Project Report: H-AT

Authors

- Dat Quoc Ngo, dqn170000
- Hepson Sanchez, hms160230

Problem Description

Text is an unstructured data type that contains an enormous amount of information and poses challenges of quick information retrieval to computers. Unlike humans, computers do not own the reasoning capability to retrieve information. Hence, we designed H-AT, a program to quickly extract information given desired templates.

The 3 information templates that H-AT can extract are:

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

Solutions

H-AT is designed as a script-based program that takes as input a list of text files for information extraction and returns the corresponding templates filled with information. Initially, H-AT extracts NLP features such as dependency parse trees, name entities, lemmas and other features that serve as basics for filling information into templates.

Implementation

Programming tools

H-AT is built based on:

- Python3 - a primary programming language in H-AT
- Spacy - a common NLP package consisting of ML-based functions to extract dependency parse trees, semantic parsing, etc.
- Scikit-learn - a Machine Package that provides fundamentals for Spacy.
- NLTK - a common NLP package consisting of rule-based functions to extract synsets from WordNets, etc.

Program Architecture

main.py

- Clas **IE** - execute the pipeline of filling templates
    - Function **extract -** for each text file, read text, extract NLP features using Spacy and NLTK, and finally feed extracted NLP features into **NLP class** to find all possible matches for the given templates.
- Class **DefaultHelpParser** - to ensure the command-line arguments are provided correctly.
    - Function **error** - raise errors relevant to provided command-line arguments

nlp.py

- Class **NLP** - class to extract NLP features and to fill information to templates
    - Function **_get_features** - for every sentence and every token, their corresponding features are extracted. The feature lists include:
        - POS Tagging
        - Dependency Parse Trees
        - Name Entities
        - Lemmas
        - Hypernyms, hyponyms, meronyms, holonyms
    - Function **fill_born** - find all possible matches of BORN templates
        - Check for the "bear" lemma within sentences
        - Retrieve all entities tagged with PERSON (Person), DATE (Date), LOC (Location), and GPE (Geopolitical).
        - For sentences with the "bear" lemma and at least 1 PERSON entity:
            - Retrieve the nsubjpass or nsubj tokens in the sentence
                - Mark this with **bornee** if its entity type is also PERSON
            - Retrieve all the pobj (preposition objects) of the sentence
                - Mark these with **date** if their entity type is DATE
                - Mark these with **loc** if their entity type is LOC or GPE
            - In a linear fashion, plug in the bornee, date, and loc items found into the **BORN template** until there's no more **bornees** in the sentence with respective **date** and **loc.**
    - Function **fill_acquire** - find all possible matches of ACQUIRE templates
        - Iterating over each sentence in the text
        - Retrieve all entities tagged with ORG (Organization) and DATE (Date)
        - Check the passive and active forms of the text by comparing the first ORG element with the token labelled as either **nsubjpass** or **nsubj** in by the dependency parse tree.
            - If passive, the **buyer** is the last ORG element.
            - If active, the **buyer** is the top ORG element
        - Then, respectively retrieve the **next ORG and DATE elements** and group them with **buyer** into **a single tuple of ACQUIRE template**

- Function **fill_part_of** - find all possible matches of PART_OF templates
  - Check for the "be", "is", and "part" lemmas within sentences
  - Retrieve all entities tagged with ORG (Organization), LOC (Location), and GPE (Geopolitical).
  - For sentences with these lemmas and at least 2 PERSON OR 2 (LOC or GPE) entities:
    - In a linear fashion, plug in the found entities into the **PART-OF template** until there's no more pairs in the sentence.
- Function **fill** - for each given text, find all possible matches of 3 templates
  - This function calls all 3 of the fill functions above:
    - **fill_born**
    - **fill_acquire**
    - **fill_part_of**
- Function **extract** - for every provided text, returns a list of sentences, a list of tokens, and a list of features returned by the function **_get_features** above
  - Calls the **_get_features()** function to retrieve the task 1 features of the document.

