1 Problem Statement

A substantial amount of valuable knowledge is recorded in the form of unstructured text data, such as news, emails, journal articles, etc. Information extraction deals with automatically extracting structured data from unstructured and semi-structured data sources. Within information extraction, the task of identifying semantic relations among the text entities is referred to as *relation extraction.* The goal of this project was to build a system to extract information templates of the following forms:

- BORN (*Person/Organization*, *Date*, *Location*)
- ACQUIRE (*Organization*, *Organization*, *Date*)
- PART_OF
  - PART_OF (*Organization*, *Organization*)
  - PART_OF (*Location*, *Location*)

To build this system, we were provided with 30 text files containing Wikipedia articles that are split as follows:

- 10 articles related to Organizations
- 10 articles related to Persons
- 10 articles related to Locations

2 Proposed Approach

The general outline of our project can be broken down into three main stages:

1. *Feature Extraction.*
2. *Preliminary filtering.*
3. *Relation extraction.*

Language in general, is complex. Words can be used in different senses, and their order of appearance can mean completely different things in similar-looking scenarios. Thus, using linguistic knowledge associated with the text while building NLP systems is considered to be extremely useful. Identifying key information associated with the words in the text such as their parts of speech, the underlying phrase-structure rules, entity chunking etc. is paramount. We used each of these components to enrich the features extracted from the Wikipedia articles.

Before extracting the relations from the articles, a basic filtering was done to weed out the sentences/phrases with little meaning. In general, a complete sentence is considered to be one which contains atleast one subject, one predicate, one object, and closes with a punctuation. Considering subject and object are always nouns and the predicate is always a verb, the text files were subjected to a parts-of-speech based filtering, and all those sentences which didn't fit this description were filtered out.

There are various pattern-based and machine learning based approaches to extracting relations from text. Considering the size of the corpus given to us wasn't very large, we decided to explore a rule-based approach making use of pattern-extraction algorithms for this task. Different strategies were devised for each of the templates. Finally, considering the given templates were specific with respect to the labels of the named entities they were associated with, it was used to filter the extracted templates.

