Named Entity Recognition (NER) in Lorimer's *Gazetteer* with NLTK

I. Project motivation.

This project sits at the intersection of Data Mining, Digital Humanities and Natural Language Processing, particularly, Information Extraction. We aim at detecting the geographical Named Entities from the John G. Lorimer's *Gazetteer of the Persian Gulf, Central Arabia and Oman*, a canonical artifact of British imperial knowledge production about the Gulf region, including contemporary Iran, Iraq and the GCC states. The dataset includes more than 800 text files pertaining to each geographical location. View more details about the dataset here.

Whereas a group of researchers have been annotating the Gazetteer manually and in a semi-automated way using Recogito, an NER system with higher levels of automation would enable a more in-depth and computational approach to the analysis of the data. In particular, an automated NER system enables the extraction of an unprecedented amount of information in a short span of time and helps to navigate some of the limitations common for manual and semi-automated systems (continuous inter-annotation collaboration, time and labor constraints). Moreover, the output of the automated system, the dictionary of Named Entities from all text files allows reevaluating existing assumptions about the data, finding new patterns and answering research questions that would be considerably difficult to answer before (e.g. "How often text about location A references location B?", "what is the connection between the text files" etc.).

Moreover, the Gazetteer has an incomparably diverse and extensive set of geographical named entities that could be of great use for future visualization and data mining projects. You can also learn more about similar Data Mining projects in Humanities in this paper.

I. **What is NER and its importance**

Named Entity Recognition (NER) is the task of locating, extracting and classifying names with a specific set of named entity types (e.g. Person, Organization, Location). This task can be broken down into two sub-tasks: identifying the boundaries of the named entity and identifying its type. Below are the most common types of Named Entities that are supported by NLTK, Stanford CoreNLP and other libraries.

NE Type	Examples
ORGANIZATION	*Georgia-Pacific Corp., WHO*
PERSON	*Eddy Bonte, President Obama*
LOCATION	*Murray River, Mount Everest*
DATE	*June, 2008-06-29*
TIME	*two fifty a m, 1:30 p.m.*
MONEY	*175 million Canadian Dollars, GBP 10.40*
PERCENT	*twenty pct, 18.75 %*
FACILITY	*Washington Monument, Stonehenge*
GPE	*South East Asia, Midlothian*

Figure 1. This table shows examples of named entities and their types. Source: NLTK textbook.

NER has valuable applications for many important NLP tasks with the most impactful being Information Retrieval, a task of identifying and retrieving documents based on a query (e.g. Google search). IR can benefit from NER in two ways: recognizing named entities (NEs) within the searched documents, and then extracting the relevant documents considering their classified NEs and how they are related to the query (Rosso 2009a).

For example, the word "Aljazeera" can be tagged either as an Organization (news source) or as a Location (island) - correct NE tagging will facilitate extraction of the correct documents based on a query.

In these series of blog posts, we explore two Named Entity Recognition models: an NER model using NLTK (Natural Language Toolkit) and an NER model custom-trained for Arabic language using Spacy.

III. Named Entity Recognition with NLTK

III. A. About NLTK and its Information Extraction architecture, as outlined by NLTK.

The **Natural Language Toolkit**, or more commonly **NLTK**, is a set of libraries written in python for Natural Language Processing for English-language texts by Steven Bird and Edward Loper at the University of Pennsylvania. NLTK is a leading platform for many NLP tasks including Named Entity Recognition, therefore, an NER model based on NLTK can serve as a good baseline. However, as authors recognize themselves (NLTK, chapter 7), it is trained primarily on English language text and, thus, does not always predict Named Entity labels and values for foreign language or transliterated texts.

