Chapter 14

Introduction to Clinical Natural Language Processing with Python

Leo Anthony Celi, Christina Chen, Daniel Gruhl, Chaitanya Shivade,
and Joy Tzung-Yu Wu

Abstract Background: Many of the most valuable insights in medicine are contained in written patient records. While some of these are coded into structured data as part of the record entry, many exist only as text. Although a complete understanding of this text is beyond current technology, a surprising amount of insight can be gained from relatively simple natural language processing. **Learning objectives:** This chapter introduces the basics of text processing with Python, such as name-entity recognition, regular expressions, text tokenization and negation detection. By working through the four structured NLP tutorials in this chapter, the reader will learn these NLP techniques to extract valuable clinical insights from text. **Limitations:** The field of Natural Language Processing is as broad and varied as human communication. The techniques we will discuss in this chapter are but a sampling of what the field has to offer. That said, we will provide enough basic techniques to allow the reader to start to unlock the potential of textual clinical notes.

Keywords Natural language processing · Concept extraction · Text analytics

14.1 Introduction

Natural Language Processing (NLP) is the ability of a computer to understand human language as it is spoken or written (Jurafsky and Martin 2009). While that sounds complex, it is actually something you've probably been doing a fairly good job at since before you were four years old.

Most NLP technology development is akin to figuring out how to explain what you want to do to a four-year-old. This rapidly turns into a discussion of edge cases (e.g., "it's not *gooder*; it's *better*"), and the more complicated the task (i.e., the more

L. A. Celi · C. Chen

Institute for Medical Engineering and Science, Massachusetts Institute of Technology,
Cambridge, MA, USA

D. Gruhl · C. Shivade · J. T.-Y. Wu (✉)

International Business Machine Research, Almaden, San Jose, CA, USA

e-mail: joy.wu@ibm.com

© The Author(s) 2020

L. A. Celi et al. (eds.), *Leveraging Data Science for Global Health*,
https://doi.org/10.1007/978-3-030-47994-7_14

poorly structured the language you are trying to interpret) the harder it is. This is especially true if you are hoping that an NLP system will replace a human in reliably extracting domain specific information from free text.

However, if you are just looking for some help wading through potentially thousands of clinical notes a bit more quickly, you are in luck. There are many “4-year-old” tasks that can be very helpful and save you a lot of time. We’ll focus on these for this chapter, with some examples.

14.2 Setup Required

This chapter aims to teach practical natural language processing (NLP) for clinical applications via working through four independent NLP tutorials. Each tutorial is associated with its own Jupyter Notebook.

The chapter uses real de-identified clinical note examples queried from the MIMIC-III dataset. As such, you will need to obtain your own Physionet account and access to use the MIMIC dataset first. Please follow the instructions here to obtain dataset access: <https://mimic.physionet.org/gettingstarted/access/>.

However, you will not need to setup the MIMIC SQL dataset locally to download the datasets required for this chapter. For each section, the necessary SQL code to query the practice datasets will be given to you to query the datasets yourself via MIMIC’s online Query Builder application: <https://querybuilder-lcp.mit.edu>.

The NLP demonstration exercises in the chapter are run in the Python Jupyter Notebook environment. Please see the Project Jupyter website for the installation instructions (<https://jupyter.org/install>).

14.3 Workshop Exercises

14.3.1 *Direct Search Using Curated Lexicons*

See Jupyter Notebook: Part A—Spotting NASH

The first example is the task of using notes to identify patients for possible inclusion in a cohort. In this case we’re going to try to find records of patients with Nonalcoholic Steatohepatitis (NASH). It is difficult to use billing codes (i.e., ICD-9) to identify patients with this condition because it gets confounded with a generic nonalcoholic liver disease ICD-9 code (i.e., 571.8). If you need to explicitly find patients with NASH, doing so requires looking into the text of the clinical notes.

In this example, we would like the system to “find any document where the string “NASH” or “Nonalcoholic Steatohepatitis” appears”. Note that in this first filter, we are not going to not worry if the phrase is negated (e.g., “The patient does not

have NASH”) or if the phrase shows up as a family history mention (e.g., “My mom suffered from NASH”). Negation detection will be dealt with separately in tutorial 3. Since Nash is a family name, however, we *will* need to worry about “Thomas Nash” or “Russell Nash”. In general, any further context interpretation will need to be screened out by a human as a next step or be dealt with by further NLP context interpretation analysis.

Accessing notes data

First, we need access to the data. Go to: <https://querybuilder-lcp.mit.edu>. Login with the username and password you have obtained from Physionet to access the MIMIC-III database.

