entities and related technologies but a scalable platform that performs end to end knowledge graph construction is not available [36]. Another research we found which worked in a similar direction as our project but with a little different perspective.

The extraction of context and discovery of knowledge from the textual data involves the role of natural language processing. The challenge is not only to extract the meaningful information but to store it in an efficient graph based data store to represent it in the format of graph [40]. The graph based structure helps in the linking of information among each other and in particular we can run our custom queries to generate desired insights.

3.2 A scalable Knowledge Graph Platform

Dstlr is an open source project for scalable, end-to-end knowledge graph construction from unstructured text [15]. It takes a collection of documents and extracts mentions and relations to create a raw knowledge graph, links mentions to entities in wikidata and then enriches the knowledge graph with facts from wikidata [15]. This project proposed the construction of a knowledge graph by integrating four existing tools which are Apache Solr, Apache Spark, Stanford CoreNLP and Neo4j. It supported the entire data management life cycle of documents. It is using wikidata for the purpose of verification of mentions extracted from the raw text. In this context, mention and entity are treated differently. There are four assumptions which are as follows.

1. Each document consists of zero or more mentions.
2. Each document may have zero or more relations between mentions. Mentions can occur in any number of relations.
3. Each mention in the document has zero or exactly one link to an entity in the external knowledge graph of wikidata.
4. Entities are linked in a random number of facts in the external knowledge graph.

In the below Figure 3.1, it shows the flow of data across different stages of the Dstlr platform. In the extraction phase, textual documents are filtered into a raw knowledge graph which is enriched with the facts from an external knowledge graph [36]. Spark is used for execution in a scalable manner and Neo4j is responsible for storing data in the form of nodes and relationships. Applications built on top of this platform leverage the power of declarative cypher query language for

the manipulation of the data in the knowledge graph [36].

Apache Solr is used to store all documents to be processed instead of holding it in the file system which will be significantly slower. The integration of document store provided Dstlr search capability of doing analysis on the subset of documents. Anserini is a toolkit which connects Apache Solr and dump the textual documents into it.

The execution layer of this platform heavily relies on the Apache Spark and it performs two major steps of knowledge graph construction, extraction and enrichment [36]. It will populate the raw knowledge graph with mentions, entities and relations discovered from the unstructured text. This research also finds the Stanford CoreNLP is the most useful toolkit for the analysis of natural language. Dstlr is using Spark to scale out the tasks of Stanford CoreNLP and process large document collections in a scalable fashion [36].

In the enrichment phase, it extracts entities from the external knowledge graph which is wikidata that occurs in the unstructured text and then enriches its own knowledge graph [36]. The external knowledge graph facts are used as a support of high quality end results. Spark stores each entity, relations and its mapping in the RDD (Resilient Distributed Datasets). The execution of the enrichment process is coordinated by Spark via the manipulation of DataFrames. For each entity in the corpus, it produces a row in a Spark DataFrame containing the entity URI, the relation type, and its value. These are then bulk-loaded into Neo4j [36].

The fact verification is done by matching the high quality external source such as wikidata, and it concluded that with reasonable certainty the information found in the source document is valid. There are some ambiguities in cases that further require human validation where source data does not match in wikidata, which is a huge collection of all entities. As per this research, existing knowledge graphs are typically constructed through a combination of different processes, ranging from manual entry to semi-automated techniques [36].

3.3 Community Question Answering System Using Knowledge Graphs

Traditionally, a simple user query returns a lot of search results that are more or less relevant. This takes a lot of time for the user to review many of them and filter the most suitable answer. The community based question answer forums such as Stack Overflow [27], Quora [24], yahoo answers [17], etc generate useful knowledge by the experts of relevant domains. A research is published to examine the content of existing questions in order to answer a new question based on the

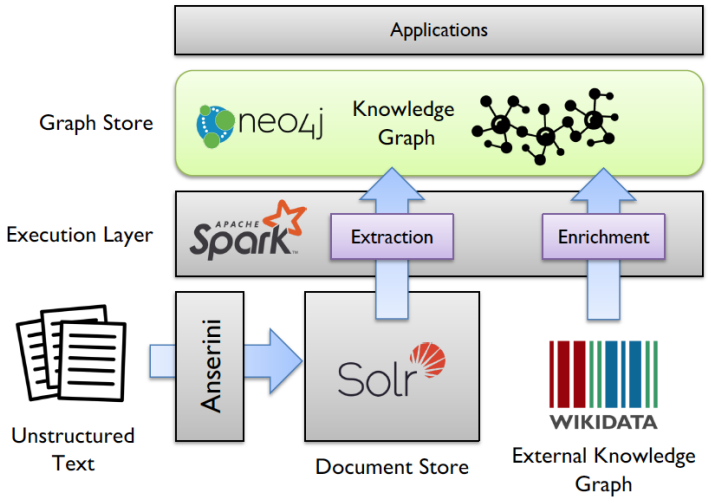

Figure 3.1: The architecture diagram of Dstlr platform [36].

knowledge graph structure by linking new questions to the graph of keywords [35].

It used a semantic graph based approach as questions and answers are very short in length hence the statistical methods are not useful here. The frequency of keywords is not enough to capture the information of the question hence this makes it difficult to match existing questions with new one as the text is of very short length. For the better understanding of limited input text it also takes the help of existing knowledge bases such as wikipedia, DBpedia, etc to build their own semantic graph of a particular domain. It uses the DBpedia knowledge base for the linking of entities extracted from current information and these are validated from wikipedia knowledge base. The end result is a semantic graph of keywords [35].

The construction of a knowledge graph begins with the input from an expert of domain with relevant titles and subtitles of the subject. There are three types of main nodes in the graph which represent relation from high level concept to specific details of keywords of a particular subject. The first node is the main title of the subject, then titles and subtitles nodes are added in the given hierarchy. Then, to each title, it adds the previously validated term sets of the title preprocessing step, as "keyword nodes" [35]. In the below Figure 3.2 it shows the architecture diagram of the graph construction process for this system. This is not a fully automated approach for the construction of a semantic knowledge graph of keywords for a particular domain. It uses data extracted from DBpedia knowledge base into a specific domain for representation of knowledge [35]. It has three major steps for the construction of a semantic graph of keywords.

Preprocessing

The input to this pipeline is provided by the expert of the domain of a specific subject. It contains main titles, chapter titles and subtitles. The system removes the stop words and generates different combinations of keywords and the teacher will choose the most meaningful terms [35]. It also defines multiple rules to avoid the duplicate keywords and merge them into one node of a graph. Initially the graph is in the form of tree structure as the first node represents the main title of the subject and on top of that it adds further titles and subtitles nodes in the given hierarchy. In each title, it adds previously validated term sets of the title preprocessing step as keyword nodes [35].

Extraction

In this phase, it extracts keywords from DBpedia which are found in the titles. For each node in the graph, it searches DBpedia pages with titles and labels similar to the node using SPARQL queries. In these pages there are wikilinks which mention this page inside them as well and it shows the inter connection between two pages which led to the idea of two concepts are semantically related to some degree [35]. They are extracting such pages using SPARQL queries and forming a list of candidate keywords which is used in the next stage. The result can be ambiguous as different phrases have different meanings based on the theme of the article such as IP stands for Internet protocol, Intellectual property, Industrial policies etc. It filters these pages by comparing the categories to the titles of its input and selects the most likely one.

Validation

After performing the linking of initial nodes with keywords of DBpedia it validates the relevance using the information from Wikipedia pages. Some of the results extracted from DBpedia might not be relevant to the content of the subject. Thus, it filtered the candidate keywords in order to remove irrelevant terms and it is done using a weight function which considers few parameters in the relevance validation phase. In the end , the graph is optimized and organized by removing duplicate keywords [35].

3.4 Domain Targeted Knowledge Extraction

One of the interesting research we found is about knowledge extraction based on a domain to grow its knowledge base. It is fully focused on the construction of a high precision knowledge base only containing (subject, predicate, object) sentences about the world for supporting the question answers of some application

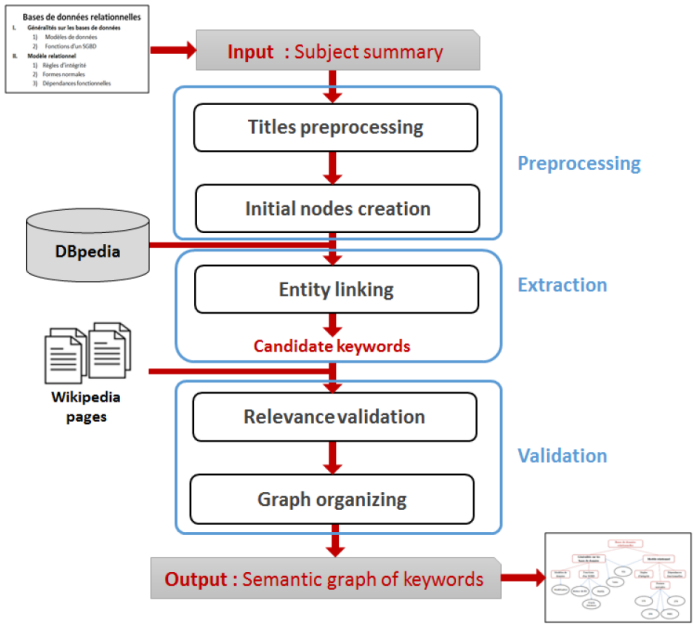

Figure 3.2: A semantic based graph construction pipeline [35] .

[41]. There are existing knowledge bases which are rich in terms of named entities but they are limited in general knowledge about common nouns. The existing resources also provide limited information about a domain of interest and constructed using a small set of relations [41].

To overcome these challenges, this research proposed a pipeline to produce domain specific high precision knowledge extraction by using Open IE (information extraction), crowd sourcing and a canonical schema learning algorithm that produces the knowledge of a particular domain.

This research proposed a scalable knowledge extraction pipeline with six stages that is able to extract a large number of targeted facts of a particular domain but it is building a knowledge base instead of knowledge graph [41]. The knowledge base can be used as a baseline for the construction of a knowledge graph. With the help of a knowledge graph we can discover more inter-relations among these triples.

The input to the pipeline is a corpus, a defined domain vocabulary, and a small set of entity types. The pipeline uses a combination of text filtering, Open IE, Turker annotation on samples, and precision prediction to generate its output [41]. The step 4 is about annotating and scoring the tuples extracted from Open IE and 15% of data is marked by humans. Based on this percentage of data, it trained a

model to annotate the rest of relations. In the last stage, it uses a set of schema mapping rules over the tuples that identify the similar relations and map them to a canonical generalized relation [41]. These canonical, generalized relations are known as canonical schemas, and the induction algorithm is called CASI (Canonical Schema Induction) [41].

Below Figure 3.3 shows the knowledge extraction pipeline for building a highly rich domain specific knowledge base.

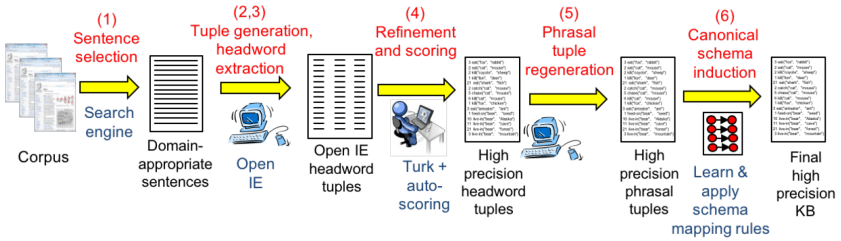

Figure 3.3: A six stage architectural pipeline of extraction and creating a domain specific knowledge base [41].

In the below Figure 3.4 it shows the output at different stages of the pipeline mentioned in Figure 3.3. In the first stage it takes a large collection of corpus along with vocabulary and types of information in that domain. This research did an experiment on the science domain. The search engine will select the top relevant sentences from the corpus, which are further annotated as candidates for the tuple generation. In the later stage, it generates the tuple using open IE and also performs some filtration of some keywords. In the refinement process humans will score the 15% of the data and remaining data is annotated with the help of a model which is trained based on the 15% of annotated data.

In the 5th step, for each headword it retrieves the original phrasal triples and adds sub-phrase versions to these phrasal tuples to the knowledge base [41]. For all constructed phrasal tuples, it computes a count threshold of PMI statistics feature which is the statistics of subject-predicate-object count and entire tuple in the google N-gram corpus. [41]. The phrasal tuple is valid and variants are also correct if the PMI score and count thresholds are equal. In the final stage, generalized relations are known as canonical schemas and all these are treated as tuples in the final knowledge base.

One of the drawbacks of this pipeline is it demands high quality vocabulary and type of information as input. There are domains for which it is hard to provide. The end output is highly dependent on the input provided by the expert, domain vocabulary and end task requirements. On the other hand, triples can only identify the limited information but not the details of those events. This approach only or-

Pipeline Example Outputs:
Inputs: corpus + vocabulary + types
1. Sentence selection:
"In addition, green leaves have chlorophyll.")
2. Tuple Generation:
("green leaves" "have" "chlorophyll")
3. Headword Extraction:
("leaf" "have" "chlorophyll")
4. Refinement and Scoring:
("leaf" "have" "chlorophyll") @0.89 (score)
5. Phrasal tuple generation:
("leaf" "have" "chlorophyll") @0.89 (score)
("green leaf" "have" "chlorophyll") @0.89 (score)
6. Relation Canonicalization:
("leaf" "have" "chlorophyll") @0.89 (score)
("green leaf" "have" "chlorophyll") @0.89 (score)
("leaf" "contain" "chlorophyll") @0.89 (score)
("green leaf" "contain" "chlorophyll") @0.89 (score)

Figure 3.4: The output of each stage of extraction pipeline [41].

ganized entities and relations into flat entity types and schema clusters [41]. The pipeline is adaptable to any domain set if we know the vocabulary, type of information and the expert to annotate the tuples extracted via the open IE mechanism.