III. B. System description.

Naturally, our NER model based on NLTK follows the Information Extraction Architecture outlined by NLTK:

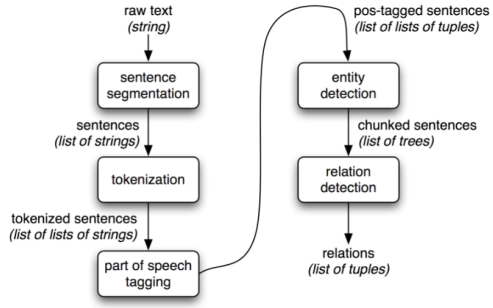


Figure 1. A sample Information Extraction Architecture pipeline. Source: [NLTK textbook.]

To summarize the steps, the system starts with **splitting the raw text** into words using a tokenizer (sometimes pre-segmenting text into sentences first). Next, each sentence is tagged with **part-of-speech tags,** which will prove very helpful in the next steps: **chunking** and finally **named entity detection**.

1. Text preprocessing.

Firstly, to facilitate work of the classifier and convert input texts to the suitable format for NLTK, we need to preprocess input texts. After experiments and literature review, we performed the following preprocessing steps:

a. Transliterate common Arabic letters to English letters based on the list provided by Professor Wrisley. This step was introduced after the initial run of the system showed that entities starting with an Arabic letter are not recognized; for example, the word Ārabistan is not recognized. This could be potentially explained by the fact that the NLTK model is not trained on non-English words (see more in the 'Flaws' section).

letter	example	original letter	transformed letter	transformed example
"a" with line on top	Sohār	ā	a	Sohar
"A with line on top	Āl	Ā	A	Al
"i" with line on top	Sa'īd	ī, Ī	i	Said
"u" with a line on top	Khābūrah	ū,Ū	u	Khaburah
"o"with a line on top				
apostrophe, beginning of word " ' "	'Omān			Oman
apostrophe, end of word " ' "	Sabai'			Sabai
apostropher, middle of word " ' "	Ka'ab-as-Sitātleh			Kaab-as-Sitatleh

Figure 3. Conversion of Arabic letters to their English counterparts. Credits to Professor Wrisley.

b. Remove stop words. Stop words are defined as words that do not have semantic importance and are commonly removed in Information Retrieval tasks. In our model, we remove stop words defined by NLTK, which usually include pronouns, articles, etc. Learn more about stop words in NLTK here.

Based on the initial runs of the model, we also created our own list of stop words, which currently includes month and day of the week names. These words are recognized as a named entity by NLTK, but are not of interest for our current research questions. See figure below.

```
[ ] stopWords=[
    'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday','Sunday', 'January',
    'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October',
    'November', 'December', 'Mondays', 'Tuesdays', 'Wednesdays','Thursdays', 'Fridays', 'Saturdays', 'Sundays'
    ]
```

Figure 4. List of stop words created for our model. Source: NER with NLTK pipeline.

2. Import libraries.

After preprocessing, we import required libraries: we import nltk and also use glob, os, csv to handle file reading and writing. We load the nltk 'words' corpora to use the english language training corpus. We download the punkt library for sentence tokenization, averaged_perceptron_tagger for part of speech tagging and 'maxent_ne_chunker' for chunking the tokens using part of speech tags.

```
import nltk
import glob
import os
import csv
nltk.download('words')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('maxent_ne_chunker')
from nltk import word_tokenize,pos_tag,ne_chunk
import matplotlib as mpl
import matplotlib.pyplot as plt
```

Figure 5. List of libraries required for our model. Source: NER with NLTK pipeline.

3. Reading the files.

We first define the file path and then use the glob library to iterate through each text file in the folder. We have also created entities_filtered and entity_name arrays to store the needed information for each file and ner_full and all_entity_names arrays to store information across all files - we will write our output to a csv file. Then, we clean the filename and finally read each file as a large string.

```
path='/content/drive/MyDrive/Lorimer/'
#store full entity objects for all text files
ner_full=[]
#store all entity names for all text files
all_entity_names=[]
for filename in glob.glob(os.path.join(path, '*.txt')):
    #store only named entities
    entities_filtered=[]
    #store only named entities without label (GPE,LOC,etc)
    entity_names=[]
    with open(filename, 'r') as file:
        filename1=filename.replace("../Lorimer/","")
        #store filenames to write to the csv later
        filenames.append(filename1)
        #replace newlines to read the file
        data = file.read().replace('\n', '')

        tokens = word_tokenize(data)
        print("All tokens")
        for x in tokens:
          print(x)
```

Figure 6. Reading the input files and converting raw text to string. Source: NER with NLTK pipeline.