Since NASH is one of the causes of liver failure or cirrhosis, for the purpose of this example, we are going to narrow the search by exporting 1000 random notes where “cirrhosis” is mentioned in the notes. In a real example, you might want to apply other clinical restrictions using either the free text or the structured data to help you better target the notes you are interested in analysing.

In the query home console, paste in the following SQL commands and click “Execute Query”.

```
“MySQL
SELECT SETSEED(0.5);
SELECT *, RANDOM() as random_id
FROM (
  SELECT row_id, subject_id, text
  FROM noteevents
  WHERE text LIKE '%cirrhosis%'
  ORDER BY row_id, subject_id
  LIMIT 1000
) A;
...”
```

After the query finishes running, you should see the tabular results below the console. Now click “Export Results” and pick save as “part_a.csv”. Save the file to the directory (i.e., folder) where you are running your local Jupyter notebook from.

Setting up in Jupyter Notebook

Now we can do some NLP exercises in Jupyter notebook with Python. As with any Jupyter script, the first step is simply loading the libraries you will need.

```
“python
# First off - load all the python libraries we are going to need
import pandas as pd
import numpy as np
import random
from IPython.core.display import display, HTML
...”
```

Then we can import the notes dataset we just exported from Query Builder to the Jupyter Notebook environment by running the following code:

```
“python
filepath = ‘replace this with your path to your downloaded .csv file’
```

```
notes = pd.read_csv(filepath)
...
```

Note, if you already have the MIMIC dataset locally set up, the following code snippet will allow you to query your local MIMIC SQL database from the Jupyter notebook environment.

```
“python
# Data access - if you are using MySQL to store MIMIC-III
import pymysql
conn = pymysql.connect(db='mimiciii', user='XXXXXX', password='YYYYYY',
host='localhost')
notes = pd.read_sql_query("SELECT ROW_ID, TEXT FROM NOTEVENTS WHERE
TEXT LIKE '%cirrhosis%' LIMIT 1000", conn)
...

python
# Data access - if you are using Postgres to store MIMIC-III
import psycopg2
params = { 'database': 'mimic', 'user': 'XXXXXX', 'password': 'YYYYYY', 'host': 'localhost' }
conn = psycopg2.connect(**params)
notes = pd.read_sql("SELECT ROW_ID, TEXT FROM NOTEVENTS WHERE TEXT LIKE
'%cirrhosis%' LIMIT 1000", conn)
...”
```

NLP Exercise: Spotting ‘NASH’ in clinical notes with brute force

We now need to define the terms we are looking for. For this simple example, we are NOT going to ignore upper and lower letter cases, such that “NASH”, “nash”, and “Nash” are considered as different terms. In this case, we will focus exclusively on “NASH”, so we are less likely to pick up the family name “Nash”.

```
“python
# Here is the list of terms we are going to consider “good”
terms = [ 'NASH', 'nonalcoholic steatohepatitis' ]
...”
```

This is the code that brute forces through the notes and finds the notes that have an exact phrase match with our target phrases. We’ll keep track of the “row_id” for future use.

```
“python
# Now scan through all of the notes. Do any of the terms appear? If so stash the note
# id for future use
matches = []
for index, row in notes.iterrows():
    if any(x in row['text'] for x in terms):
        matches.append(row['row_id'])
print("Found “ + str(len(matches)) + “ matching notes.”)
...”
```

Lastly, we pick one matching note and display it. Note, you can “Ctrl-Enter” this cell again and again to get different samples.

```
“python
# Display a random note that matches. You can rerun this cell to get another note.
# The fancy stuff is just highlighting the match to make it easier to find.
display_id = random.choice(matches)
```

```

text = notes[notes['row_id'] == display_id].iloc[0]['text']
for term in terms:
    text = text.replace(term, "<font color='red'>" + term + "</font>")
display(HTML("<pre>" + text + "</pre>"))

```

14.3.2 Adding Flexibility in Search with Regular Expressions

While simple word matching is helpful, sometimes it is more useful to utilize more advanced searches. For example, extracting measurements (i.e. matching numbers associated with specific terms, e.g. HR, cm, BMI, etc.) or situations where exact character matching is not desired (e.g. if one would also like to capture plurals or other tenses of a given term). There are many task specific examples like these where regular expressions (“regex”) (Kleene 1951) can add flexibility to searching information in documents.

You can think of regular expressions as a set of rules to specify text patterns to programming languages. They are most commonly used for searching strings with a pattern across a large corpus of documents. A search using regular expressions will return all the matches associated with the specified pattern. The notation used to specify a regular expression offers flexibility in the range of patterns one can specify. In fact, in its simplest form, a regular expression search is nothing but an exact match of a sequence of characters in the text of the documents. Such direct term search is something we discussed in the previous example for spotting mentions of NASH.