3.5 Knowledge Graph from Unstructured Text

This is another approach of creating a knowledge graph from unstructured text known as T2KG. Many approaches to create a knowledge graph focus on the mapping of entities to its identical entity in an existing knowledge graph which is the subject or object of a triple [39]. The mapping of predicates extracted from unstructured text to its identical predicate is not considered and this research contributed to it. This way of mapping can reduce the heterogeneity of two graphs and increase the search ability over a knowledge graph. It proposed a hybrid combination of rule based approach and similarity based approach for mapping a predicate to its

identical predicate in a knowledge graph [39].

Below in Figure 3.5, it shows the architecture diagram of the T2KG system. It has five stages to process the unstructured text and at each stage it enriches the corresponding content using existing knowledge from knowledge graphs. In the first component it maps the entities followed by all Coreference resolution of entities in the second component. In the third stage triples are extracted which are a combination of subject-predicate-object. In a triple integration step, it generates text triples by using outputs from entity mapping component, the coreference resolution component and triple extraction component [39]. In the previous component we extract relation triples from unstructured text but the entity mapping and coreference resolution among the entities of such triples are not performed which leads to an ambiguous set of triples and interlinking to entities in the knowledge graph is not established [39]. Finally the predicate component will map the predicate of a text triple to a predefined predicate in existing knowledge graphs.

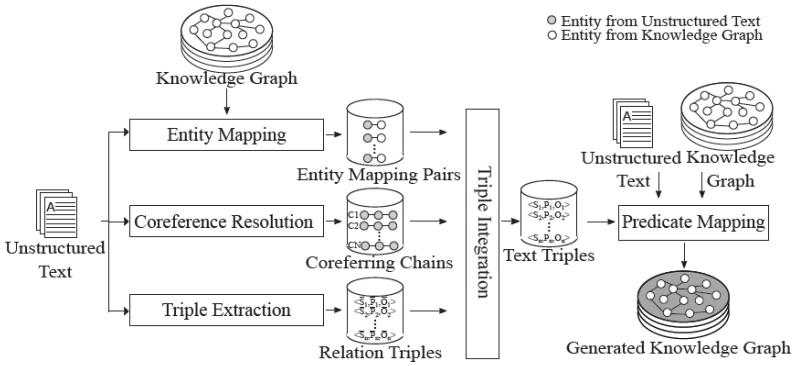

Figure 3.5: The Architectural data flow diagram of the T2KG system [39].

This is an automatic knowledge construction process which highly focuses on existing knowledge at each step in parallel with creating its own. The hybrid approach of mapping predicates using rule based and similarity based is used to achieve the goal of more homogeneity and reducing the sparsity of data.

3.6 Summary

In this chapter we have studied existing approaches which are used for the construction of a knowledge graph. The process of creating a knowledge graph consists of multiple stages and integrating tools together to achieve this task. By combining different technologies in an efficient way we can achieve better results. There are still challenges in the processing of unstructured data such as parsing of PDF and PPT formats to extract the textual data for further processing. It requires further

processing and engineering to generate a knowledge graph, which is proposed in our project in the next chapter. Our project will enable businesses to understand their raw text and discover the insights and relations among them.

Chapter 4

Design and Implementation of Knowledge Graph Pipeline

In this chapter, we introduce the architectural design and implementation of the data processing pipeline that is the basis for the construction of knowledge graphs from unstructured data. It targets **RQ1: How to design and implement a data processing pipeline for generating knowledge graphs?** In Section 4.1, we present how data will flow across multiple stages of the pipeline and mention the document types as a source of unstructured data. In Section 4.2, we define the extraction of raw text from these documents using different libraries. In Section 4.3, we describe the storage of processed unstructured data into a defined data model mapped to a Mongo database. In Section 4.4, we explain about how we make sense out of this raw data using natural language processing techniques, which helps in finding key aspects in the raw data. Subsequently in Section 4.5, we explain the last stage of the pipeline which is how the creation of a knowledge graph is achieved using a graph database and the integration of Neo4j in our graph processing service. In Section 4.6, we are evaluating the performance of our pipeline on a sample data set and summarizing this chapter in the Section 4.7.

4.1 Design of Knowledge Graph Pipeline

Knowledge Graphs are an important aspect for offering wide ranging integrated data to intelligent applications and ease out the ways of discovering underlying relations inside the data. We aim for the design and development of a data processing pipeline from an engineering perspective, which is designated for the construction of a knowledge graph. The data flows across three multiple stages in the pipeline, document parsing, natural language processing and sinking into a graph database. Our pipeline has two different data stores (Mongo Database and Neo4j Graph Database). Mongo database is dedicated to store textual data extracted from documents and Neo4j stores only key information such as entities and their relationships in the form of graph structure, which is further extracted from this text using natural lan-

guage processing as explained in the Section 4.4. At each step of the pipeline, data is manipulated and made ready for the next step. In Figure 4.1, the architecture diagram of the knowledge graph pipeline shows the flow of the data along various components.

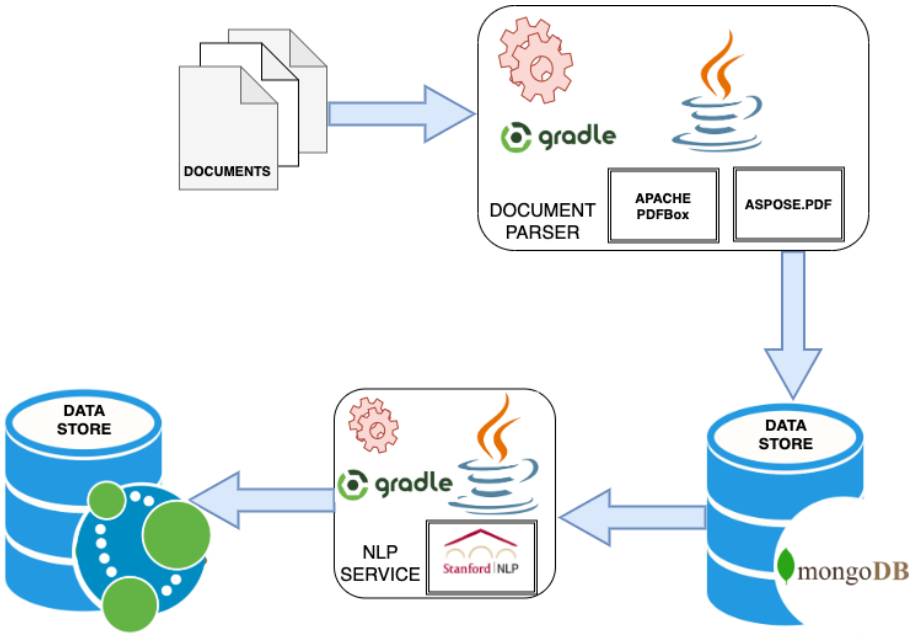

Figure 4.1: Knowledge Graph Pipeline for Unstructured Data.

Data Collection Sources

The execution of the pipeline is highly dependent on the data source. In our case, we are focusing on the processing of unstructured textual data originating from documents of different formats mainly PDF and PPT. All these documents are provided by the clients and other means of sources will be stored on the on-premises hardware or on the cloud storage. This storage acts as an access point for our document parsing service and all documents will be accessible for further processing. For the purpose of analysis, we are also crawling textual data from wikipedia articles of a relevant category and its subcategories.

Dependency Management Tool

In the pipeline, our core services are based on Java programming language and we use gradle as a build automation tool. Gradle is an open source build automation system mainly used for JVM(Java virtual machine) based languages. In our services, it compiles various dependent jars of libraries mentioned in the build.gradle

file. It also provides an easy way to customize the default behaviour and retrieves dependencies from Maven repository.

4.2 Document Parsing Service

The document parsing service extracts text from documents into an intermediate data format for further analysis. Due to the heterogeneous nature of these documents, the parsing not only transforms information but also misses out some relevant information. Thus, the result is an abstraction of the original document that means some of the information is lost. It is important to preserve a minimum amount of information that is needed to understand the parsed data by humans or machines. Primarily, we are only interested in the textual data and our pipeline does not analyze images and tabular data. Moreover, document parsing is the first step in the chain of data processing pipeline where we will precisely extract sentences, paragraphs and other metadata information such as title, font type, font name etc.

There are many libraries available which helps us in the building of custom implementation for the extraction of textual data from documents. Various libraries have support under different technology frameworks and programming languages. In below Table 4.1, we list some of the popular libraries widely used for parsing and text extraction from documents.

Table 4.1: Document Parsing Libraries.

Name	**Cost**	**Language Support**
Apache PDFBox	Free - Open Source	Java, Python wrapper
iText	Free - Open Source	Java, C#
PDFMiner	Free - Open Source	Python
Aspose.PDF	Paid - based on several parameters (From $999)	Java, C++, C#
PDFTRON	Paid - $4500 per year.	C++, Java, Python, PHP, Ruby, C
PDFTextStream	Paid - based on several services (From $100)	Java, C#

In our project, we are developing our document parsing service based on two different libraries which are discussed in the section below.

4.2.1 Integration of Apache PDFBox

The most challenging part of PDF documents are they do not have a well defined structure or specifications with columns like in CSV or tags such as in XML. It is a type of document with floating text with different styles on each page. These documents are designed from the perspective of humans not computers. To overcome

these challenges, we build our own document parsing service with the high level support of Apache PDFBox. We integrate this library using the gradle dependency management tool in Java programming language. Using this library, we will query our documents to search, filter, merge, sort and extract floating text from any PDF documents in a useful way.

Apache PDFBox is an open source tool written in Java programming language and it is widely used for working with PDF documents. In our project, we are leveraging this library only for the purpose of text extraction. Apart from text extraction, it supports creation of new PDF documents and editing of existing PDF documents. This library is developed by Apache Software Foundation [5].

We investigate extraction of text using the PDFTextStripper class and other properties such as font type, font size etc. The text can be extracted in the form of words, lines, paragraphs and also along with coordinates of characters. The formatting and arrangement of the chunks of text is not taken under consideration by this class. The end goal is to retrieve text of each page of the PDF document. We map the output text in the form of a data model which has different properties such as title, page number, list of sentences, font type and font size.

4.2.2 Integration of Aspose.PDF

With this library, we are processing both PDF and PPT documents for the purpose of text extraction. We develop our second document parsing service by integrating this library using Maven dependency tool which provides underlying interfaces. For PDF, we transform the streams of bytes from PDF into a unified Document class which is analyzed by paragraphAbsorber to split it into pages. Each page processes sections of a page and provides text out of it. In the case of PPT, the byte stream is transformed into a Presentation wrapper class which is further converted into the collection of slides. Each slide is processed to retrieve text boxes in it with the help of a text frame. For both formats we mapped corresponding output to our defined data model.

Apart from text extraction, it also supports creation of new PDF documents and editing of existing PDF documents. This library is developed by a company called Aspose Pty Ltd [2]. It is a family of products which offers a wide range of libraries for working with more than 100 file formats. The Aspose.PDF is a commercial library developed for working with documents and supports multiple document formats together with PDF and PPT. This library provides support in Java, C++ and .Net (a framework of C). In our project, we choose java based implementation as our many existing services are written in Java and it is also fast and lightweight.

The document parsing service implementation is based on the Java programming language using two different libraries, Apache PDFBox and Aspose.PDF. We choose Apache PDFBox because it is widely used by many developers. The library is continuously evolving with rapid development by open source contributors and holds extensive community support. One of the biggest non profit organizations,

Apache foundation is maintaining its releases and it is available free for use. We choose Aspose.PDF as it supports PPT, other formats which we can integrate in the future and our company already owns license from the respective provider.

4.2.3 Wikipedia Data Parsing

The main purpose of this project is to analyze client's data which comes in PDF and PPT documents but due to the confidentiality agreement of the client data, the results of their data cannot be published under this thesis. Our company mainly facilitates clients for developing business with insights of ongoing marketing trends. In order to make a relatively suitable analysis we choose to explore the content of wikipedia from the category of Marketing. Wikipedia is a multilingual online repository of textual data created and maintained as an open source project by a community of volunteers altering, using a wiki-based editing system [32].

Wikipedia has a wide collection of text based on different categories and sub categories belonging to different dimensions. The wikipedia data is structured and organized to a certain proportion with sections and tags which makes its parsing and extraction easier by using different libraries. There are many libraries available in python programming language for accessing this knowledge base programmatically. One of them is called urllib that provides necessary methods to pull content from HTML based web pages from this source.

In our project, we use one of the most popular python based libraries named Wikipedia-API. It's easy to use and supports extraction of texts, sections, links, categories, translations, etc from Wikipedia [33]. It also simplifies the design of the script by invoking wikipedia pages with only the name of category as a parameter instead of complete URL as in urllib. Based on the need, we can also define the depth of subcategories articles to be processed. The Wikipedia library helps us in the extraction of references listed in the article.

In our pipeline shown in Figure 4.1, we can feed the PDF articles mentioned in the references as a further on top analysis to the original article. As a part of this thesis, we analyze wikipedia text of a specific category Marketing [8] for the construction of a knowledge graph.

We can rely on this data as this is managed by non profit Wikimedia foundation and they provide essential infrastructure for hosting free knowledge. The data is also verified by volunteers of the world and also shares information that represents human diversity as well [31]. With our research and understanding about different libraries available in different languages we use them to extract data in a proficient way.

From Figure 4.1, we replace the two components of the knowledge graph pipeline and update the pipeline for wikipedia data parsing in the below Figure 4.2.