4. Tokenization.

Tokenization refers to splitting the text into tokens, which is a sequence of characters that we want to treat as a group such as hairy, his, or :). In our model, we use the built-in word_tokenize function and below is the snippet of the code and output. Notice that NLTK tokens can include words, punctuation, apostrophes etc.

```
tokens = word_tokenize(data)
print("All tokens")
for x in tokens:
  print(x)
```

```
76
25th
February
No
.
516
4th
March
1906
;
Government
India
's
Foreign
Proceedings
April
1901
may
also
```

Figure 7. Tokenization of the text and model output. Source: NER with NLTK pipeline.

5. Part of Speech tagging.

A part-of-speech tagger, or POS-tagger, processes a sequence of words, and attaches a part of speech tag to each word. NLTK provides documentation for each tag, which can be queried using the tag, e.g. nltk.help.upenn_tagset('NN'). Some corpora have README files with tagset documentation, see nltk.corpus.corpora_name.readme(). A list of most common NLTK part of speech tags can be viewed in the Penn Treebank POS tagset.

Part of speech tagging is necessary since many words like ski and race can be used as nouns or verbs and tag information helps to extract the semantic meaning of the word. POS tags are used as input for chunking, the next step in NER recognition. Named entities usually have a tag NNP (proper noun). We use nltk's pos_tag function and have the following output: Our pipeline produces the following output:

```
pos_tags=nltk.pos_tag(tokens)
print("POS Tags")
for x in pos_tags:
  print(x)
```

```
('516', 'CD')
('4th', 'CD')
('March', 'NNP')
('1906', 'CD')
(';', ':')
('Government', 'NNP')
('India', 'NNP')
("'s", 'POS')
('Foreign', 'NNP')
('Proceedings', 'NNP')
('April', 'NNP')
('1901', 'CD')
('may', 'MD')
('also', 'RB')
('consulted.8', 'VB')
('.', '.')
('Some', 'DT')
('authorities', 'NNS')
(',', ',')
('however', 'RB')
(',', ',')
('suppose', 'VBP')
('purely', 'RB')
('indigenous', 'JJ')
```

Figure 8. Part of speech tagging of the text and model output. Source: NER with NLTK pipeline.

6. Chunking.

The basic technique that NLTK uses for entity recognition is called chunking, which segments and labels multi-token sequences (see figure 9). The smaller boxes show the word-level tokenization and part-of-speech tagging, while the large boxes show higher-level chunking. Each of these larger boxes is called a **chunk**. Like tokenization, which omits whitespace, chunking usually selects a subset of the tokens. Also like tokenization, the pieces produced by a chunker do not overlap in the source text.