The specific syntax used to represent regular expressions in each programming language may vary, but the concepts are the same. The first part of this tutorial will introduce you to the concept of regular expressions through a web editor. The second part will use regular expressions in Python to demonstrate the extraction of numerical values from clinical notes.

Regular Expression Rules

Sections 14.3.2.1 and 14.3.2.2 will both be using some of the regular expression rules shown below.

By default, X is just one character, but you can use () to include more than one. For example:

- A+ would match A, AA, AAAAA
- (AB)+ would match AB, ABAB, ABABABAB

Special Characters

{ } [] () ^ \$. ! * + ? \ (and - inside of brackets []) are special and need to be “escaped” with a \ in order to match them (which tells us to ignore the special characteristics and treat it as a normal character).

For example:

Table 14.1 Regex—basic patterns

Regex pattern	Matching
.	Anything
\d	Digit in 0123456789
\D	Non-digit
\w	“word” (letters, digits, _)
\W	Non-word
\t	Tab
\r	Return ^a
\n	Newline ^a
\s	Whitespace (space, tab, newline/return)
\S	Non-whitespace

^aDepending on the file, line breaks can be \r, \n, or \r\n. \r and \n match the same text

Table 14.2 Regex quantifiers

Quantifiers	Matching
X*	0 or more repetitions of X
X+	1 or more repetitions of X
X?	0 or 1 instances of X
X{m}	Exactly m instances of X
X{m,}	At least m instances of X
X{m,n}	Between m and n (inclusive) instances of X

- Matching . will match any character (as noted in Table 14.1).
- But if you want to match a period, you have to use \ (Table 14.2).

14.3.2.1 Visualization of Regular Expressions

To best visualize how regular expressions work, we will use a graphical interface. In a web search engine, you can search for “regex tester” to find one. These regular expression testers typically have two input fields:

1. A Test String input box which contains the text we want to extract terms from.
2. A Regular Expression input box in which we can enter a pattern capturing the terms of interest.

Below is an example.

(1) In the **Test String** box, paste the following plain text, which contains the names of a few common anti-hypertension blood pressure medicines:

“plain text
LISINOp^ril 40 MG PO Daily
captopril 6.25 MG PO TID

Table 14.3 Examples of regular expression in matching drug names

Pattern	Meaning
.	A period catches all characters (each one is a different color)
Pril	This only catches the phrase “pril”
.*pril	This catches 0 or more characters before “pril”
[a-z]*pril	This catches 0 or more characters, lower case, but does not match spaces or numbers etc.
[abcdefghijklmnopqrstuvwxyz]*pril	Notice that everything inside of the bracket is a character that we want to catch; it has the same results as the pattern above
[aA-zZ] + pril	This catches words with one or more character prior to ending in “pril”
[aA-zZ]{2,}	Pril this catches words with 2 or more characters prior to ending in “pril”
lisinoprillosartan	This catches “lisinopril” or “losartan”
\d	This catches numerical digits
\d{2}	This catches two numerical digits

```
I take lisinopril 40 mg PO Daily
April
pril
'''
```

- (2) In the **Regular Expression** box, test each one of the patterns in Table 14.3 and observe the difference in items that are highlighted.

14.3.2.2 Regular Expressions in Action Using Clinical Notes

See Jupyter Notebook: Part B—Fun with regular expressions

In this tutorial, we are going to use regular expressions to identify measurement concepts in a sample of Echocardiography (“echo”) reports in the MIMIC-III database. Echocardiogram is an ultrasound examination of the heart. The associated report contains many clinically useful measurement values, such as blood pressure, heart rate and sizes of various heart structures. Before any code, we should always take a look at a sample of the notes to see what our NLP task looks like:

```
'''plain text
PATIENT/TEST INFORMATION:
Indication: Endocarditis.
BP (mm Hg): 155/70
HR (bpm): 89
Status: Inpatient
Date/Time: [**2168-5-23**] at 13:36
Test: TEE (Complete)
Doppler: Full Doppler and color Doppler
```

```

Contrast: None
Technical Quality: Adequate
'''

```

This is a very well-formatted section of text. Let us work with a slightly more complex requirement (i.e., task), where we would like to extract the numerical value of the heart rate of a patient from these echocardiography reports.

A direct search using a lexicon-based approach as with NASH will not work, since numerical values can have a range. Instead, it would be desirable to specify a pattern for what a number looks like. Such pattern specifications are possible with regular expressions, which makes them extremely powerful. A single digit number is denoted by the notation `\d` and a two-digit number is denoted by `\d\d`. A search using this regular expression will return all occurrences of two-digit numbers in the corpus.