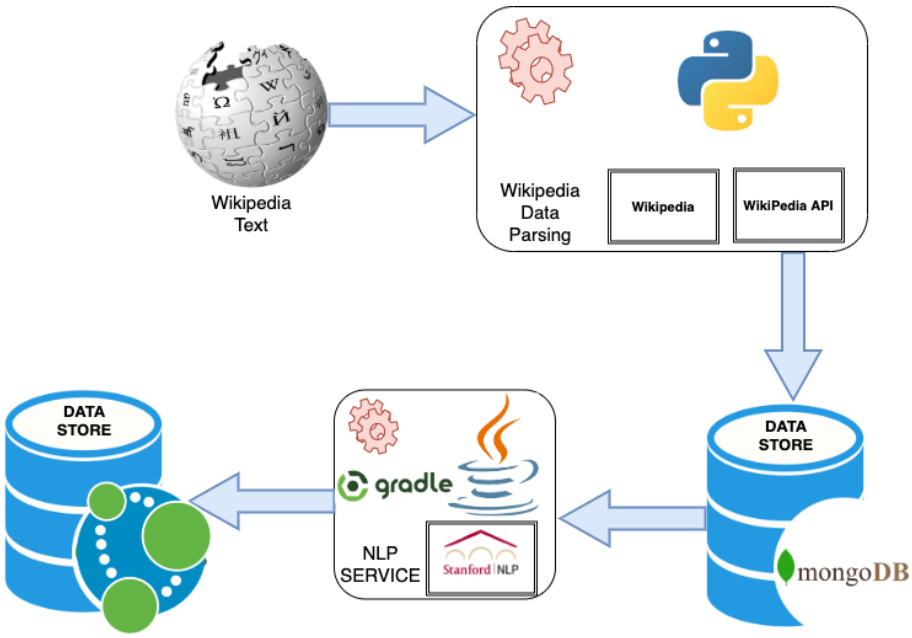

Figure 4.2: Knowledge Graph Pipeline for Wikipedia Data.

4.3 Data Storage

The distinct instances of document parser service use different libraries for the extraction of data from different types of documents and open source knowledge space such as Wikipedia. Upon extraction of textual data from various sources the intermediary output is stored in the Mongo database. We define a data model which maps the output of the document parser service in a JSON format and pushes data as a JSON document in the Mongo database. The Mongo database provides integration of its functionality with many programming languages and we only use it in Java and python (as our document parsing services are written in these two programming languages). In our project, we are using a community version of Mongo database which is open source and free to use. There is an enterprise edition available based on subscription with advanced features and comprehensive deployment support of Mongo database, on disk encryption, auditing etc [19]. Based on our requirements, we get all the needed basic functionality with the community version.

There are many relational and non relational databases available but we choose MongoDB for the following characteristics. MongoDB is an object-oriented, simple, dynamic and scalable NoSQL database [30]. Unlike traditional relational databases, MongoDB stores data in the form of documents in a collection instead of storing the data into columns and rows. MongoDB stores data in flexible, JSON-like documents meaning fields can vary from document to document and data

structure can be changed over the period of time [19]. The aim of MongoDB is to implement a data store that provides high performance, quality and automatic scaling [30], which eventually will expedite our pipeline intermediary phase. The document storage mechanism of MongoDB makes it faster than MYSQL because the entire related data is in one place rather than in different tables. Hence, the retrieval speed of data is faster to fetch a single document from MongoDB than to apply JOIN across different tables is MYSQL.

Our needs are not mainly composed of transactional systems that fully comply with ACID properties and it makes NoSQL a better choice here than SQL. MongoDB is also an ideal fit for the cloud deployment as it is enabled with horizontal scale-out architecture empowered by sharding and agility supported by the cloud computing [30]. The down side of MongoDB is it does not support the execution of stored procedures and functions for binding logic at the level of the database. In our project, we are not using anything related to it so it will not affect the design of our pipeline. MongoDB is also highly recommended for storing content of articles and blogs.

Data stored in the MongoDB can be accessed through a rich query language which supports all read and write operations. Our document parsing services will only insert data into mongoDB whereas retrieval is done by the natural language processing module for further processing.

4.4 Natural Language Processing Service

The main perspective of building sense out of unstructured textual data is enabled by a sub field of artificial intelligence namely natural language processing. This raw format of data cannot be easily converted into rows and columns directly to store in a relational database, however, we need to extract useful features out of it to make it more sensible. The textual data extracted in the earlier step is ingested to this stage for further syntactical and semantical analysis by using natural language processing techniques. The module of natural language processing is considered as the intermediary and highly centralized component in our pipeline.

There are several natural language processing libraries available which helps in the extraction of insights from the mostly used natural languages. The major challenge for all natural language processing libraries is the process, storage and maintenance. The task of building a natural language processing pipeline is not an easy process as it is the mixture of several functionalities such as splitting of sentences, POS(Part of Speech) tagging, lemmatization, etc. For better results, a large processing power is required to build models from large and diversified data sets. For our data processing pipeline, we overcome this challenge by using the Stanford coreNLP library that provides default models which are trained on huge

data sets.

Stanford CoreNLP, a very popular library for natural language processing is developed by Stanford university. The library is written in Java programming language and offers a variety of other mediums for interaction such as command line, native Java programmatic APIs or deployed as a web API server [22]. In our data processing pipeline, we are interacting with this library in a programmatic way by including gradle dependencies in our Java based natural language processing service. It provides basic building blocks of semantic analysis and packaged with the predefined set of state of the art trained models for running natural language processing tasks. It supports many natural languages but in our project we only use it for the purpose of English.

From the implementation perspective, our natural language processing service instantiate an annotation pipeline with several annotators to annotate textual data. This pipeline object is a map with the key annotators in the sequence. The annotators are declared as an individual task that should be applied on the text assigned in the annotation object.

In the beginning, the annotation object contains only raw text and the annotation pipeline will execute the declared annotators to construct the result object. This result object will contain all the examination data included by the annotators and the output can be in XML or plain text forms. An annotation object stores the results of each annotator in a map data structure. Due to map data structure, we can access results in O(1) time complexity, which makes it faster. Each annotator is also defined with the key from the annotation object then it applies natural language processing on it and writes the result back to the annotation object. As we have multiple annotators, hence the result of each one is written under different keys rather than being overwritten.

we are examining the Stanford coreNLP library for the purpose of named entity recognition, semantic analysis, POS tagging for multi-word expressions and relation extraction which will be helpful for the construction of a knowledge graph and defining correspondence between named entities. Apart from built in named entity recognition models, we can define our own custom models for this purpose which will make the context of particular domain text much clearer and meaningful. The library offers very flexible and extensible methods which can be easily configured. It provides necessary basic building blocks for developing higher level and domain specific customization of textual analysis.

The Figure 4.3 presents the system architecture of Stanford coreNLP annotation pipeline and it represents the execution flow of the annotation pipeline and some widely used annotators in our natural language processing service.

Annotators mentioned in Figure 4.3 are widely used in natural language pro-

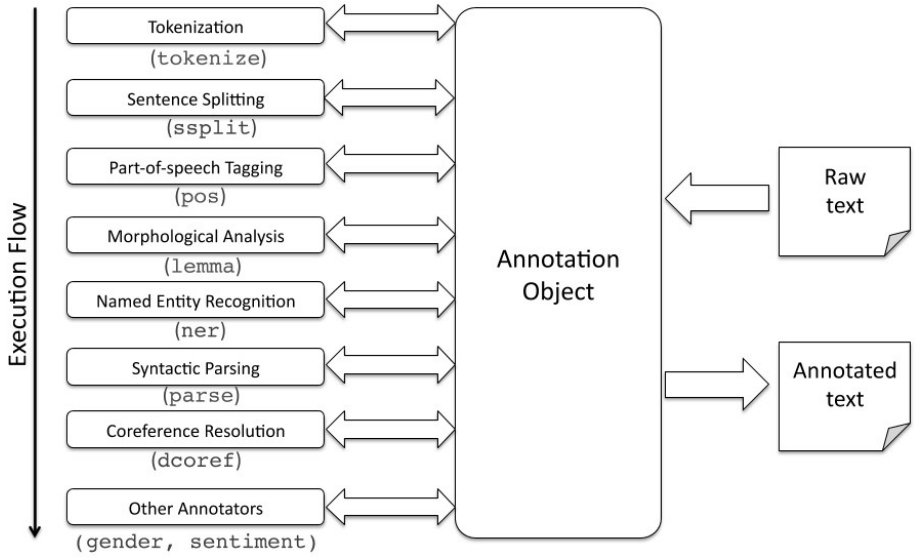

Figure 4.3: Stanford CoreNLP Annotation Pipeline [22].

cessing and briefly described below [22]. Our natural language processing service is using a subset of these mentioned to extract relevant semantics out of text.

1. **Tokenize:** It is considered as the first processor used in the pipeline. It is the process of dividing text into smaller chunks known as tokens. Tokens can be words, numbers and punctuation marks. The annotator does this task by locating boundaries of words. The character offsets of each token is also saved by the tokenizer.

2. **Ssplit:** It is used to split a sequence of tokens into sentences.

3. **POS:** It is used for labelling each token with corresponding parts of speech tag.

4. **Lemma:** It generates the word by normalizing the tokens to their root forms.

5. **NER:** The named entity recognition annotator helps in identifying and then categorizing key information in the text. The tokens are labelled based on the predefined models. NER in English by default identifies entity type of person, company, location, currency, number, date, time, etc.

6. **Parse:** The parse annotator figures out the grammatical structure of the sentences and outputs syntactical analysis in the form of phrase structure tree. It also provides basic to advanced dependencies.

7. **Sentiment:** It assigns an overall score to the given text and classifies its sentiments as a positive, negative or neutral. It is built upon a deep learning model and computes sentiments based on the composition of words in longer

phrases. It is not easy to fool the system as the prediction is not a summation of count based approach of individual positive and negative words. With this intent of the text is lost and important knowledge is lost.

8. **openIE:** It stands for open information extract which represents the domain triples and considered as very useful information for the construction of a knowledge graph. It defines subject, a relation and the object of relation.

4.5 Knowledge Graph Service

Knowledge graph service is the final stage of our data processing pipeline. The natural language processing service will produce data for the generation of a knowledge graph. The data in the form of nodes and relationships is dumped in the graph database known as Neo4j [21]. The graph structure provides a better visualization of knowledge in a more convenient and understandable manner. The relations between different things are easy to discover for the end user.

There are many graph databases available but we choose the most popular and widely used Neo4j in our data processing pipeline because it is the leading native graph database and graph platform. Apart from this, a very vast documentation is provided by Neo4j organization which makes the concepts clear and easier to interpret for developers, many big organizations are also using this in production and a lot of community support is available too. In this project, we are using the community version 4.0.4 of Neo4j although there is a commercial license for enterprise with extra features and support is present, such as security, high performance and clustering of multiple nodes.

Neo4j helps us in representing the information in the form of a graph. Neo4j supports integration of its driver for connecting and interacting with database in many programming languages natively such as Java, .Net, JavaScript, Go and Python. There are wrappers that exist in other programming languages which are contributed by the community as an extension of existing one.

The Neo4j database can be installed on the local machine or on any commodity node in the cloud. When we start Neo4j as a server it provides an interactive user interface via a built in Neo4j browser application that is running locally on the hosted machine. Using Cypher query language we can run queries on the browser based console and explore results in different formats (graph, table, code and text).

4.5.1 Integration of Spring Boot and Neo4j in Graph Processing Service.

Our graph processing service is based on Java programming language using the popular framework Spring Boot which also provides many wrappers supporting Neo4j operations. Our graph processing service interacts with Neo4j database programmatically in Java. There are various components used in our graph processing

service for the integration of Neo4j. Below Figure 4.4 depicts the high level architecture of the integration design.

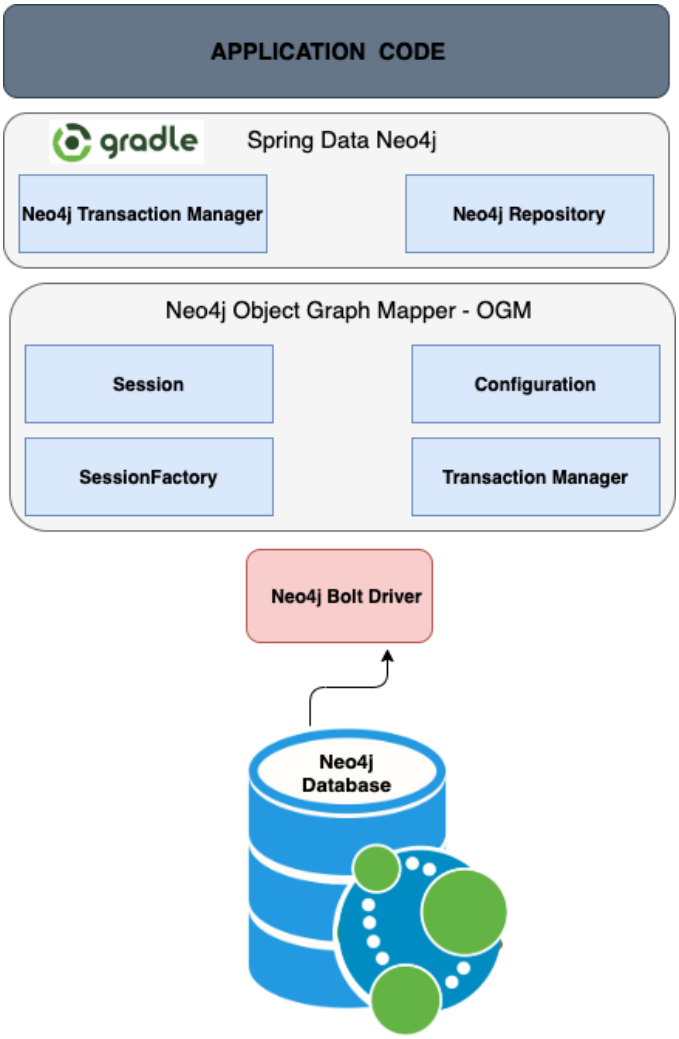

Figure 4.4: Spring Boot Data and Neo4j Integration Architecture Diagram.

4.5.2 Neo4j Bolt Driver

The graph processing service creates a connection to Neo4j using a Bolt binary protocol. It is mainly used for client server communication in database applications and is also used for Neo4j [26]. The underlying dependencies of drivers are managed by a build automation tool known as Gradle and it provides necessary interfaces and methods for its successful creation. The credentials are provided us-

ing a configuration object. The configuration object is used by the SessionFactory which is required by spring data Neo4j to create a Session object as needed. The classpath of the data models package which are mapped to the nodes and relationships in the Neo4j is also provided to the SessionFactory as a second parameter for scanning all domain objects. We configure SessionFactory in such a way that it inspects all required packages. The session is managed by the Neo4j Transaction Manager which handles the transaction of the database at the application level. Our application is connected to the only one instance of Neo4j, hence there is one SessionFactory per application.