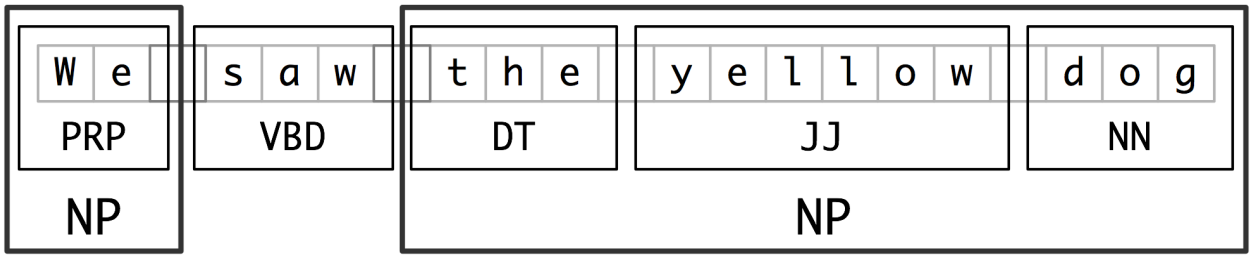
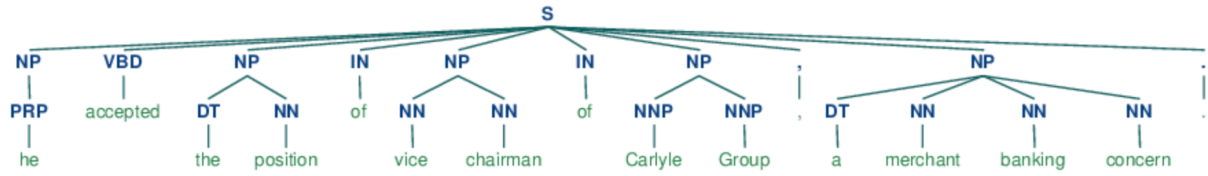


Figure 9. Segmentation and Labeling at both the Token and Chunk levels. Source: [NLTK textbook.]

Chunks can also be visually represented as a tree, view a figure here.



N**oun phrase chunking**, or **NP-chunking, a** required step for NER , refers to searching for chunks corresponding to individual noun phrases. Chunking can be accomplished using regular expressions if we provide a sequence of tags that defines a chunk. In this project, however, we use the nltk built-in library called 'maxent_ne_chunker' to create chunks; see the output and code below.

```
chunks = ne_chunk(pos_tags)
print("All chunks")
for x in chunks:
  print(x)
```

```
('Government', 'NNP')
('of', 'IN')
(GPE India/NNP)
('for', 'IN')
('October', 'NNP')
('1905', 'CD')
('may', 'MD')
('be', 'VB')
('consulted.11', 'NN')
('.', '.')
('The', 'DT')
('table', 'NN')
('below', 'NN')
('may', 'MD')
('be', 'VB')
('compared', 'VBN')
('with', 'IN')
('that', 'DT')
('at', 'IN')
('page', 'NN')
('66', 'CD')
('of', 'IN')
('the', 'DT')
(LOCATION Persian/NNP Gulf/NNP)
('Administration', 'NNP')
```

Figure 10. Chunking tagged words and model output.

Take a look at the underlined output tuples containing the tokens and Named Entity tags. Note that NEs can consist of more than one word: *India* is tagged as GPE or Geopolitical entity and consists of one token, while *Persian Gulf* is a two-token NE and has a Location tag. Description of most common Named Entities is available here.

Moreover, note that not all personal pronouns (NNP) are tagged as Named Entity (have NE tag). For example, the word October is NNP, but is not an NE (doesn't have an NE tag).

7. Named Entity Recognition.

Finally, we use the generated chunks to find named entities, which have category labels such as PERSON, ORGANIZATION, and GPE, etc. NLTK provides a classifier that has already been

trained to recognize named entities and can be accessed with the function nltk.ne_chunk().