3 System Design

3.1 Architecture

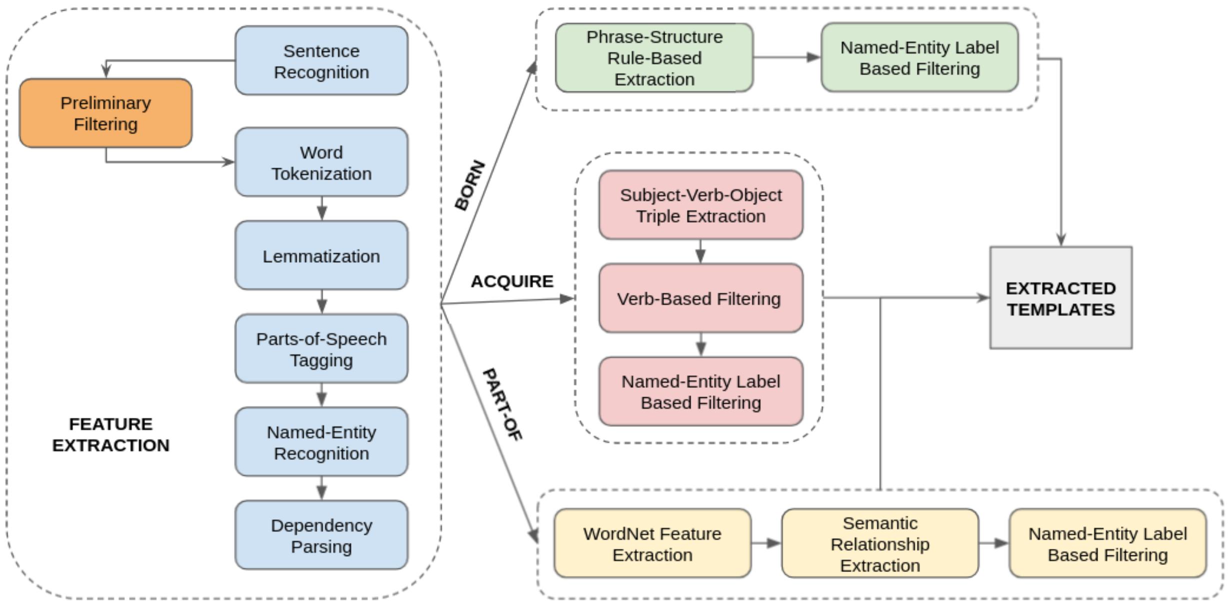

Sentence Recognition
Preliminary Filtering
Word Tokenization
Lemmatization
Parts-of-Speech Tagging
Named-Entity Recognition
Dependency Parsing
FEATURE EXTRACTION
BORN
ACQUIRE
PART-OF
Phrase-Structure Rule-Based Extraction
Named-Entity Label Based Filtering
Subject-Verb-Object Triple Extraction
Verb-Based Filtering
Named-Entity Label Based Filtering
EXTRACTED TEMPLATES
WordNet Feature Extraction
Semantic Relationship Extraction
Named-Entity Label Based Filtering

Figure 1: Architectural Design of the Pipeline

3.2 General Overview

The `chomskIE` system is designed to be as modular as possible. While the design of our system is inspired by the `spaCy` library, individual components can be swapped out trivially to use various other libraries. In order to enable this, we defined a `Document` data structure similar to the `Doc` object in `spaCy`. With each feature/relation extraction step in the pipeline, this document gets appended with the information extracted from the step, which can be used later in the pipeline.

3.3 External Packages/Libraries Used

This project was developed with `Python3` programming language. To do the various tasks within the software, external packages like `spacy`, `ftfy`, `textacy` and `datefinder` were used.

4 Full Implementation Details

4.1 Feature Engineering

The feature engineering stage has two phases:

1. Read the raw text in the correct encoding.
2. Extract low-level and high-level features using a deep Natural Language Processing Pipeline.

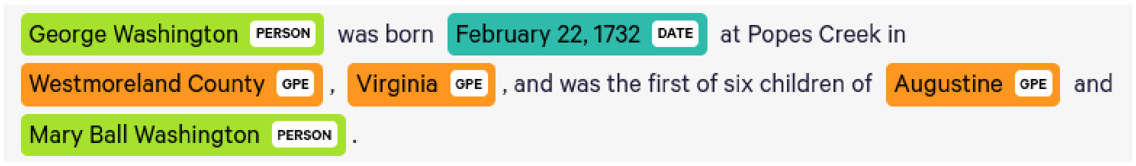
While most of given text files are correctly encoded as Unicode text, some of the files have encoding issues. To handle this and avoid glitches due to incorrectly encoded text, `ftfy` was used. Our system automatically fixes the encoding issues after reading the data from the files, and can be extended to various input sources.

The deep Natural Language Processing Pipeline consists of the following stages:

1. `SentenceRecognizer` uses rules and patterns to detect sentence boundaries.
2. `WordTokenizer` split sentences into individual tokens in a non-destructive fashion. It also allows us to merge named entities into a single token.
3. `Lemmatizer` maps tokens to their lemmas using pre-written rules.
4. `PartOfSpeechTagger` uses a pre-trained statistical model to assign part of speech tags to the tokens

5. `NamedEntityRecognizer` uses a pre-trained statistical model identify non-overlapping spans in the sentence that contains Named Entities.