Results and Error Analysis

Entities and all template matches for document **3.txt** is displayed below.

```
Extracting NLP Features:
-----------------------
Extracting NLP features for article, 3.txt
100%|                                                                                | 5/5 [00:00<00:00, 33.20it/s]

Entities in document, 3.txt

[[('Washington', 0, 10, 'GPE'), ('John Washington', 31, 46, 'PERSON'), ('1656', 61, 65, 'DATE'), ('Sulgrave', 71, 79, 'GPE'),
('England', 81, 88, 'GPE'), ('the British Colony of Virginia', 92, 122, 'GPE'), ('UK', 124, 126, 'GPE'), ('5,000 acres', 148,
159, 'QUANTITY'), ('Little Hunting Creek', 179, 199, 'FAC'), ('the Potomac River', 203, 220, 'LOC')], [('George Washington', 0
, 17, 'PERSON'), ('February 22, 1732', 27, 44, 'DATE'), ('Popes Creek', 48, 59, 'GPE'), ('Westmoreland County', 63, 82, 'GPE')
, ('Virginia', 84, 92, 'GPE'), ('first', 106, 111, 'ORDINAL'), ('six', 115, 118, 'CARDINAL'), ('Augustine', 131, 140, 'GPE'),
('Mary Ball Washington', 145, 165, 'PERSON')], [('three', 76, 81, 'CARDINAL'), ('first', 111, 116, 'ORDINAL'), ('Jane Butler',
 129, 140, 'PERSON')], [('Little Hunting Creek', 20, 40, 'FAC'), ('Ferry Farm', 50, 60, 'ORG'), ('Fredericksburg', 66, 80, 'GP
E'), ('Virginia', 82, 90, 'GPE')], [('Augustine', 5, 14, 'PERSON'), ('1743', 23, 27, 'DATE'), ('Washington', 29, 39, 'GPE'), (
'Ferry Farm', 50, 60, 'ORG'), ('ten', 65, 68, 'CARDINAL'), ('Lawrence', 100, 108, 'PERSON'), ('Little Hunting Creek', 119, 139
, 'FAC'), ('Mount Vernon', 155, 167, 'GPE')]]
```

```
Extracting templates:
-----------------------
Extracting templates for article, 3.txt

Extracted templated for document, 3.txt
{'document': '3.txt', 'extractions': [{'template': 'BORN', 'sentences': 'George Washington was born February 22, 1732 at Popes
 Creek in Westmoreland County, Virginia, and was the first of six children of Augustine and Mary Ball Washington.', 'arguments
': {'1': 'Washington', '2': None, '3': 'Creek'}}, {'template': 'PART_OF', 'sentences': "Washington's great-grandfather John Wa
shington immigrated in 1656 from Sulgrave, England to the British Colony of Virginia, UK where he accumulated 5,000 acres of l
and, including Little Hunting Creek on the Potomac River.", 'arguments': {'1': 'Washington', '2': 'Sulgrave'}}, {'template': '
PART_OF', 'sentences': 'George Washington was born February 22, 1732 at Popes Creek in Westmoreland County, Virginia, and was
the first of six children of Augustine and Mary Ball Washington.', 'arguments': {'1': 'Popes Creek', '2': 'Westmoreland County
'}}, {'template': 'PART_OF', 'sentences': 'When Augustine died in 1743, Washington inherited Ferry Farm and ten slaves; his ol
der half-brother Lawrence inherited Little Hunting Creek and renamed it Mount Vernon.', 'arguments': {'1': 'Washington', '2':
'Mount Vernon'}}]}

Extracting NLP Features:
```

Most errors occur with PART_OF templates. This is due to the wrongly detected entities. Consider the last PART_OF template:

{

'template': 'PART_OF',

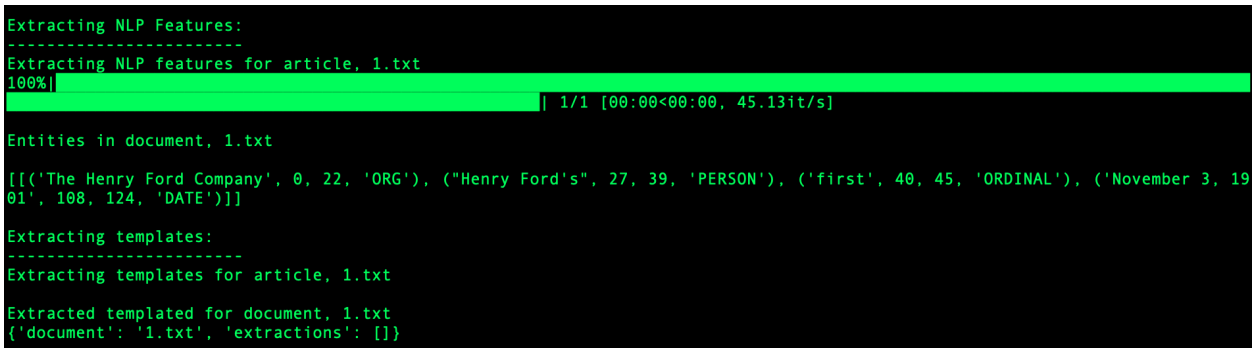
'sentences': 'When Augustine died in 1743, Washington inherited Ferry Farm and ten slaves; his older half-brother Lawrence inherited Little Hunting Creek and renamed it Mount Vernon',

'arguments': {'1': 'Washington', '2': 'Mount Vernon'}

}

The sentence is about Washington's brothers. However, the PART_OF template's result consists of Washing and Mount Vernon. Obviously, Mount Vernon is a location. However, the word "Washington" refers to Mr. Washing, the first U.S. president. It is tagged as a location/state. Hence, our PART_OF algorithm misunderstood this and incorrectly group "Washington", a PERSON and "Mount Vernon", a Location together into the PART_OF template. Hence, a better Name Entity Recognition is necessary.

Pending Issues



Another issue remaining in H-AT is missing the use of coreference. Coreference is necessary in identifying references between words. The result of missing coreference is the empty template matches in document **4.txt.**

Another issue is the lengthy running time. Since H-AT implements the linear code flow, there are bottlenecks at extracting NLP features. To remove the bottlenecks, multiprocessing/threading could be taken into consideration for real-life deployment.

Improvements

The first improvement we would do is adding coreference as a feature. We attempted adding coreference, however, it was a very difficult task to successfully integrate it into the NLP pipeline and get back useful results.

Another improvement would be the use of the dependency parse tree. We also attempted to use this parse tree. For example, for the BORN template, what we attempted was:

1. For each sentence:
    a. Find the lemma "bear" within the dependency tree
        i. If present, check its children for a nsubj or nsubjpass
            1. Check if the entity type is "PERSON" or "ORG"
                a. Assign this to the BORN template's 1st param
        ii. Check its children for a prep (preposition)
            1. Check the children of the prep for a "LOC" or "GPE" entity
                a. Assign this to the BORN template's 3rd param
            2. Check the children of the prep for a "DATE" entity
                a. Assign this to the BORN template's 2nd param

The 3rd major improvement would be improved performance by optimizing the code and possibly using Python's multiprocessing library. Our current program sometimes would take 5 minutes to run on an older processor, and many of its tasks could be split up using multiprocessing, such as the parsing of all the input documents and analysis of the sentences via our pipeline. Multiple documents and multiple sentences can be done at the same time.

Another potential improvement is using coreference. By using coreference, the abstract nouns/subjects could be clustered with proper pronouns. By that way, more template matches will be found and be more accurate. For example, IBM shifted its operation plans into seeking for advanced technologies and acquiring them. This includes acquiring A (2012), and B(2013). Obviously, if coreference is used, then the word “this” could be clustered with “IBM”. Hence, given the above algorithm, the coreference helps find more template matches.