We store named entities in two ways: in the 'entities_filtered' array, we store the NEs with their full info (POS tag and label) and in the 'entity_names' we also store only their values ('Iran'). The content of the array 'entity_names' is particularly useful for the OpenGulf project as it allows to isolate and analyze the geographical NEs in each text file. Below you can view the sample code and output. Note that, at the end of the inner loop, we are storing info from each file in larger, outer-loop-level arrays (ner_full and all_entity_names) to access NEs across all files.

```
    entity_names=[]
    entities_filtered=[]
    print("Labelled entities (Org, Person, GPE etc.):")
    for chunk in chunks:
      if hasattr(chunk, 'label'):
        #chunk.label(), ' '.join(c[0] for c in chunk)
        entity=str(chunk.label())
        print("Entity: "+str(chunk.label())+", ",end="")
        name=chunk[0][0]
        if name not in stopWords:
          entity_names.append(name)
          print("Name: "+str(name)+", ",end="")
          pos_tag=chunk[0][1]
          print("POS tag: "+str(pos_tag))
          string1=str(entity)+","+str(name)+","+pos_tag
          entities_filtered.append(string1)
#store all entity names for all text files
all_entity_names.append(entity_names)
#store all entity info for all text files
ner_full.append(entities_filtered)
```

```
Labelled entities (Org, Person, GPE etc.):
Entity: PERSON, Name: Abduli, POS tag: NNP
Entity: ORGANIZATION, Name: Trucial, POS tag: NNP
Entity: ORGANIZATION, Name: Sharjah, POS tag: NNP
Entity: PERSON, Name: Town, POS tag: NNP
Entity: ORGANIZATION, Name: Ghallah, POS tag: NNP
Entity: GPE, Name: Shamailiyah, POS tag: NNP
Entity: ORGANIZATION, Name: Khalaibiyah, POS tag: NNP
Entity: PERSON, Name: Wadi, POS tag: NNP
Entity: PERSON, Name: Shaikh, POS tag: NNP
Entity: GPE, Name: Ghafiris, POS tag: NNP
Entity: PERSON, Name: Hanbali, POS tag: NNP
Entity: ORGANIZATION, Name: Sharqiyin, POS tag: NNP
Entity: PERSON, Name: Shurafa, POS tag: NNP
Entity: GPE, Name: Makkah, POS tag: NNP
Entity: ORGANIZATION, Name: Obaidli, POS tag: NNP
Entity: ORGANIZATION, Name: Shibkuh, POS tag: NNP
Entity: GPE, Name: Persia, POS tag: NNP
Entity: ORGANIZATION, Name: Abdah, POS tag: NNP
Entity: ORGANIZATION, Name: Shammar, POS tag: NNP
Entity: GPE, Name: Najd, POS tag: NNP
Entity: GPE, Name: Karun, POS tag: NNP
```

Figure 11. Named Entity classification and model output.

8. Output description and writing to csv.

Lastly, we write output of our NER model into csv files. We have two outputs from our model:

1. A list of all entities with their names, corresponding labels and POS tags. The generated csv file is called 'all-entities.csv'.

   In the csv file (see figure 12 below), the name of each text file AAA precedes the table consisting of three rows: i) labels of all entities found in the file AA, ii) corresponding names of these entities and iii) Part of Speech tags of these entities.

allEntities.csv

	A	B	C	D
1	kaunak.txt			
2				
3	ORGANIZATION	KAUNAKکاونك	NNP	
4	PERSON	Geographical	NNP	
5	ORGANIZATION	Dizful	NNP	
6	GPE	Northern	NNP	
7	GPE	Dizful	NNP	
8	GPE	Shushtar	NNP	
9	PERSON	Kaunak	NNP	
10	ORGANIZATION	Diz	NNP	
11	GPE	Kaunak	NNP	
12	ORGANIZATION	Dizful	NNP	
13	GPE	Kaunak	NNP	
14	ORGANIZATION	Kadkhuda	NNP	
15	GPE	Kaunak	NNP	
16	GPE	Kaunak	NNP	
17	GPE	Bakhtiyaris	NNP	
18	PERSON	Shushtar	NNP	
19	PERSON	Dirakwand	NNP	

2. A list of all entity names. The generated csv file is called 'all-entity-names.csv'. This particular file has already been used at OpenGulf to retrieve a list of dominant entities for each file.
   In the csv file, the first column of each row is the name of the text file and each row contains a list of entities for a particular text file (see below).