4.5.3 Spring Data Neo4j OGM - Object Graph Mapper

The output of natural language processing service will be transformed in the form of a data model which will be stored in the Neo4j database. This is a user defined data model which is mapped to the graph database with the help of object graph mapper. Neo4j object graph mapper is a powerful library that provides a simple mechanism to manage domain objects with Neo4j. It maps nodes and relationships to the objects in the domain model defined as a plain old java object, POJO classes. The class attributes are mapped to the properties of node or relationship. This wrapper simplifies the persistence and retrieval of data to and from Neo4j by providing key abstractions which takes away all the complexities of low level drivers [16].

With the support of object graph mapper we can also run custom native queries where default query operations are not sufficient. The basic create, read, update and delete - CRUD operations are easily performed with the help of Neo4JRespository interface, supported by this library. Once the output is retrieved from the graph database, the graph can be returned in the JSON format to a user interface (frontend application) for displaying it to the end user.

In Figure 4.5 the diagram shows the flow of data from graph processing service to and from Neo4j and then forwarding it on any frontend application.

4.6 Overview of Experiments

We are doing experiments for the analysis of performance of our services involved in the data processing pipeline. The performance analysis is based on the quality of output and the execution time. The execution time will help us to examine the computational expensive component in the pipeline. The experimental analysis will help us to further improve the architecture and design of the pipeline. We want data extraction and parsing of documents as fast as possible because further components in the pipeline are dependent on this step.

Our experiments with respect to each component are as follows.

- Document parsing service will evaluate the execution time of parsing of each page from documents and compare the result between two parsers.

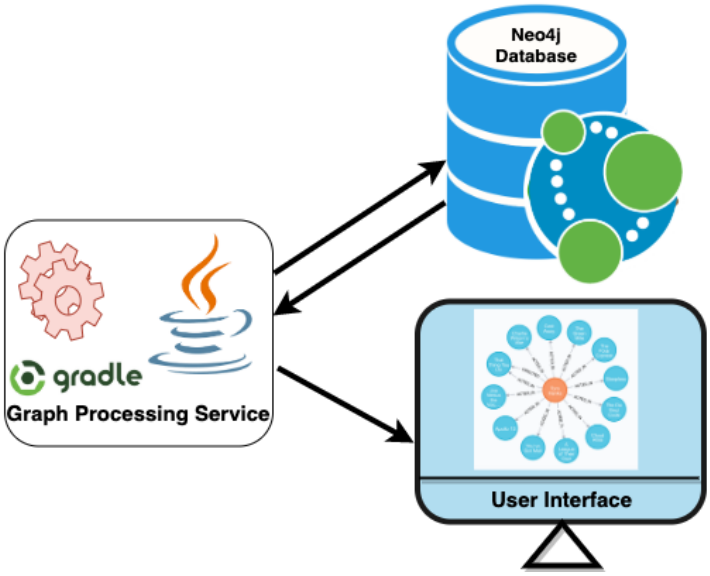

Figure 4.5: Data Flow across Graph Processing Service, Neo4j and User Interface.

- Wikipedia parsing service will present the details of data extraction and the next component will analyze this raw text.

- Natural language processing service will analyze the results and execution time of key features which helps us in understanding the semantics of the raw text extracted in the earlier step.

- The knowledge graph service will show an example of a knowledge graph retrieved from Neo4j, the graph database.

4.6.1 Experiment Requirements and Evaluation

Each service is dedicatedly designed for one task, which will make it highly flexible, maintainable, scalable and testable. With the microservice architectural design, all services are loosely coupled and can be deployed independently. It also makes organization and fixing of bugs easier with less ripple impact on other components in the system. For our testing purpose, each service is executed manually and output of one service is used by the following service which means that we cannot run all services in parallel for the same input text. We run individual service which are designed and developed for dedicated tasks in the pipeline as shown in Figures 4.1 and 4.2 on a machine with the following hardware specifications and required software packages.

Hardware Specifications

Operating System: Macintosh Operating System Catalina, Version 10.15.4
CPU: 2.2 GHz Quad-Core (4 cores) Intel Core i7
Random Access Memory: 16 GB
Solid State Drive Storage: 256 GB

Software Package Specifications

There are some software package requirements listed below for the execution of each service.

1. **Document Parser Service:** Document parser service is based on the Java programming language, hence it requires a Java Development Kit (JDK) of version 11 installed on the machine. The service itself consists of several gradle dependencies such as spring boot framework of version 2.0.3, Apache PDFBox library of version 2.0.14 , Mongo database driver of version 3.10.2 and lombok plugin of version 1.18.4 which reduces the boiler plate code by providing specific annotations such as @Data. The second instance of the document parsing service works similarly but with a major difference of the Aspose library dependency of version 18.5.

2. **Wikipedia Data Parsing Service:** Wikipedia data parsing service is based on the Python programming language, hence it requires Python3 package version 3.7 installed on the machine. This service depends on two libraries wikipedia-API and wikipedia. The installation of these two libraries is done using a python package manager known as PIP3 which comes along with the Python3. It also uses a Mongo database of version 3.10.2 installed via PIP3.

3. **Natural Language Processing Service:** The natural language processing service is based on the Java programming language, hence it also requires a Java Development Kit (JDK) version 11 installed on the machine. The service itself consists of several gradle dependencies of spring boot framework of version 2.0.3, Stanford coreNLP library of version 3.9.1 with models for English language, Mongo database driver of version 3.10.2 and Neo4j dependencies which are mentioned in knowledge graph service.

4. **Knowledge Graph Service:** The knowledge graph service is based on the Java programming language, hence it also requires a Java Development Kit (JDK) of version 11 installed on the machine. The service also consists of a gradle file which manages core dependencies for Neo4j such as Neo4j-ogm-core of version 3.2.11, spring data Neo4j of version 2.3.0, neo4j-ogm-api of version 3.2.11 and spring boot framework of version 2.0.3.

Parser Performance Evaluation

For the performance evaluation of document parser service based on different libraries, we run each parser 1000 times on PDF documents with increasing range of pages with the factor of 5. The line chart in Figure 4.6 shows the comparison between Apache PDFBox and Aspose.PDF based implementations. From the plot, it is evident that Apache PDFBox is much faster than Aspose.PDF. With regards to the performance for the largest input size, the Aspose.PDF based approach is approximately $\sim$ 10x slower than Apache PDFBox implementation. With Apache PDFBox the time is roughly increasing constantly as number of pages increases but with Aspose.PDF it is not increasing in a dramatic way. Hence for our data processing pipeline, we will choose Apache PDFBox.

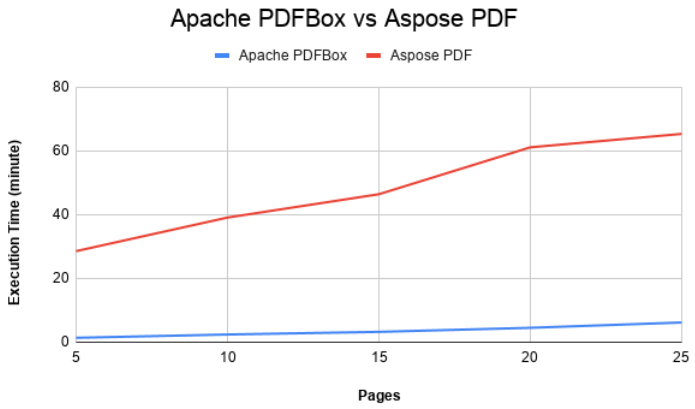

Figure 4.6: Execution time of Apache PDFBox and Aspose.PDF on different number of pages.

Wikipedia Parsing Service Example

For a sample analysis of textual data from a particular domain, we explore a category of Marketing on wikipedia. Our wikipedia parsing service extracts textual data from all underlying pages and sub-categories main page under this category. In wikipedia, the hierarchy structure of categories are nested but we are only looking at one level. The Table 4.2 shows the following results.

Table 4.2: Wikipedia Parsing Service Results.

Category	Marketing
Execution Time	22.74 seconds
Total Pages	28
Subcategories	30
Data Size	1.3MB
Total Links	362

Natural Language Processing Service Analysis

Our natural language processing service has a pipeline of annotators which runs on all sentences. The Figure 4.7 shows the plot of execution time among 9 annotators (tokenize, ssplit, pos, lemma, depparse, ner, natlog, parse, sentiment) on sentences. This service processes the textual data extracted from wikipedia of which details are mentioned in table 4.2. As per the graph there is a linear relationship among them, execution time increases as the number of sentences increase. Due to variation of sentence length, there is an edge case where time decreases for a certain number of sentences and then again it's increasing in a linear manner. There are roughly 6700 sentences in this dataset for which it roughly takes approximately 72 minutes which shows that 90 sentences per minute. The time per sentence will be affected by the length of sentences as it tokenizes the paragraph based on the full stop.

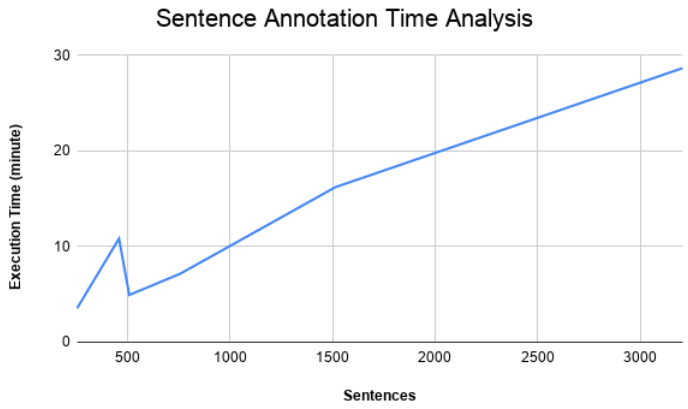

Figure 4.7: Linear Relation of annotating number of sentences with respect to time.

Named entity recognition is the key feature offered by the natural language processing domain. It helps us analyze the core aspects discussed in the documents. In our dataset, we run NER annotator to find the key entities mentioned in the text. With the default models Stanford coreNLP recognizes 21 entity types in this text. The Figure 4.8 shows the count of each entity type in a bar chart. From the bar chart **NUMBER** type is the most highest occurring whereas **CRIMINAL_CHARGE** is the lowest in the number. It is not the distinct count of entities in the text instead the overall frequency of each entity.

In this textual data, we process approximately 6700 sentences. An interesting characteristic is around 49% sentences have no any entity mentioned, whereas 51% sentences consist of one or more entities. The Figure 4.9 illustrates the horizontal plot of the bar chart of NER and without NER sentences. In order to reduce the noise from data for further analysis we can ignore such sentences which consist of zero entity.

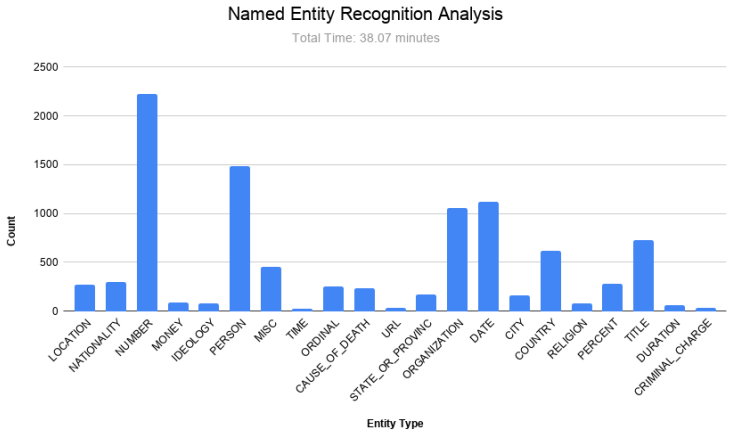

Figure 4.8: Representation of different entity types and their count in the dataset.

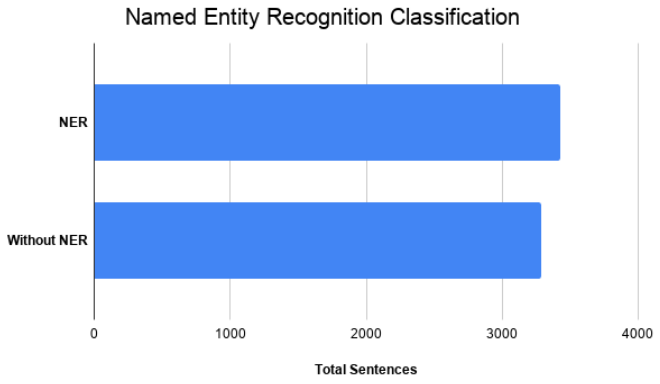

Figure 4.9: Classification of sentences with NER and without NER.

One of the major challenges with coreNLP is that it loads all the models in the memory while applying corresponding annotators on the input text. For very large input text we need more memory or we use a few annotators. One way of handling this is running multiple instances of natural language processing service to handle each annotator individually. This approach of parallelism will significantly optimize the performance of the pipeline and reduce the risk of out of memory exceptions.

Another solution to this is to process a large text with all annotators into small batches. With this approach, we can run a single instance of natural language processing service but it will take more time as compared to previous technique.

Sentiment Analysis

The overall semantics of the text is extracted by the sentiment analysis. It gives the feedback of the text in terms of positive, negative and neutral. For positive the value is 1, for negative the value is -1 and for neutral it's 0. The Figure 4.10 shows that the text of wikipedia for selected category has more negative sentences as compared to positive and neutral. With the help of sentiment analysis we can better understand consumer opinion for clients in the market.