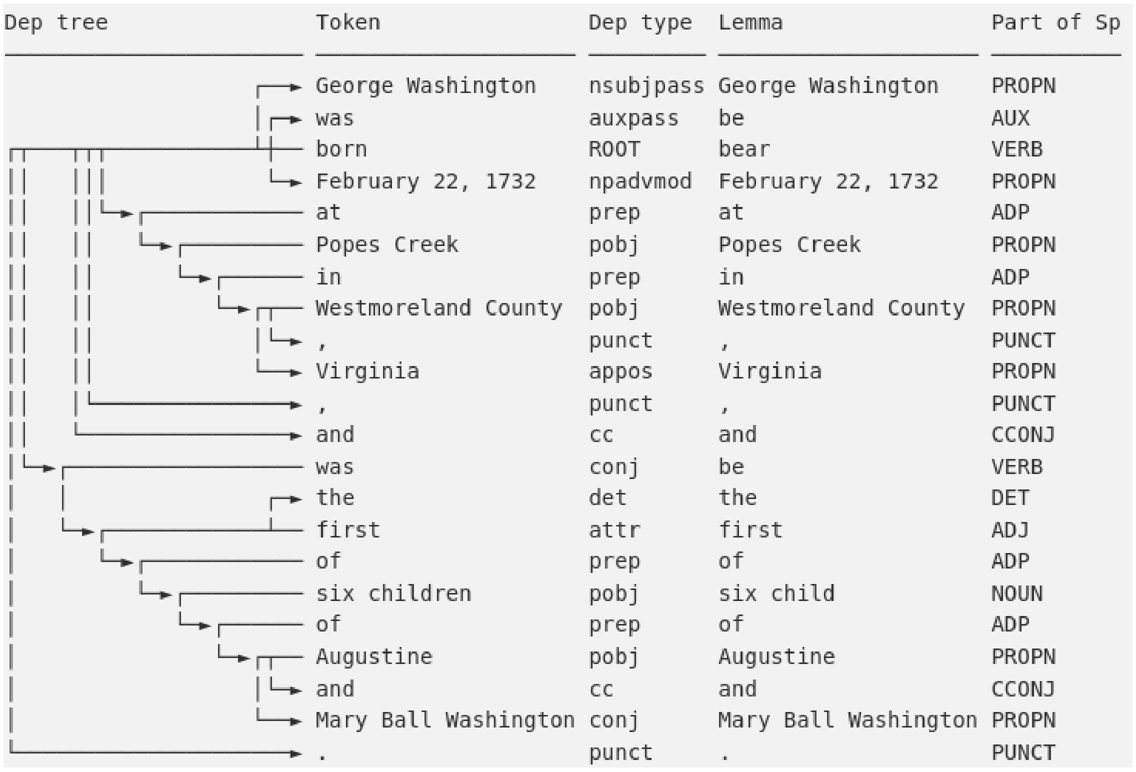
6. `DependencyParser` uses a pre-trained statistical model to parse the sentence into a dependency tree.



George Washington PERSON was born February 22, 1732 DATE at Popes Creek in Westmoreland County GPE , Virginia GPE , and was the first of six children of Augustine GPE and Mary Ball Washington PERSON .

Figure 2: The named entities detected in a sentence

Figure 2 shows the named entities detected by our pipeline in an example sentence. Figure 3 depicts the dependency parse tree for the sentence which was re-tokenized to keep the named entities as single tokens. In the figure, the leaves of the tree also contain other NLP features like lemmas and Part of Speech tags for the tokens.



```
Dep tree                 Token                Dep type  Lemma                Part of Sp
───────────────────────  ───────────────────  ────────  ───────────────────  ──────────
                    ┌──► George Washington    nsubjpass George Washington    PROPN
                    │┌─► was                  auxpass   be                   AUX
┌┬────┬┬┬───────────┴┴── born                 ROOT      bear                 VERB
││    │││            └─► February 22, 1732    npadvmod  February 22, 1732    PROPN
││    ││└─►┌────────────  at                  prep      at                   ADP
││    ││   └─►┌─────────  Popes Creek         pobj      Popes Creek          PROPN
││    ││      └─►┌──────  in                  prep      in                   ADP
││    ││         └─►┌┬──  Westmoreland County pobj      Westmoreland County  PROPN
││    ││            │└─►  ,                   punct     ,                    PUNCT
││    ││            └──►  Virginia            appos     Virginia             PROPN
││    │└──────────────►   ,                   punct     ,                    PUNCT
││    └───────────────►   and                 cc        and                  CCONJ
│└─►┌─────────────────    was                 conj      be                   VERB
│   │              ┌─►    the                 det       the                  DET
│   └─►┌───────────┴──    first               attr      first                ADJ
│      └─►┌───────────    of                  prep      of                   ADP
│         └─►┌────────    six children        pobj      six child            NOUN
│            └─►┌─────    of                  prep      of                   ADP
│               └─►┌┬─    Augustine           pobj      Augustine            PROPN
│                  │└─►   and                 cc        and                  CCONJ
│                  └──►   Mary Ball Washington conj     Mary Ball Washington PROPN
└─────────────────────►   .                   punct     .                    PUNCT
```