all-entity-names.csv

Open with

kaunak.txt	KAUNAKكاونك	Geographical	Dizful	Northern	Dizful	Shushtar	Kaunak
rud_hilleh_district.tx	DISTRICT	Geographical	Persian	Gulf	Hayat	Angali	Shabankareh
mazara_.txt	MAZARAمزارع	Geographical	Wadi	Sultanate	Oman	Bani	Bani
fahal.txt	FAHAL	Geographical	Masqat	Oman	Fahal	Portuguese	Isle
taimah.txt	TAIMAHتيمه	Geographical	Jabal	Hail	Madinah	Taimah	Haddaj
zaid_bani.txt	ZAID	BANI	زيد	Geographical	Arab	Southern	BaniTamim
qamain.txt	QARNAIN	Geographical	Abu	Arzanah	Das	Zirko	Qamain
jaramah_khor_al.txt	JARAMAH	الجرامه	Geographical	Oman	Khor	north-west	Muwalikh
qarah_barr_al.txt	QARAH	القاره	Geographical	Eastern	Sanjaq	Hasa	Qatar
hazaim.txt	HAZAIMحزيم	Geographical	Kuwait	Salu	Shaqq	Maqta	Labibah
farsi.txt	FARSIفارسي	Geographical	Persian	Musallamiyah	Hasa	Baraki	Tangistan
arabi.txt	ARABIعربيAn	Musallamiyah	Farsi	Arabi			
dhafrah.txt	DHAFRAH	Geographical	Oman	Trucial	Persian	Great	Southern
sirr_wadi_as.txt	SIRR	Geographical	Najd	Shaqrah	Anaizah	Anaizah	Shaqrah
akhdhar_jabal.txt	AKHDHAR	JABAL	Western	Oman	Hajar	Oman	Batinah
dahanah.txt	DAHÁNAHدهنه	Geographical	Central	Arabian	Persian	Dahánah	Jabrin
tayin_wadi.txt	TAYIN	WADI	طايين	Geographical	Oman	Eastern	Wadi
halfain_wadi.txt	HALFAIN	WADI	حليفين	Geographical	Oman	Arabia	Wadi

3. Additionally, the NER system is capable of outputting only the first 20 NEs in a single file, rather than all of them. See the code here.

Below is the code that is used to generate the output files:

```
headers=[]
for fe in filenames:
    headers.append(fe)

with open('allEntities.csv', 'w') as f:
    writer = csv.writer(f)
    # write the header
    for i in range(len(ner_full)):
        f.write(str(filenames[i]))
        f.write("\n")
        f.write("\n")
        for k in range(len(ner_full[i])):
            f.write(str(ner_full[i][k])+",")
            f.write("\n")
        f.write("\n")
```

```
with open('all-entity-names.csv', 'w') as f:
    for i in range(len(all_entity_names) ):
        values = ','.join(str(v) for v in all_entity_names[i])
        f.write(str(filenames[i])+",")
        f.write(values)
        f.write("\n")
```

IV. Discussion of advantages and potential flaws of the NLTK model.

One of the potential weaknesses of the NLTK NER classifier is that it is trained on an English language dataset. Therefore, for texts containing non-English and transliterated words such as Lorimer's Gazetteer, NLTK does not always recognize entity names and tags correctly. Our dataset mostly includes transliterated entity names (from Arabic to English) with the majority of the entity names used in a specific historical and geographical context and, thus, less likely to appear in the NLTK's training data. Moreover, there are differences among Arabic to English transliteration formats, which further complicates the NER task for an only-English trained classifier. To address these challenges, we have created a custom-trained NER model with Spacy, which is trained based on the manually annotated dataset of entity names and labels from the Gazetteer.

There are other general (language-independent) challenges that arise with the NER task. The word North and May can be parts of named entities for DATE and LOCATION, respectively, but could both be part of a PERSON. Conversely *Christian Dior* looks like a PERSON but is more likely to be of type ORGANIZATION. A term like *Yankee* will be an ordinary modifier in some contexts, but will be marked as an entity of type ORGANIZATION in the phrase *Yankee infielders*. Moreover, in the named entity recognition, both the beginning and end of multi-token sequences have to be identified: NEs can be multi-word names such as Persian Gulf or Oman National Library. There is no perfect solution to these general challenges yet, however, being able to train on manually selected texts (Lorimer’s dataset) and continuously finetune the classifier based on performance allows to address most of the potential challenges.