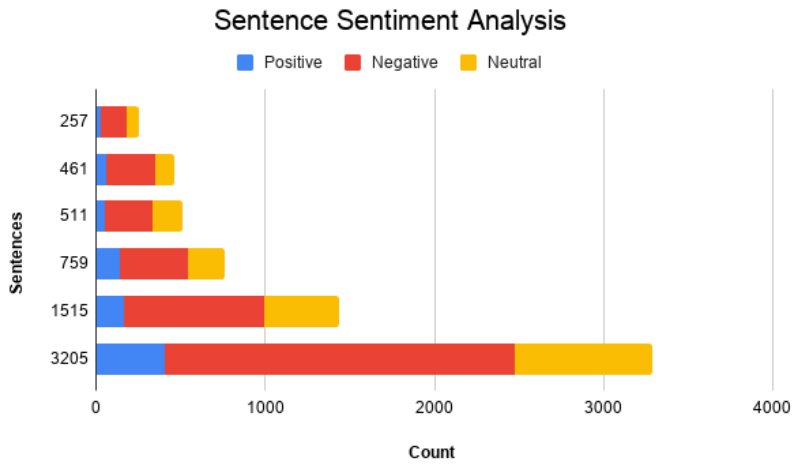

Figure 4.10: Distribution of sentences with all sentiments in the dataset

Example Knowledge Graph

The core information extracted from textual data using natural language processing service is stored in the Neo4j database. By running Cypher queries in Neo4j we can create dynamic graphs based on the need of the customers. The data and relations in the Neo4j can be interlinked easily for better visualization and faster traversing. When it comes to large datasets the relations are not easy to traverse around different dimensions but graph databases made it quicker and easier. In our sample dataset, we will show an example graph in Figure 4.11 which reflects information about a wikipedia page and it can be associated with any document of this category in the future. The graph in Figure 4.11 shows a total of 14 nodes with 13 edges defining relationship among them. The creation time of this graph is 758 milliseconds.

4.7 Summary

In this chapter, we have come up with an architecture of a data processing pipeline for the construction of a knowledge graph. We develop microservice for each task

Figure 4.11: A sample knowledge graph representing category and some of its linked links.

of a pipeline and evaluate their performance with a sample dataset from wikipedia. Due to the confidentiality agreements of clients with the company, we cannot process their data under this project. The core component of this pipeline is natural language processing service which helps us in the extraction of key information from raw textual data.

The first part of this chapter focuses on the architectural components of the pipeline. We analyze different libraries and frameworks for the development of this pipeline. The main part of this implementation is done in the Java programming language. The main source of data stream for this pipeline is in the form of PDF's and PPT's. For experimental purposes, we process the sample data set of wikipedia from the category of Marketing.

In our evaluation, we find that Apache PDFBox is much faster than Aspose.PDF whereas Aspose.PDF is also able to parse PPT's but Apache PDFBox does not. The Stanford coreNLP library is packaged with built in models for the recognition of entities and sentiment analysis. We can also create our custom models and train our classifier on our dataset, given that we have enough annotated samples of the data. We determine the performance of the Stanford coreNLP library on wikipedia

articles belonging to one category. The results are shown in the figures in the Section 4.6. Our pipeline uses two different data stores, Mongo database which is responsible for storing intermediary results and the final result of meaningful information extracted by natural language processing service is stored in the Neo4j which constructs the knowledge graph.

However, the entire pipeline cannot be run in parallel for the same documents but components are independently designed for easy maintainability and scalability.

Chapter 5

Analysis of Cloud Based Natural Language Processing Services

In this chapter, we introduce three cloud based natural language processing services from Amazon, Google and IBM. This chapter addresses our **RQ2: How to choose the most suitable cloud based natural language processing service based on cost, performance analysis along with features and flexibility of individual service with other cloud providers?** In Section 5.1, we briefly describe the fundamentals of cloud computing and services. In the next three sections; Section 5.2 discusses Amazon Comprehend, Section 5.3 presents Google Cloud Natural Language API and Section 5.4 inspects IBM Watson Natural Language Understanding. In each section of these services, we describe the high level introduction, features, pricing models, perform experimental evaluation of all features on a sample data set and present results of them. Subsequently in the Section 5.5, we perform a detailed comparison analysis among features of these three services and provide the metrics of choosing the most suitable based on the quality of output, performance, flexibility of configurable parameters and cost. We prove our choice of cloud based natural language processing service based on the experimental analysis performed in each of Section 5.2, 5.3 and 5.4. Finally in Section 5.6, we provide a short summary of this chapter.

5.1 Introduction

The notion of cloud computing is offering a wide range of computing resources to the consumers that can be easily accessed over the internet and the entire infrastructure is managed by the cloud service providers [18]. The advent of cloud computing takes away the complexity of setting up the huge infrastructure of servers, etc from the users and enables customers to focus on building their core business values for end users. The cloud technology helps us in the rapid development of applications, scalable deployment, efficient management of resources and its optimized pricing model of pay per use.

Today, not all but most of the organizations are using the power of cloud computing to become more agile and faster in the delivery of their core business values to the customers . There are many cloud providers but few big names lead a significant market share are Amazon, Google, IBM and Microsoft. All of them offer different services ranging from computational power to processing to storage and much more. In the RQ1, we propose the architectural design and implementation of a data processing pipeline which has a core component of natural language processing. There are many open source libraries available for natural language processing but the cloud based solutions can decrease the engineering effort, maintenance of codebase with a little cost as per use [18].

For early projects, it is wiser to use such services than hiring a full fledged team of experts, which is significantly expensive for the company. For us, cloud computing is abstracting the complex problems into a simple easy to use service. In the RQ2, we are analyzing cloud based natural language processing services from different perspectives offered by three well known providers Amazon, Google and IBM.

5.2 Amazon Comprehend

The natural language processing service from amazon is known as Amazon Comprehend was launched in 2017. This service is continuously trained on various models with a bunch of annotated data sets from different domains which makes text analysis efficient and easy for the end user [3]. Behind the scene, it is using deep learning, a max net waste sequence to sequence models for training. Amazon Comprehend claims the addition of new training data sets on a daily basis which means that models get better with improved accuracy over the period of time [3].

It has the capability of language detection, recognizing the entities and those are categorized like organization, person, brands, etc, key phrases which are noun based keywords to really understand the content, sentiments and syntax analysis in the text. Amazon Comprehend provides a well built-in feature of examining a bunch of documents to classify them based on the frequency of similar keywords within them. This type of topic modelling is beneficial to sort out a large corpus of documents into topics that are relevant to each other. The job of topic modelling requires a large number of documents and for suitable results at least 1000 documents are recommended. The number of topics is a configurable parameter and it can detect a maximum of 100 topics in a collection [3].

Amazon Comprehend is a web based service that can be consumed directly by calling API in one of the programming languages supported by this service. We can also submit a job via a web based user interface known as Amazon Comprehend console. The input data for the job is stored in the S3 bucket. Amazon S3 is a simple storage service offered by Amazon Web Services and in our case we are storing text files. Documents can be submitted for analysis in two formats (one document per file and one document per line) but for large documents one

document per file is highly recommended. In this format, each file is treated as one document. This is the most suitable option for our company as we have collection of large corpus.

The built-in feature of Detect Entity in Amazon Comprehend classifies all entities into 9 categories. The Table 5.1 summarizes the information of each type of entity.

Table 5.1: Amazon Comprehend Default Entity Types.

COMMERCIAL_ITEM	It marks the branded products.
DATE	It recognizes different variants of date, day, month, year or time.
EVENT	It can be any festival, party or election etc.
LOCATION	It determines the country, province, city, street, building etc.
ORGANIZATION	It points out large organizations, government, religion, sports teams, institutions etc.
OTHER	Entity which does not classified under any other mentioned types.
PERSON	It identifies the human names, group of people, characters etc.
QUANTITY	It filters out numeric values such as currency, numbers, percentages, bytes, etc.
TITLE	It spots the tagline given to any piece of work such as book, song, movie etc.

5.2.1 Amazon Comprehend Custom

Amazon Comprehend runs all operations on our text based on the default built in models. In order to train a custom model specifically on a particular data set, amazon comprehend custom provides this ability without even worrying about the techniques of machine learning based natural language processing algorithms. It uses automatic machine learning to build customized natural language processing models on our dataset. For a limited amount of data, it uses a technique of transfer learning for building a highly accurate model. In transfer learning, Amazon also trains our custom model based on existing trained models of entity recognition with large data sets of different dimensions. With this approach it improves the results of the custom entity recognizer model in case of fewer training data.

Custom Entity Recognition

With custom entity recognition, we can recognize new types of entities which are not supported by default models listed in the Table 5.1. The service requires a data set for the training of the model containing a set of such entities, with a set of annotated documents or a list of entities and their labels. The service itself will

decide the best algorithm and parameters for the training of the model and will seek for the most desirable combination of these factors. There is a limitation of 12 custom entities at a time per model. After training, we can detect a maximum of 12 custom entities on our test data.

Custom Classification

It also provides the feature of building and training of our own custom models for the classification of documents. We can train our custom classifier model in two ways (Multi-class or Multi-label mode). Multi-class models will assign a single class whereas multi-label models can assign more than one class to each document. In the training of a model, a minimum of 10 documents are required for each class but for better results amazon recommends at least 50 or more documents for each class. The total size of training data should not exceed 5GB. It uses between 10 and 20 percent of training data for the testing purpose on a newly trained custom classifier. Once training and testing of a model is completed, it returns some metrics such as Accuracy, Precision, Recall, F1-score and Hamming loss.

Amazon Comprehend Medical

This service offers the analysis of health related data which is very unstructured in the form of doctors notes, lab reports, test results and discharge summaries [4]. This service can analyze and examine documents only in English language. The detected entities are also linked to the concepts in medical ontologies, including the RxNorm and ICD-10-CM knowledge bases [2]. The entities are detected in the following five categories such as ANATOMY, MEDICAL CONDITION, MEDICATION, PROTECTED HEALTH INFORMATION and TEST TREATMENT PROCEDURE [4]. It also provides a dedicated feature of detecting protected health information as Detect PHI associated with a confidence score. It covers personal information of patients such as name, address, age, profession, email etc. With the help of this service we can offer better insights of health data to our clients from the industry of pharmacy and hospitals.

5.2.2 Pricing Analysis

Amazon Comprehend charges for the documents we analyze without any minimum or upfront cost. It charges based on the amount of text we analyze with respective features. The cost is measured in units of 100 characters with a minimum charge per request is 3 units (300 characters). Initially, 5M characters (50K units) can be processed for free. The topic modelling is charged based on the size of the documents instead of units. The first 100MB are charged at a flat rate of $1 and above every next MB it costs $0.004 per MB [3].

Below table 5.2 shows the cost of each feature based on different segment sizes of units.

Table 5.2: Amazon Comprehend Cost Per Feature.

Feature	Up to 10M Units	From 10M-50M Units	Above 50M Units
Entity Recognition, Sentiment Analysis, Key Phrase Extraction, Language Detection	$0.0001	$0.00005	$0.000025
Syntax Analysis	$0.00005	$0.000025	$0.0000125

For the custom training of a model it charges $3 per hour and $0.5 per month for the model management. In case of asynchronous classification and entity recognition it charges $0.0005 per unit (a unit consists of 100 characters) while for synchronous classification and entity recognition it charges $0.0005 per IU (inference unit) per second. One inference unit gives the throughput of 100 characters per second.

The comprehend medical API request is charged per unit which is composed of 100 characters. The Detect entity feature costs $0.01 per unit and Detect PHI costs $0.0014 per unit. For the first three months we can process 2.5M characters (25K units) free of cost.

5.2.3 Performance Analysis

For the performance analysis, We will evaluate each feature of this service on a sample data set of category Marketing from wikipedia, which consists of around 60 documents. For each feature, we will analyze the results and execution time. We create and run analysis jobs via Amazon Comprehend console, which will read input data set from S3 bucket and output is also stored in the S3 bucket. The response is returned in the JSON (JavaScript Object Notation) format with different attributes for each feature. The integration of AWS SDK (Software Development Kit) in the Java programming language requires authentication using an IAM (Identity Access Management) user and in our case, we are using an AWS Educate account which does not provide enough rights to create this user.

Entity Recognition

There are a total of 8559 entities detected in the entire text of around 6700 sentences. The response consists of the following attributes. Each entity has a begin and end offset mentioned along with the type and text of the entity itself. For each entity, it also returns a score which shows the level of confidence that this service has correctly detected the entity type. We can define a threshold to filter out the entities with low scores to reduce the risk of false detections. Based on our assumption the entities of threshold score below 0.5 will be discarded or those entities might need human review. The assignment of each entity to a category is visualized in the below bar chart.

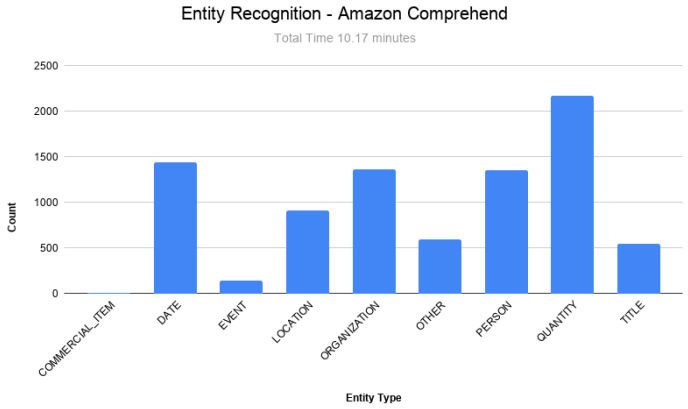

Figure 5.1: Representation of different entity types and their count in the dataset.

KeyPhrase Detection

This is a very useful feature for the deep understanding of documents based on a combination of popular words also known as multi-word expressions. It automatically recognizes keyphrases without taking the input size of the phrase. In the response, it mentions the begin and end offset of each phrase along with the score which defines the level of confidence that it finds the phrase correctly. We assume that the key phrases of threshold score below 0.5 will be removed. It also returns several single word phrases, which are not considered in our analysis. This job takes a total execution time of 10.1 minutes. In the bar chart below, we list keyphrases of the first 10 files.