Figure 3: Dependency parse tree along with lemmas and part of speech tags for the tokens in a sentence.

4.2 Preliminary filtering

Since the given data files are the text form of Wikipedia articles, they contain some non-sentence artifacts like:

- Headings and subheadings
- Ordered and unordered lists
- References

We pre-process the text by splitting the text into paragraphs (via newlines `"\n\n"`). `spaCy` by default uses a dependency-tree based sentence boundary detection. Considering the Wikipedia articles were generally well-structured grammatically, we realized this was effective in identifying different sentences in the paragraphs within the text.

Before subjecting each of these sentences to the feature-extraction pipeline, we determine if the sentence is complete by checking if it has at least one subject-predicate pair (via part of speech tags). Such a technique helped filter the non-sentence artifacts.

4.3 Relation Extraction

Our general approach to relation extraction was using dependency tree based rules.

4.3.1 Template #1: BORN

There are two kinds BUY relations we wanted to extract:

1. BORN(Person, Date, Location)
2. BORN(Organization, Date, Location)

The anchor for the first BORN template is just the word `"born"`. This form of relation occurs most commonly in passive voice like ”x was born in y on z.” So, we used the dependency relation *nsubjpass* to determine the person.

The anchors for the second BORN template are the words {`"found", "establish"` }. This form of relation occurs most commonly in active voice like ”x founded y in z on w.” So, we used the dependency relation *dobj* to determine the founded organization

The Date and Location arguments are extracted by matching tokens whose head has the Part of Speech tag *IN* (as in ”x founded y *in* date”).

Finally, the extracted tuples are validated by ensuring that the entity types of the arguments are correct (Eg, `GPE` for Location)

4.3.2 Template #2: ACQUIRE

For every sentence in the document, first ordered subject-verb-object triples (SVOTriples) were extracted based on a rule-based pattern matching algorithm designed around the phrase-structure dependencies. `textacy` library was used for extracting such triples.

A set of verbs was constructed containing synonyms of the verb *acquire*. All the subject-verb-object triples extracted for a sentence which didn't contain verbs or their lemmas from this list were considered to be irrelevant and filtered out.

Next, a named-entity label based filtering was carried out. All subject-verb-object triples the named-entity label of whose subject was `ORG` were retained. In order to find the third argument in the template, those words were picked from the dependents of the root verb which also carried an `ORG` named-entity label.

One special case in this template was the extraction of date objects. We realized that the dependency parse structure generated by `spaCy` doesn't take into consideration the dependency relation between the root and date units within the sentence. Also, the named-entity based `DATE` labeling wasn't always accurate. So, a regular expression based matching algorithm was used to identify various formats of dates or references to dates in the sentence which fit with the template.

4.3.3 Template #3: PART_OF

The strategy employed for extracting PART_OF templates was devised around identifying the meronymy semantic relations between different words in the sentence. In order to enable this, first a set of synonyms was constructed for all the words in the sentence using Wordnet data resource. Next, sets of meronyms and holonyms were retrieved for each of these words.

