LING 406: Intro to Computational Linguistics
Spring 2024

Assignment #1: *Temporal Named Entity Recognition*

Issued: Jan. 25, 2024
Due: Feb. 4, 2024 11:59 pm
Credits: 60 points

Objective

Searching large collections of unstructured data requires fast, convenient, and robust processing methods. Consider, for example, the amount of text data being generated on Social Media every day. Unstructured data is usually complex and noisy. Fortunately, there are techniques useful to extract important features from such data.

In this assignment, you will get hands-on experience with regular expressions that can be used in frequent pattern matching and searching as part of natural language processing applications. This combination of string manipulation functions and regular expressions will prepare you for more complex computational linguistics tasks later in the course. For example, some common text processing tasks where regular expressions can come handy are tokenization, chunking and stemming.

Description

For many text-based applications (i.e., Information Extraction, Automatic Question Answering), date and time **named entity recognition** is a tremendously useful task, yet a challenging one to provide consistent results from application to application. The reason is that there are so many arbitrary dates, ranges, and holidays that can be expressed in so many ways and are context dependent.

In this assignment, however, you are going to build **a simple temporal named entity recognizer** using regular expressions. In particular, you are asked to design and write a program that recognizes simple date expressions of two types:

1. Fixed dates (including exact time if mentioned in the expression)

   Examples: “January 15, 2014”, “the 21st of December”, “01/15/2014” (only the American notation), “Monday”, “Monday the 23rd”, “Monday, 2pm”, “Monday afternoon”

2. Holidays (only American holidays; to guide you, here is a list of Federal holidays: https://www.redcort.com/us-federal-bank-holidays)

Examples: “Labor Day”, “Memorial Day”.

Keep in mind that you have to identify and extract the longest expression mentioned to receive full credit. For example, if the text says

“.. The mayor addressed the audience on Feb. 14th, 2012..”,

your program should extract "Feb. 14th, 2012" and not only "Feb. 14th" or just “2012”.

Your program should recognize all such “absolute” dates, like those shown above and use representations that capture classes of words (e.g., days of the week, months, holidays such as Labor Day, Memorial Day, Christmas, etc.). Your system should NOT extract dates relative to a particular day, like "the day before yesterday", “two years ago”, dates relative to temporal focus (“3 days later”), absolute dates with imprecise/broad reference (“in the beginning of the 80s”, “Women often wore bonnets in the 19th Century”), relative dates with special forms (seasons), basic duration ("during 3 years"), duration as interval (e.g., “from February 11th to October 27th . . .”[1];), relative duration, w.r.t utterance time (“for a year”), and temporal atoms (“three days, four years, . . .”). Years before 1000 AD are not of interest here and should not be considered.

We will not test your system on incorrectly formatted or misspelled dates (such as “the 13 of May” or “December, 25th"). We might test your system on dates with incorrect numbers, like “October 95th, 2018”. Thus, you have to make sure you have the appropriate checks in place for the date number in your regular expression. However, we will only require that you match date numbers between 1-31. Although this is not correct in real world applications, (February 31st is not a real date), we will not penalize your system if it matches such dates.

Special case: Words like “May” and “June” are ambiguous in English: they can identify the month or a person’s name, location, etc. Disambiguating such expressions goes beyond the scope of this assignment. Thus, it will not be considered an error if the system recognizes “May” as a date when the context shows it is not.

Grading

Your system should take as input a text file (input.txt) – e.g., like a concatenation of the bullet point sentences in the Appendix which show all correct dates in bold – and should identify and extract a list of dates as output (one per line), saved as output.txt. We will run your program on the examples below as well as on another file with *different instances of the same types of date expressions*. Your grade will be proportional to the performance of the output.

You should also detail (in a file called system-description.txt) the design decisions you made for your system (e.g., what regular expression types / data structures you decided to use to capture the different types of temporal named entities and why).

[1] *Note, however, that “February 11th” and “October 27th” are positive examples

The assignment is worth 60 points: 10 points for the design and 50 points for the implementation of the system. Each type of regular expression considered (i.e., NOT individual instances, but the “Fixed dates” and “Holidays” category) is worth 25 points (for a total of 50 points). For each type, we will deduct 5 points per missing date entity instance (if you miss more than one instance). We will also penalize for incomplete date instances.

Note: All programs must be written according to the specifications mentioned in the Course Description file (see Canvas). It is your responsibility to make sure your code runs properly and accomplishes the task before you submit your work. You are advised to contact the TAs ahead of time if you need assistance with the assignment.

Deliverables:

Using GitHub, add, commit, and push your source code, a README.md file, the output.txt file, the system-description.txt file, as well as any files we need to access in order to grade the assignment correctly. The README.md file should indicate the name of the program, a short description of the problem solved and a short description on how to run your code. You must comment your code accordingly (i.e., following good coding practice guidelines). 5 points will be deducted if any of these deliverable files is missing.

Important:

1) You are NOT allowed to make use of (online) tools to capture regular expressions based on a given input text. You have to figure out what regular expressions will match the input dates on your own!

2) Regular expressions are very useful in capturing various dates of the same type. Do NOT hardcode your date instances into your system or write very long and obfuscating one-liners! Justify your design in your comments and the README.md and system-description.txt files.

Appendix

Your code should extract the following types of dates, as instantiated by the examples below extracted from various Social Media platforms. The bolded text indicates what your regular expression should match in each sentence.

- He will arrive on **January 5th, 2015**.
- King was born on **January 15, 2029** in Atlanta, Georgia.
- It was a **Tuesday afternoon**.
- Sam was born in **May.**
- He was born on **the 13th of May**.
- **May 13, 2007** was a **Sunday**.
- We fly to Denver on **Monday**.
- What happened then was an introduction to the events of **September 11, 2001**.
- National Boyfriend Day (sometimes referred to as National Boyfriend's Day) on **October 3rd** recognizes the sweetheart in your life.
- Her grandmother was born in **1935**.
- **November 29th** was on a **Monday**, in the 49th week of **1948**.
- How fortunate that the world did not end on **Friday, December 21st, 2012**!
- Today is **the 1st of January, 2020**.
- It should fall between the two calendar dates, **1/1/2020** and **2/1/2020**.
- What happened on **May** 32nd?
- What is celebrated every year on **the 5th of May**?
- Do we work on **May 5th**?
- **June** is my favorite time of the year!
- **June** is my best friend.
- We will revisit this after **Labor Day**.