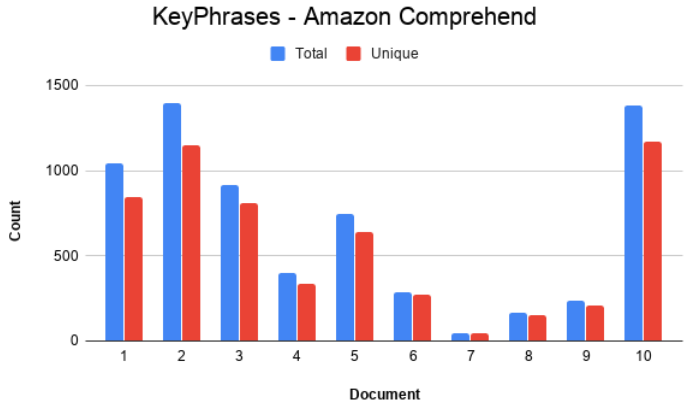

Figure 5.2: Representation of total vs unique keyphrases in each document.

Topic Modelling

With topic modelling, the number of topics is defined as the input parameter and we set this to a value of 5. In response, it returns two files, topic-terms.csv and doc-topics.csv. The topic-terms.csv consists of a list of topics and by default each topic has 10 most likely terms associated with a weight. In the below Table 5.3, we include the result of each topic's top weighted term.

Table 5.3: Amazon Comprehend Topic Classification based on Weight.

Topic	Term	Weight
000	Product	0.123
001	Market	0.204
002	Study	0.002
003	Company	0.471
004	Tag	0.344

In the second file doc-topics.csv, each document is assigned a topic with a proportional number which shows the relevancy of that topic. We find that a single document can be labelled with multiple topics with different proportion values. We start the experiment with 5 topics and increased the number of topics by 5 at a time up to 50 and calculated the execution time for each run that is shown in the graph below. On average, it takes the same execution time for any number of topics but the slight difference is due to the provisioning of the resources in the cloud.

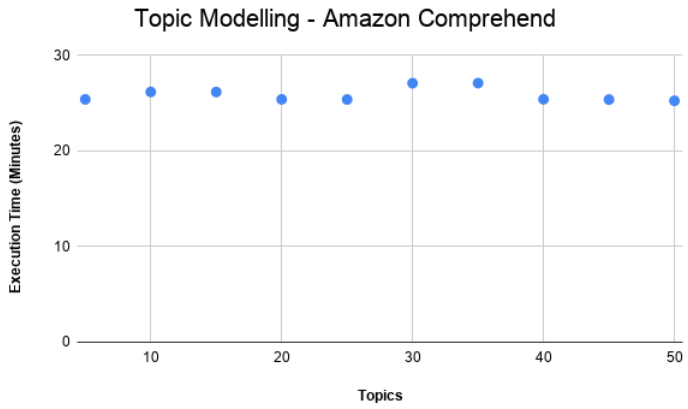

Figure 5.3: A time series representation of incremental number of topic classification.

Sentiment Analysis

In sentiment analysis, the limit of a single file must not exceed 5120 bytes. For this feature, the optimal choice is document per line instead of document per file.

In response, it returns all sentimental scores which are positive, negative, neutral and mixed. The optimal choice is choosing the highest value among them and that is also labelled with a tag of sentiment explicitly in the response. In the document per line format, it is able to detect 5992 sentences. On average, the entire data set is classified as neutral as shown in the below bar chart 5.4.

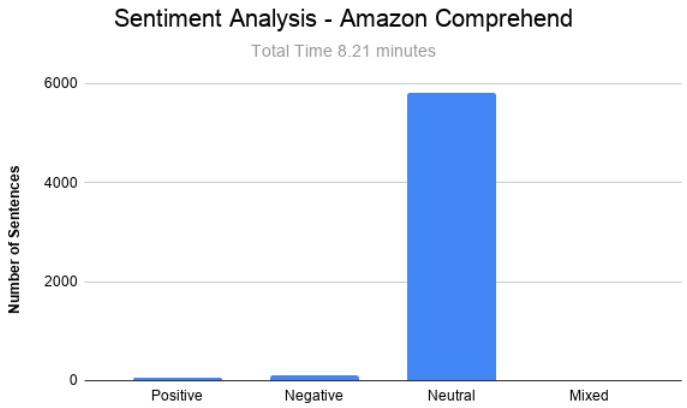

Figure 5.4: A sentimental analysis of 5992 sentences into four categories.

Primary Language

It helps in the detection of the most nominated language in the document. It provides the detection of 67 different languages. In our sample data set, it correctly identifies the nominated language which is English. The response of this feature consists of language label and score which reflects the confidence level of it's detection. In case of fewer terms from other languages, it also returns the label of those languages along with the score. In our case, it detects around 50 languages but the score is quite small such as the English comes up with 0.98 whereas others are in magnitude of 0.003 etc.

Syntax Analysis

It is used for the syntactic parsing of the sentences and returns the corresponding part of speech tag for each word. We can filter out results based on needed tags such as only nouns, adjectives and verbs. This information can be very helpful for the construction of a knowledge graph and helps us to gain a richer understanding of relationships between words in the document. For each tag, it also returns the begin and end offset and confidence score which shows the level of correctness. Based on the need of application, we can define a threshold to filter out the tags with low score. It can detect 17 types of parts of speech. We cannot perform this analysis via Amazon Comprehend console.

5.3 Google Cloud Natural Language API

Google offers Cloud Natural Language API service under the family of AI machine learning products. Using this service, we can extract meaningful information such as people, events, places and sentiments of the textual data [9]. The natural language API uses the same underlying technology which is used by Google Assistant to understand the language. It provides the features of entity detection, sentiments of detected entities, sentiment analysis, syntax analysis and classification of categories. All of these features can be accessed via an API call and it automatically detects the language if it's not mentioned in the request. We can also perform multiple operations in one API call using the method annotateText [9].

Google also provides a cloud based shell editor to run the code for requesting these APIs in our desired supported programming language. We can also run analysis over text via gcloud command in the terminal. The text can be passed as a parameter in the request or for a large number of documents a Google cloud storage URI will be passed in a gcsContentUri field [9].

In sentiment analysis, the API returns score as well as magnitude. The value of score ranges from -1 to 1 covering the gross emotions of the text whereas value of magnitude ranges from 0.0 to +inf indicating the cumulative emotion of each expression. The value of magnitude is directly proportional to the length of the text. It does not mention the particular emotion such as "angry" or "sad" but only negative or positive. In entity analysis, all entities are returned in the highest to lowest order of their salience scores which shows their importance in the whole text.

With default models, Google Cloud Natural Language API returns 13 types of entities which are described in the table below.

Table 5.4: Google Cloud Natural Language API Default Entity Types.

PERSON	LOCATION	ORGANIZATION
EVENT	WORK_OF_ART	CONSUMER_GOOD
PHONE_NUMBER	ADDRESS	DATE
NUMBER	PRICE	OTHER/UNKNOWN

One of the very useful features is categorization of large corpus. It analyzes the document and returns the relevant category based on the text in the document. Google Cloud Natural Language API provides a list of categories from more general to specific context, which is shown below in the Table 5.5. There are 10 categories with each having a large number of subcategories for specific interpretation of the text in the document. These all categories will help us to better understand the domain of the random corpus. This feature is only available for English language version of documents.

Table 5.5: Google Cloud Natural Language API default list of Categories.

Category Name	Total Sub Categories
Arts & Entertainment	61
Autos & Vehicles	21
Beauty & Fitness	19
Books & Literature	6
Business & Industrial	53
Computers & Electronics	35
Finance	18
Food & Drink	21
Games	29
Health	50

Google Cloud Natural Language API supports very few natural languages for its features. The below Table 5.6 provides the list of languages based on each feature.

Table 5.6: Google Cloud Natural Language API languages support for each Feature.

Language	Entity Analysis	Sentimental Analysis	Syntactic Analysis	Entity Sentiment Analysis
Chinese	✓	✓	✓	✗
English	✓	✓	✓	✓
French	✓	✓	✓	✗
German	✓	✓	✓	✗
Italian	✓	✓	✓	✗
Japanese	✓	✓	✓	✓
Korean	✓	✓	✓	✗
Portuguese	✓	✓	✓	✗
Russian	✓	✗	✓	✗
Spanish	✓	✓	✓	✓
Arabic	✗	✓	✗	✗
Dutch	✗	✓	✗	✗
Indonesian	✗	✓	✗	✗
Polish	✗	✓	✗	✗
Thai	✗	✓	✗	✗
Turkish	✗	✓	✗	✗
Vietnamese	✗	✓	✗	✗

Google AutoML Natural Language

For the analysis of domain specific entities and categories in the text, we can build and train our custom models using AutoML natural language service. The training of a model can take up to 3 hours. Google provides an interactive console to annotate our data, build the model and evaluate the model with measurements

of precision and recall [6]. It also provides the feature of building and training a custom model for sentiment analysis. The dataset requires a manual annotation of sentiment numbers for the sentences. We can also train the model for analyzing a document category based on the content in the document.

5.3.1 Pricing Analysis

With Google cloud natural language API we pay for the feature we use without any extra cost. The API charges in terms of units (1000 characters). Initially, 5M characters (5K units) will be analyzed for free for all features except content classification. The categorization of content is a text intensive task hence it can categorize 30M characters (30K units) of text for free. By default natural language API supports a threshold of 20M units of processing per month but in case of requiring more than this limit Google offers to create custom solutions based on the need. Below table summarizes the price of each feature per 1000 units in a month [9].

Table 5.7: Google Cloud Natural Language API Cost per Feature.

Feature	**5K-1M Units**	**1M-5M Units**	**5M-20M Units**
Entity Analysis	$1	$0.5	$0.25
Sentiment Analysis	$1	$0.5	$0.25
Syntax Analysis	$5	$0.25	$0.125
Entity Sentiment Analysis	$2	$1	$0.5

For the classification of content it has three different segment sizes of units as follows.

Content Classification	**30K-250K Units,** $2	**250K-5M Units,** $0.5	**5M+ Units,** $0.1

5.3.2 Performance Analysis

For the performance analysis, We will evaluate each feature of this service on a sample data set of category Marketing from wikipedia, which consists of around 60 documents. For each feature, we will analyze the results and execution time. We are integrating the Google Cloud Natural Language API in Java programming language as the Google cloud shell is not supporting natural language API operations on large corpus. The gradle dependency is used to provide all underlying classes and interfaces. In addition to this, we set an environment variable Google_APPLICATION_CREDENTIALS which points to the location of the

private key file of the Google project and served the purpose of authentication. Our Java based implementation reads input from local directory and writes output in the similar directory. The response is returned in the JSON (JavaScript Object Notation) with different attributes for each feature.

Analyzing Entities

There are a total of 46845 entities detected in the entire text of around 6700 sentences. The response consists of the following attributes. Each entity has a name, entity type and also specify the type of noun which can be a common noun, proper noun or unknown. It returns all the entities in the order of salience score, which describes significance of the entity to the document. It only includes begin offset not end offset in case we specify the encoding type. Besides this it also provides the metadata information about entity knowledge repository which is not useful for our company. Based on the response, the default model does not recognize entities with a high score as all entities are ranked less than 0.5. Based on our assumption the entities of threshold salience score below 0.5 will be discarded or those entities might need human review. This job takes a total execution time of 2.1 minutes.

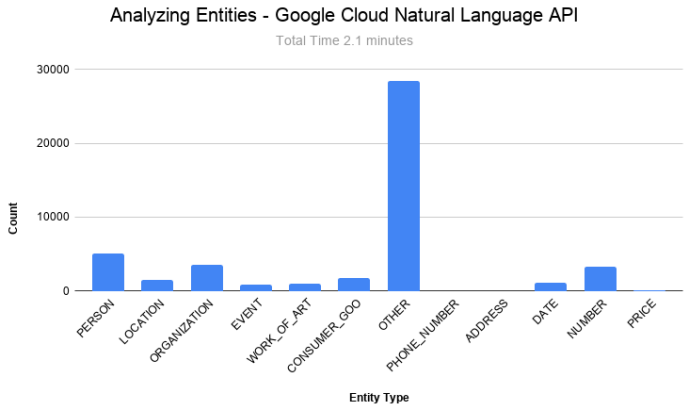

Figure 5.5: Representation of different entity types and their count in the dataset.

Analyzing Sentiments

In sentiment analysis, it determines the sentiment of each sentence as well as of the whole document. The language is determined by the API if not specified in the request. For each document, it returns a document level score between -1 to 1 which shows the overall sentiment and a magnitude which reflects the strength of that emotion ranges from 0 to +inf. An array of sentences is also returned with each sentence having a value of text, begin offset, score and magnitude. It is able to detect 7176 sentences in 3.47 minutes. In the below pie charts, the left one shows

the percentage of document level sentiments where many of them are classified as a neutral. On the right hand side it shows the sentence level sentiments where roughly all sentiments occurred in equal number.

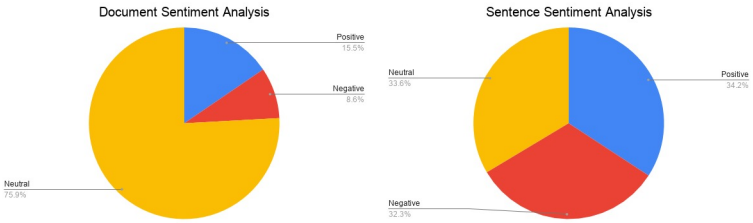

Figure 5.6: Representation of sentimental distribution among documents and sentences into three categories.

Analyzing Entity Sentiment

This feature returns the sentiment of each entity detected in the document. It returns the similar response as of entity analysis and a list of entity mentions with value of score and magnitude for that occurrence of an entity. It also returns the overall sentiment of each entity in the entire document. Entity based sentiments can be a good mechanism to filter out the entities of threshold below or above some sentimental score. It helps us in understanding about the attitude of entities mentioned in the document. There are a total of 42371 entities detected and the below graph shows the overall sentimental distribution of all entities. Most of the entities are classified as neutral whereas on average an equal fraction of entities are positive and negative.