A set intersection between the meronyms and the original list of tokens in the sentence was computed. A similar set intersection was computed between the holonyms and the list of tokens. This helped identify the relevant tokens in the sentence that exhibited the meronymy semantic relation.

From both these sets, the meronyms were mapped with their corresponding holonyms to generate the PART_OF templates. Considering the template arguments had a restriction to either be a `ORG-ORG` pair or a `LOC-LOC` pair, this was incorporated for further filtering out irrelevant relations, and retrieve the relevant PART_OF templates.

5 Results and Error Analysis

While using evaluation metrics like precision and recall was intuitive, we found that it wasn't very feasible in this case because of the phrase-matching involved in comparing the arguments of the extracted relations. Thus, to test the efficacy of the pipeline, a small `dev` dataset was constructed with selected examples from the given corpus. The entire pipeline was developed with the goal of improving the quality and quantity of the relations extracted from this dataset. Here are a few sample relations extracted by the pipeline [1][2]:

1. BORN

   - `Abraham Lincoln was born on February 12, 1809, as the second child of Thomas and Nancy Hanks Lincoln, in a one-room log cabin on Sinking Spring Farm near Hodgenville, Kentucky.`

     ```
     Argument-1: Abraham Lincoln
     Argument-2: February 12, 1809
     Argument-3: Hodgenville
     ```

   - `In May 2002, Musk founded SpaceX, an aerospace manufacturer and space transport services company, of which he is CEO and lead designer.`

     ```
     Argument-1: SpaceX
     Argument-2: May 2002
     ```

2. ACQUIRE

   - `Compaq acquired Zip2 for US$307 million in cash and US$34 million in stock options in February 1999.`

     ```
     Argument-1: Compaq
     Argument-2: Zip2
     Argument-3: February 1999
     ```

   - `In 2015, Tesla acquired Riviera Tool & Die (with 100 employees in Michigan), one of its suppliers of stamping items.`

     ```
     Argument-1: Tesla
     Argument-2: Riviera Tool & Die
     Argument-3: 2015
     ```

[1] These examples are picked from the /outputs directory in the submitted code repository, which contains JSON files for all .txt files in corpus.

[2] https://www.github.com/aashishyadavally/chomskIE

3. PART-OF

   - `They met in Springfield, Illinois in December 1839 and were engaged a year later.`

     ```
     Argument-1: Springfield
     Argument-2: Illinois
     ```

   - `The Mahatma Gandhi District in Houston, Texas, United States, an ethnic Indian enclave, is officially named after Gandhi.`

     ```
     Argument-1: Houston
     Argument-2: Texas
     ```

In general, we realized that the default spaCy named-entity recognition wasn't as effective as we would have liked. For example, in Figure 2, 'Augustine' was identified as `GPE`, while it should have been labeled as `PERSON`. Considering filtering based on named entity labels is a major part of our approach due to the restricted nature of the relations, errors like these significantly hinder the performance of the software.

Using WordNet for identifying words based on their semantic relationship was a central part of our strategy for extracting PART_OF templates. However, we realized that WordNet doesn't encompass as many words/relations specific to our corpus, and thus, our ability to find these templates effectively was hindered.

6 Conclusion

Lessons Learned & Potential Improvements

The premise of the pipeline for extracting ACQUIRE templates relies on the effective retrieval of subject-verb-object triples from the text. As we realized, the pipeline wasn't very effective in identifying all such triples in complex sentences, i.e. those containing more than one dependent clauses. As a result, there were cases where such templates weren't identified by the software. This can be a potential improvement, and more effective subject-verb-object triple extraction can be used to improve the overall efficacy of the system.

Another improvement for the relation extraction algorithms that we would like to explore in the future is ensemble methods. For example, the Snorkel system provides ways to combine various heuristic rules in a semi-supervised manner. This approach would have been viable for this project if we had more data.