Accessing notes data

Again, we will need to query and download the Echocardiogram reports dataset from MIMIC's online Query Builder: <https://querybuilder-lcp.mit.edu>. Once logged in, paste the following SQL query code into the Home console and click "Execute Query".

```

'''MySQL
SELECT row_id, subject_id, hadm_id, text
FROM noteevents
WHERE CATEGORY = 'Echo'
LIMIT 10;
'''

```

All clinical notes in MIMIC are contained in the NOTEEVENTS table. The column with the actual text of the report is the TEXT column. Here, we are extracting the TEXT column from the first ten rows of the NOTEEVENTS table.

Click "Export Results" and save the exported file as "part_b.csv" file in the directory (i.e., folder) where you are running your local Jupyter notebook from. If you have the MIMIC-III database installed locally, you could query the dataset from the notebook locally as shown in tutorial "1. Direct search using curated lexicons"; simply replace the relevant SQL code.

Setting up in Jupyter Notebook

First, we import the necessary libraries for Python.

```

'''python
import os
import re
import pandas as pd
'''

```

Next, we import the echo reports dataset to your Jupyter notebook environment:

```

'''python
filepath = 'replace this with your path to your downloaded .csv file'
first_ten_echo_reports = pd.read_csv(filepath)
'''

```


Let us examine the result of our query. We will print out the first 10 rows.

```
“python
first_ten_echo_reports.head(10)
...”
```

Let us dig deeper and view the full content of the first report with the following line.

```
“python
report = first_ten_echo_reports[“text”][0]
print(report)
...”
```

Arrays start numbering at 0. If you want to print out the second row, you can type:

```
“python
report = first_ten_echo_reports[“text”][1]
...”
```

Make sure to rerun the block after you make changes.

NLP Exercise: Extracting heart rate from this note

We imported the regular expressions library earlier (i.e., `import re`). Remember, the variable “report” was established in the code block above. If you want to look at a different report, you can change the row number and rerun that block followed by this block.

```
“python
regular_expression_query = r’HR.*’
hit = re.search(regular_expression_query,report)
if hit:
    print(hit.group())
else:
    print(‘No hit for the regular expression’)
...”
```

We are able to extract lines of text containing heart rate, which is of interest to us. But we want to be more specific and extract the exact heart rate value (i.e., 85) from this line. Two-digit numbers can be extracted using the expression `\d\d`. Let us create a regular expression so that we get the first two-digit number following the occurrence of “HR” in the report.

```
“python
regular_expression_query = r’(HR).*(\d\d)’
hit = re.search(regular_expression_query,report)
if hit:
    print(hit.group(0))
    print(hit.group(1))
    print(hit.group(2))
else:
    print(‘No hit for the regular expression’)
...”
```

The above modification now enables us to extract the desired values of heart rate. Now let us try to run our regular expression on each of the first ten reports and print the result.

The following code uses a “for loop”, which means for the first 10 rows in “first_ten_echo_reports”, we will run our regular expression. We wrote the number 10 in the loop because we know there are 10 rows.

```

python
for i in range(10):
    report = first_ten_echo_reports["text"][i]
    regular_expression_query = r'(HR).*(\d\d)'
    hit = re.search(regular_expression_query,report)
    if hit:
        print('{} :: {}'.format(i,hit.group(2)))
    else:
        print('{} :: No hit for the regular expression')

```

We do not get any hits for reports 3 and 4. If we take a closer look, we will see that there was no heart rate recorded for these two reports.

Here is an example for printing out the echo report for 3; we can replace the 3 with 4 to print out the 4th report.

```

python
print(first_ten_echo_reports["text"][2])

```

14.3.3 Checking for Negations

See Jupyter Notebook: Part C—Sentence tokenization and negation detection

Great! Now you can find terms or patterns with brute force search and with regex, but does the context in which a given term occurred in a sentence or paragraph matter for your clinical task? Does it matter, for example, if the term was affirmed, negated, hypothetical, probable (hedged), or related to another unintended subject? Often times, the answer is yes. (See Coden et al. 2009 for a good discussion on the challenges of negation detection in a real-world clinical problem.)

In this section, we will demonstrate negation detection—the most commonly required NLP context interpretation step—by showing how to determine whether “pneumothorax” is reported to be present or not for a patient according to their Chest X-ray (CXR) report. First, we will spot all CXR reports that mention pneumothorax. Then we will show you how to tokenize (separate out) the sentences in the report document with NLTK (Perkins 2010) and determine whether the pneumothorax mention was affirmed or negated with Negex (Chapman et al. 2001).