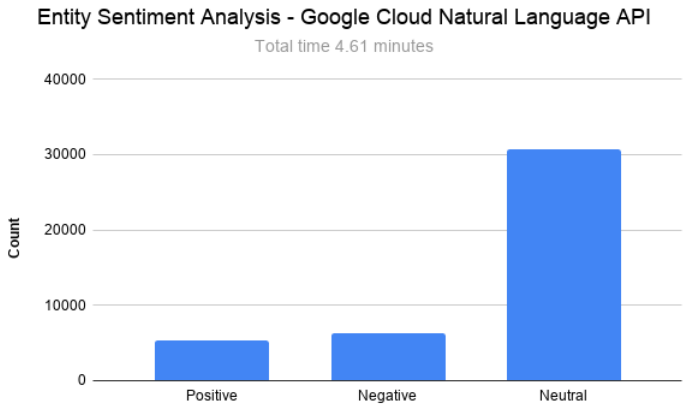

Figure 5.7: Representation of sentimental analysis of entities into three categories.

Classifying Content

It assigns one or multiple categories with a confidence score to a document based on the diversity of the content. There is no such maximum number limit of a category assigned to a document. For very small documents of few lines, it throws an error. For our data set, there are a total 74 categories returned in response. From 74, 41 corresponds to Business and Industrial, 8 relates to People and society and remaining are around different categories. From this classification of documents, we can only pick the categories of our relevance based on the threshold of confidence score returned with each category. For our sample data set, we analyze that 25 categories have confidence scores of above or equal to 0.9 and for suitable results we can ignore the categories below 0.7.

Analyzing Syntax

It inspects the internal structure of the document. It breaks up the input text into sentences and words and provides linguistic information about them using part of speech tagging. In response, it returns a list of tokens which have begin offset, lemma, part of speech tag, tense, voice and many other attributes. It does not provide any score which can help us in filtering the incorrect tags. It annotates all the tokens in our documents in 2.43 minutes. For many tokens the tense is unknown as it is only applicable for the verbs. Based on the need and requirement of the application, we can filter specific types of parts of speech to evaluate the document in a context.

5.4 IBM - Watson Natural Language Understanding

IBM Natural Language Understanding service helps us extract relevant insights from unstructured text at a large scale. It has capability of parsing content from raw HTML (Hypertext Markup Language), public URL and text. On passing a public URL, the service by default cleans random text, advertisements, etc before parsing the content for the better results [28].

It provides the features of detecting entities, categories of document, concepts discussed in the content, emotions of the content which can be sad, happy, etc, recognition of keywords, sentiment analysis, relation between entities, extracting metadata of the document and semantic roles which is parsing sentences into subject-action-object form. All of these features have different parameters which can be set in our request and that makes this service flexible and easy to use. We can access all these features via IBM cloud shell or integrating the Natural Language Understanding toolkit in any one of supported programming languages. It automatically detects the language if not specified in the request parameters [28].

Watson is a super computer from IBM which is capable of processing natural languages and it is trained on out of box machine learning and deep learning models for text extraction and mining to provide better results with high accuracy for given

input. On a free plan, IBM stores customers requested data for the training of models but for premium plans it cannot due to the confidentiality of the data. Using IBM watson knowledge studio, we can create our custom models for the purpose of custom entities detection, sentimental analysis and relation extraction [28].

The features of this API are not available for all supported human languages hence in the below table 5.8 we list down the feature, its supported language and configurable parameters.

Table 5.7: IBM Watson Natural Language Understanding Service Cost per Feature.

Feature	**Languages**	**Parameters**
Concepts	English, French, German, Italian, Japanese, Korean, Portuguese, Spanish	Limit Constraints: value ≤ 50
Emotion	English	Default: true Set this to false if document level emotion is not required.
Entities	English, French, German, Italian, Japanese, Korean, Portuguese, Russian, Spanish, Swedish	Limit Constraint: value ≤ 250 sentiment, emotion, mentions, model Id.
Keywords	English, French, German, Italian, Japanese, Korean, Portuguese, Russian, Spanish, Swedish	Limit Constraint: value ≤ 250 sentiment, emotion.
Relations	Arabic, English, German, Japanese, Korean, Spanish	Custom Model Id to override the default model
SemanticRoles	English, German, Japanese, Korean, Spanish	Limit, keywords, entities
Sentiment	Arabic, English, French, German, Italian, Japanese, Korean, Portuguese, Russian, Spanish	Default: true Set this to false if document level sentiment is not required.
Categories	Arabic, English, French, German, Italian, Japanese, Korean, Portuguese, Spanish	Limit Constraints: value ≤ 10

5.4.1 Pricing Analysis

IBM charges for the text we process without any underlying cost. The API charges in terms of natural language units, NLU (10,000 characters). In the Lite plan, 30K NLU items are processed for free. This plan includes one custom model which can be built through watson knowledge studio. Each NLU consists of the textual characters and number of features applied. In the standard plan, we can process as much NLU as we need in a month. It charges per NLU with the following segment ranges. For the first 250K units it charges $0.003 per NLU item, above this from 250K+ - 5M it charges $0.001 per NLU item and beyond this it charges $0.0002 per NLU. The cost of custom model building via watson knowledge studio is $800 per month. We can upgrade this plan to premium Tier1, which only provides a few extra features such as logging of transactions, high availability and service level agreements on uptime. The training data of models will be stored in an isolated environment on a single machine. The extra cost of this plan is not listed and can only be discussed with them by contact [28].

5.4.2 Performance Analysis

We will evaluate performance of features of this service on our sample data set of 60 documents belonging to the category of Marketing from wikipedia. For each feature, we will analyze the output and execution time. We are integrating this service into the Java programming language using gradle dependency of version 8.5.0. The cloud instance of natural language understanding provides key information for the connectivity such as API key, URL of the service and date of version of its release. We make a request to this service using analyze() method, which takes analyzeOptions as a parameter. This parameter consists of all the feature objects and format of content (text, public URL or HTML), which is text in our case. For applying more than one analysis on the data we can declare multiple features into a single features object. The API returns response in JSON object for each requested feature. Our Java based implementation reads input from local directory and writes output in the similar directory.

Identify Concepts

It returns the high level concept of the document irrespective of frequency of terms. In our case, it truly recognizes the concept as Marketing with a confidence score of 0.9 whereas other terms related to this domain are also returned with a score above 0.5. The number of concepts is a configurable parameter and we set it to 3 in our experiment and it only takes 23 seconds to determine it. We test it for the increased number of concepts but on average the execution time remains the same as it considers more familiar words from that domain with lower confidence score. Hence in ideal cases, top 5 concepts are good for the understanding of the corpus.

Identify Entities

There are a total of 1514 unique entities detected in the entire text of all documents with following categories, 588 person, 349 Company, 130 organization, 214 location, 11 facility, 179 quantity and others. With each entity it mentions the count of that entity occurrence in the document. The response consists of the following properties. It returns a relevance score which specifies its importance in the document and the confidence score which shows the level of correct detection. The total entities are very few in number but the confidence score of 50% entities is 0.9, which shows high accuracy. Almost all of the entities have a confidence score of greater than 0.5. It takes a total execution time of 1.48 minutes.

Identify Keywords

For each document, it returns the unique number of keywords with a count of occurrences for each keyword and a relevance score. In one document, we can determine keywords a maximum number of 250 as per the documentation but any number above this also returns a valid response. It also returns the sentiment and emotions of the keywords but as per the business needs this information can assist us to understand the document in a better way. It takes a total execution time of 0.78 minutes.

Identify Relations

It only identifies the sentences in which two entities are related to each other. In response, it specifies the score, begin and end offset, type of relation among entities and in our case it determines the 2282 number of sentences in which entities are related. We can define a threshold of score 0.5 to omit the relations of entities lower than this value. Furthermore, this information is useful in the extraction of entities which are related with each other. It takes a total execution time of 4.23 minutes.

Identify SemanticRoles

It provides the analysis of the sentence into subject-action-object analysis, which is very useful information for the construction of a knowledge graph. The service automatically ignores the sentences which are not applicable in this format. In our case, it detects a total 2298 sentences in the whole corpus. It takes a total execution time of 1.18 minutes.

Identify Sentiments

This feature provides sentiment of the entire text in a document instead of a sentence. In response, it returns a label and score for the document. In our case, 52 documents are classified with positive labels with a score range of (0.6 - 0.8) and

the remaining 7 are negative but there is no neutral sentiment for any document. It takes a total execution time of 0.31 minutes.

Identify Categories

It returns upto five level taxonomy of categories. The number of categories is a configurable parameter and we set it to 3. For each category, it returns a label of category, score which shows the degree of correctness and explanation which is null for all retrieved categories. More than 50% of the result shows the right category which is Marketing with a score of 0.9. Remaining results were similar keywords from this domain with lesser value of score but none of category has score less than 0.5. It takes a total execution time of 0.37 minutes. With 10 numbers of categories it adds further terms of this domain with less score but takes the same execution time.

5.5 Comparison of Cloud based Natural Language Processing Services

The most prominent cloud based natural language processing service providers are Amazon, Google and IBM. The use case for our company is the parsing of large textual documents in order to understand the human behaviour about the products and services of our clients. We will compare features, performance, cost and quality of results from each service with different perspectives and will draw the outcome based on our needs. Our analysis is based on the default models of these services. In the below Table 5.8, we briefly summarize our findings and analysis for these services.

Table 5.8: Advantages and Disadvantages of these Services.

Advantages	**Disadvantages**
IBM provides more features.	Amazon and Google offer almost same features but less than IBM.
IBM and Amazon costs the same.	Google is expensive than IBM and Amazon.
Amazon offers language detection feature explicitly.	IBM and Google detects language by default with any mentioned feature but not explicitly.
IBM is more robust, fast and flexible.	Amazon and Google are slower than IBM and not configurable.
Google and IBM has defined categories for corpus classification.	Amazon classifies corpus based on frequency of words.
Amazon offers extra service for parsing features of health related data.	IBM and Google does not have this feature yet.

In the below Matrix 5.1, we present the ranking of all services in a visualization manner based on our analysis in terms of symbols. The high number of star presents high quality for given column and vice versa. Following the matrix below, we describe comparison of these services in a detail manner.

Matrix 5.1: Metrics analysis of all services based on the following parameters.

	Features	**Cost**	**Performance**	**Result**	**Flexibility**
IBM	*****	$	*****	*****	***
Amazon	***	$	**	***	**
Google	***	$$$	***	**	**

Natural language processing services take text as an input but that text can be in multiple file formats such as txt, pdf, html, ppt, web pages, etc. Amazon, Google both support only raw text whereas IBM supports text as well as html and web pages. Hence considering availability of multiple options of file formats, we will choose IBM.

For entity detection, Google API recognizes 6 times more entities than Amazon but all have very low confidence scores and both of them find repeated entities separately whereas IBM only returns unique entities along with count of each individual entity. In total, IBM returns 4 times less entities than Amazon but all have confidence scores greater than 0.5 which shows high accuracy of default models. Google returns all entities in the descending order of a salience score whereas Amazon and IBM does not. In Amazon and IBM, retrieving entities of top score will add an extra overhead of sorting the list, which has a time complexity of nlogn. In our case, we assume that the threshold score of the entity below 0.5 will be discarded, hence Google API results will be of no use. With respect to time, IBM has the fastest response time among all. With the metric of time and quality, we found that entity detection results from IBM are very useful as compared to Amazon and Google.

Google provides an explicit feature of analyzing sentiments of entities but similar functionality can be achieved in IBM by setting a parameter of sentiment to true in the identify entities request. IBM also provides the way of determining the emotions of the entities in the same way but Google does not have this capability yet. Amazon neither provides sentiment of entities nor emotions.

For keyphrases detection, Google API does not provide this feature. Amazon returns keyphrases in a very large number but it also considers a high magnitude of single words as phrases which are not of interest in case of multi-word expressions. IBM gives us the flexibility of the configuration of the number of keywords to determine in the document whereas Amazon does not. The confidence score of keywords from IBM is greater than from Amazon and it also takes less amount of execution time. With IBM we can discover the emotions and sentiments of keyphrases but not with Amazon. Based on the choice of parameters, time and quality

of results we will choose IBM for the purpose of this analysis.

For Sentiment analysis, Amazon returns sentiments of individual sentences rather than of the whole document. On the contrary IBM returns sentiments of the whole document instead of sentences. Google only returns sentiments at both levels. Amazon does not support text with size greater than 5120 bytes and there is no such limit in Google and IBM. The document level sentiments from Google and IBM looks opposite, as Google recognized many documents as neutral whereas IBM marks them positive. In terms of speed, IBM is 10 times faster than Google. Amazon and Google give output of sentimental score for each sentiment (positive, negative, neutral and mixed) whereas IBM only gives one label associated with corresponding score. The intent of text is not known hence we cannot make any decision based on sentimental results instead based on time IBM stands out.

For detecting the language of the document, Google and IBM do not provide any explicit feature for detecting it but with each request it automatically detects the language of the document. On the other hand, Amazon provides a feature of language detection separately with support of 67 languages. It can also recognize multiple languages in one document. Hence, Amazon is the optimal choice as it performs better in language detection as compared to Google and IBM.

For the classification of documents into categories or topics, Google and IBM have predefined general to specific categories whereas Amazon does not. Amazon and IBM take an input the number of topics to be assigned to each document and in output each category is associated with a confidence score. Amazon and Google discover the relevant categories for our corpus with a low relevance score as compared to IBM, which determines the exact same category with a very high confidence score of 0.9. IBM also provides a similar feature of analyzing the concept of the document based on the content instead of frequency of words and it will be beneficial to discover the concept and category from the same service. The result of IBM is better than Amazon and Google in terms of quality and based on another similar dedicated feature of concept hence we will select IBM for this feature.

Apart from these common features in all services, IBM provides capability of detecting sentences which follows the structure of subject-action-object which is known as semanticRoles. It also offers the ability to find the sentences where two entities are related to each other. Such pieces of information are very useful for building the knowledge graphs and assist to relate multiple documents with each other. Each feature of IBM provides the configuration of parameters which can be defined based on the need of the application while Amazon and Google does not have this flexibility. We can set such parameters to false when not needed and in this way we can save our extra computational time.

The primary criteria of the selection of natural language processing service is

the quality of output and performance and secondary will be cost. In our case, we find IBM the most useful service based on the performance, quality of output, features and flexibility of configuring the parameters. From the perspective of cost, all services charge based on the number of characters without any upfront costs. For 1M characters, Amazon and IBM charges $0.33 and Google does $1 which shows that Google is three times more expensive than Amazon. Amazon and IBM charge almost the same for the text processing but IBM provided better results. Hence this is another reason for choosing IBM.

5.6 Summary

In this project, we inspect and analyze three cloud based natural language processing services from the most renowned cloud providers which are Amazon, Google and IBM. The performance analysis of individual service is evaluated based on a common data set of wikipedia articles from the category of Marketing. Each of this service offers essentially similar common features but with different request and response parameters.

The first three parts of this chapter briefly described the features of each service along with performance analysis on the sample data set and its pricing. From the perspective of easy to understand, Amazon has very detailed documentation, IBM has access to all information in one place and Google has a poor structure of the content. We found that Amazon has dedicated service for the processing of health data which Google and IBM do not have yet.

In the comparison Section 5.5, we compared all features of services with each other. Based on the results we find that IBM performs better than Amazon and Google. In terms of cost, IBM and Amazon are three times cheaper than Google. IBM offers many extra features highlighted in Section 5.4.2 which can be very useful and supportive for the understanding of text. IBM is also comparatively faster than Google and Amazon. In a nutshell, we prefer IBM Natural Language Understanding service for the textual processing. With the help of cloud based natural language processing service, our company will not only save a significant cost, time and effort of the engineering team but the complexity of maintenance is also reduced with better and quick results.

Chapter 6

Conclusions and Future Work

The growing amount of textual data from different sources imposes new challenges for its processing in an appropriate amount of time. Data processing pipelines are designed for the purpose of its handling in an efficient way. The process of extracting meaningful insights from unstructured data will help companies to understand their potential market and customer needs. We have a very large amount of data from different clients in the form of PDF and PPT documents. In this ongoing research and development project, we are analyzing different libraries and technologies to develop a data processing pipeline for the construction of knowledge graphs. Knowledge graphs are widely used as a means of discovering information quickly and easily.

The processing of PDF and PPT documents has various challenges of parsing the textual data due to its complex structure. This format of document is filled with floating text instead of specific tags and sections. In order to extract insights from this raw text, we need to process natural languages which is achieved with the help of natural language processing techniques. We design our approach to this data processing pipeline in the first research question. In modern software development, there are many cloud based services available to overcome the challenges of in house development and maintenance. Natural language processing is the core component in our pipeline and designing the core models of machine learning requires extra effort from engineering perspective. As a part of the second research question we analyzed the cloud based natural language processing services, which can help us to analyze our text in an easy and optimal way.

In this chapter, we precisely summarize our work and present remarks about research questions which are explained in a very detail in Chapter 4 and Chapter 5. We also highlight the possible future work which can be accomplished on this existing work.

6.1 Conclusion

The section of conclusion highlights our essential findings and results gathered from the implementation and experimental analysis of our research questions.

RQ1: How to design and implement a data processing pipeline for generating knowledge graphs?

We proposed a design, architecture and implementation of a data processing pipeline for the construction of knowledge graphs. Our pipeline consists of three stages and two data stores, Mongo database and Neo4j. The pipeline begins with the extraction and parsing of documents of type PDF and PPT. In our research, we design our custom implementation by integrating two different libraries for the parsing of documents. One is Apache PDFBox which is free and open source and second one is Aspose.PDF which is a commercial library and supports the parsing of both formats of document, PDF and PPT. The following extracted text in the form of sentences and paragraphs from this stage is stored in the Mongo database.

With the help of experiments we evaluated the performance of parsers and found that Apache PDFBox is approximately 10 times faster than Aspose.PDF for large data sets. The second stage of this pipeline applies natural language processing techniques on this raw text and we observed that it is a major component of this pipeline. Our implementation of this component is using Stanford CoreNLP library which provides basic building blocks for the analysis of entity recognition, relation extraction, semantic analysis and part of speech tagging with the help of default models. For the processing of very large files, we proposed two different solutions for the problem of out of memory exception. Either we can perform different annotators in one service by chunking large amounts of text into several batches or each annotator can run as a different instance of service in a parallel fashion. In our implementation, we are running all annotators in one service in a single go. The results of this service are mapped into a graph data model and stored in the Neo4j graph database.

The graph data model is composed of nodes and relationships with attributes defining their characteristics. The knowledge graph service is the last stage of this pipeline. This service is generating insights and discovers different patterns in the form of graph structured data stored in Neo4j and answers user queries in the form of interactive graphs. The Neo4j database is integrated in a spring boot application using different drivers and package managers. The flow of data is shown in a architecture diagram of Figure 4.4. As a part of this research question, we evaluate our design using a sample data set of category Marketing from wikipedia. Our proposed architectural design and implementation of the data processing pipeline integrates already existing components in such a way which has not been done previously to support the construction of knowledge graph from unstructured text. We

found that, this representation of information will enable customers to understand the relation in the data and improve their quality based on data driven businesses.

RQ2: How to choose the most suitable cloud based natural language processing service based on cost, performance analysis along with features and flexibility of individual service with other cloud providers?

In our data processing pipeline, we found that natural language processing plays a key role in discovering the information around the raw text. It is observed that with the evolution of technology, cloud computing provides alternate solutions to improve the process of development and we are only charged as per use without any hidden cost. As a part of this project, we did our research and experiment on three cloud based natural language processing services from well known cloud providers which are Amazon, Google and IBM. We deeply analyzed each service from different perspectives such as features, cost, quality of output, performance and ease of use. We evaluated all services based on a sample data set of category Marketing from wikipedia.

The above mentioned three services are tested based on the default models. In consideration of features, we found that Amazon and Google offer almost similar features with small variants in it. Besides that, we also observed that IBM provides some extra features beyond Amazon and Google, which gives some extra bits of information out of text. We learned that results of Google Cloud Natural Language API are not desirable as compared to Amazon and IBM. On the other hand, we noticed that Google charges three times more than Amazon and IBM. Amazon and IBM charge almost the same cost for features but from experimental evaluation we found that IBM produced better results. The classification of large corpus into relevant categories is a very powerful feature available in these three services but we discovered that only Google and IBM have predefined categories whereas Amazon ranks the document based on the frequency and relevancy of words then sorts the topics based on a weight.

We also found that each service has different response parameters for similar features. From the perspective of setting custom values in request parameters, IBM is very flexible in the configuration of input parameters whereas there are no such input parameters in Amazon and Google. The custom parameters for any request enable us to retrieve only what we need except extra information. Moreover, we noticed that IBM produced useful results and comparatively faster than Amazon and Google. With the use of cloud based natural language processing services we can reduce the development time and complexity of engineering efforts with a little cost, as per our use.

6.2 Future Work

In this project, we design our own data processing pipeline for the construction of knowledge graphs using unstructured data. In addition to this, we also perform experimental evaluation of cloud based natural language processing services. However, due to the limited amount of time for this thesis, we have some suggestions to extend this work further in future. Our suggestions for future work are summarized as follows.

- To enrich more data sources into this pipeline, we can integrate speech-to-text services for further analysis of data. Furthermore, the pipeline is required to test more data sets from multiple categories as we evaluate our implementation based on one category.
- In cloud based natural language processing services, only text files are supported. The support for other formats of documents is required mainly, PDF and PPT. Apart from this, to achieve better results for Amazon and Google the optional configuration of input parameters is also needed.
- For ensuring high availability of the services on heavy load, this pipeline is required to be deployed in the form of a cluster of running multiple instances of each service. On top of this, any orchestration tool can help to achieve this task, such as Kubernetes. The automation of pipeline stages can be automated using any pipeline automation tools such as Airflow.

Bibliography

[1] 3 ways to build an etl process with examples — panoply. https://panoply.io/data-warehouse-guide/3-ways-to-build-an-etl-process/. (Accessed on 08/14/2020).

[2] About - company - aspose.com. https://company.aspose.com/. (Accessed on 08/14/2020).

[3] Amazon comprehend - natural language processing (nlp) and machine learning (ml). https://aws.amazon.com/comprehend/. (Accessed on 08/14/2020).

[4] Amazon comprehend medical. https://aws.amazon.com/comprehend/medical/. (Accessed on 08/14/2020).

[5] Apache pdfbox — a java pdf library. https://pdfbox.apache.org/. (Accessed on 08/14/2020).

[6] Automl natural language documentation. https://cloud.google.com/natural-language/automl/docs. (Accessed on 08/14/2020).

[7] A beginner's guide to the data science pipeline — by randy lao — towards data science. https://towardsdatascience.com/a-beginners-guide-to-the-data-science-pipeline-a4904b2d8ad3. (Accessed on 08/14/2020).

[8] Category:marketing - wikipedia. https://en.wikipedia.org/wiki/Category%3AMarketing. (Accessed on 08/14/2020).

[9] Cloud natural language — google cloud. https://cloud.google.com/natural-language. (Accessed on 08/14/2020).

[10] Data pipeline architecture. https://www.snaplogic.com/glossary/data-pipeline-architecture. (Accessed on 08/13/2020).

[11] Difference between structured, semi-structured and unstructured data - geeksforgeeks. https://www.geeksforgeeks.org/difference-between-structured-semi-structured-and-unstructured-data/. (Accessed on 08/14/2020).

[12] Etl database - a guide to etl/elt for data engineers and data analysts. https://www.stitchdata.com/etldatabase/. (Accessed on 08/14/2020).

[13] Etl (extract, transform, and load) process. https://www.guru99.com/etl-extract-load-process.html. (Accessed on 08/14/2020).

[14] Getting started: the 3 stages of data infrastructure — by nate kupp — medium. https://medium.com/@natekupp/getting-started-the-3-stages-of-data-infrastructure-556dac82e825. (Accessed on 08/14/2020).

[15] Github - dstlry/dstlr: scalable knowledge graph construction from unstructured text. https://github.com/dstlry/dstlr. (Accessed on 08/13/2020).

[16] Github - neo4j/neo4j-ogm: Java object-graph mapping library for neo4j. https://github.com/neo4j/neo4j-ogm. (Accessed on 08/14/2020).

[17] Homepage — yahoo answers. https://answers.yahoo.com/. (Accessed on 08/14/2020).
[18] Introduction to cloud computing. - dev. https://dev.to/daviddennis02/introduction-to-cloud-computing-5cg. (Accessed on 08/14/2020).
[19] Introduction to mongodb — mongodb manual. https://docs.mongodb.com/manual/introduction/. (Accessed on 08/14/2020).
[20] An introduction to natural language processing (nlp) — by odsc - open data science — medium. https://medium.com/@ODSC/an-introduction-to-natural-language-processing-nlp-8e476d9f5f59. (Accessed on 08/13/2020).
[21] Neo4j graph platform – the leader in graph databases. https://neo4j.com/. (Accessed on 08/14/2020).
[22] Overview - corenlp. https://stanfordnlp.github.io/CoreNLP/. (Accessed on 08/14/2020).
[23] Pipeline (computing) - wikipedia. https://en.wikipedia.org/wiki/Pipeline_(computing). (Accessed on 08/13/2020).
[24] Quora - quora, where you can exchange knowledge and understand the world better. https://de.quora.com/. (Accessed on 08/13/2020).
[25] A simple introduction to natural language processing — by dr. michael j. garbade — becoming human: Artificial intelligence magazine. https://becominghuman.ai/a-simple-introduction-to-natural-language-processing-ea66a1747b32. (Accessed on 08/13/2020).
[26] Spring into neo4j with spring data 5, spring boot 2, and neo4j! — by jennifer reif — neo4j developer blog — medium. https://medium.com/neo4j/spring-into-neo4j-with-spring-data-5-spring-boot-2-and-neo4j-3962fb1ea067. (Accessed on 08/14/2020).
[27] Stack overflow - where developers learn, share, & build careers. https://stackoverflow.com/. (Accessed on 08/13/2020).
[28] Watson natural language understanding - overview — ibm. https://www.ibm.com/cloud/watson-natural-language-understanding. (Accessed on 08/14/2020).
[29] What is a graph database? - neo4j graph database platform. https://neo4j.com/developer/graph-database/. (Accessed on 08/13/2020).
[30] When to use (and not to use) mongodb - dzone database. https://dzone.com/articles/why-mongodb. (Accessed on 08/14/2020).
[31] Wikimedia foundation. https://wikimediafoundation.org/. (Accessed on 08/14/2020).
[32] Wikipedia. https://www.wikipedia.org/. (Accessed on 08/14/2020).
[33] Wikipedia-api · pypi. https://pypi.org/project/Wikipedia-API/. (Accessed on 08/14/2020).
[34] Wtf is a knowledge graph? — hacker noon. https://hackernoon.com/wtf-is-a-knowledge-graph-a16603a1a25f. (Accessed on 08/13/2020).
[35] Oumayma Chergui, Ahlame Begdouri, and Dominique Groux-Leclet. A knowledge-based approach for keywords modeling into a semantic graph. *International Journal of Information Science and Technology*, 2(1):12–24, 2018.
[36] Ryan Clancy, Ihab F Ilyas, and Jimmy Lin. Scalable knowledge graph construction from text collections. In *Proceedings of the Second Workshop on Fact Extraction and VERification (FEVER)*, pages 39–46, 2019.
[37] Lisa Ehrlinger and Wolfram Wöß. Towards a definition of knowledge graphs. *SEMANTiCS (Posters, Demos, SuCCESS)*, 48:1–4, 2016.

[38] Trey Grainger, Khalifeh AlJadda, Mohammed Korayem, and Andries Smith. The semantic knowledge graph: A compact, auto-generated model for real-time traversal and ranking of any relationship within a domain. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 420–429. IEEE, 2016.

[39] Natthawut Kertkeidkachorn and Ryutaro Ichise. T2kg: An end-to-end system for creating knowledge graph from unstructured text. In *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[40] Kundan Kumar and Siddhant Manocha. Constructing knowledge graph from unstructured text. *Self*, 3:4, 2015.

[41] Bhavana Dalvi Mishra, Niket Tandon, and Peter Clark. Domain-targeted, high precision knowledge extraction. *Transactions of the Association for Computational Linguistics*, 5:233–246, 2017.