Accessing notes data

Again, in Query Builder <https://querybuilder-lcp.mit.edu> (or local SQL database), run the following SQL query. Export 1000 rows and save results as instructed in prior examples and name the exported file as “part_c.csv”.

```

MySQL

```

```

SELECT row_id, subject_id, hadm_id, description, text
FROM NOTEEVENTS
WHERE description IN (
'P CHEST (PORTABLE AP) PORT', 'P CHEST PORT. LINE PLACEMENT
PORT', 'TRAUMA #3 (PORT CHEST ONLY)', 'OP CHEST (SINGLE VIEW) IN O.R.
PORT', 'P CHEST (PRE-OP AP ONLY) PORT',
'CHEST PORT. LINE PLACEMENT', 'CHEST PORTABLE LINE PLACEMENT', 'P CHEST
(SINGLE VIEW) PORT',
'CHEST AP ONLY', 'O CHEST SGL VIEW/LINE PLACEMENT IN O.R.', 'CHEST
(PORTABLE AP)',
'PO CHEST (SINGLE VIEW) PORT IN O.R.', 'O CHEST (PORTABLE AP) IN O.R.', 'CHEST
(PRE-OP AP ONLY)',
'CHEST (SINGLE VIEW)', 'P CHEST SGL VIEW/LINE PLACEMENT PORT')LIMIT 100;

```

Setting up in Jupyter Notebook

Again, we will first load the required Python libraries and import the CXR reports dataset we just queried and exported from Query Builder.

```

"""python
# Basic required libraries are:
import pandas as pd
import numpy as np
import random
import nltk

# import dataframe
filename = 'replace this with your path to your downloaded .csv file'
df_cxr = pd.read_csv(filename)

# How many reports do we have?
print(len(df_cxr))
"""

```

14.3.3.1 NLP Exercise: Is “Pneumothorax” Mentioned?

Next, let's get all the CXR reports that mention pneumothorax.

```

"""python
# First we need to have a list of terms that mean “pneumothorax” - let's call these commonly
known pneumothorax variations as our ptx lexicon:
ptx = ['pneumothorax', 'ptx', 'pneumothoraces']
# Simple spotter: Spot occurrence of a term in a given lexicon anywhere within a text document
or sentence:
def spotter(text, lexicon):
    text = text.lower()
    # Spot if a document mentions any of the terms in the lexicon
    # (not worrying about negation detection yet)
    match = [x in text for x in lexicon]
    if any(match) == True:
        mentioned = 1
    else:

```

```

        mentioned = 0
    return mentioned
# Let's now test the spotter function with some simple examples:
sent1 = 'Large left apical ptx present.'
sent2 = 'Hello world for NLP negation'
# Pneumothorax mentioned in text, spotter return 1 (yes)
spotter(sent1, ptx)
...
"""python
# Pneumothorax not mentioned in text, spotter return 0 (no)
spotter(sent2, ptx)
"""

```

Now, we can loop our simple spotter through all the “reports” and output all report IDs (i.e., `row_id`) that mention pneumothorax.

```

"""python
rowids = []
for i in df_cxr.index:
    text = df_cxr["text"][i]
    rowid = df_cxr["row_id"][i]
    if spotter(text, ptx) == 1:
        rowids.append(rowid)
print("There are " + len(rowids) + " CXR reports that mention pneumothorax.")
"""

```

14.3.3.2 NLP Exercise: Improving Spotting of a Concept in Clinical Notes

Unfortunately, medical text is notorious for misspellings and numerous non-standardized ways of describing the same concept. In fact, even for pneumothorax, there are many additional ways it could “appear” as a unique string of characters to a computer in free text notes. It is a widely recognized NLP problem that one set of vocabularies (lexicons) that work well on one source of clinical notes (e.g., from one particular Electronic Medical Record (EMR)) may not work well on another set of notes (Talby 2019). Therefore, a huge part of being able to recognize any medical concept with high sensitivity and specificity from notes is to have a robust, expert-validated vocabulary for it.

There are a few unsupervised NLP tools or techniques that can help with curating vocabularies directly from the corpus of clinical notes that you are interested in working with. They work by predicting new “candidate terms” that occur in similar contexts as a few starting “seed terms” given by a domain expert, who then has to decide if the candidate terms are useful for the task or not.

There also exist off-the-shelf, general-purposed biomedical dictionaries of terms, such as the UMLS (Bodenreider 2004) or the SNOMED_CT (Donnelly 2006). However, they often contain noisy vocabularies and may not work as well as you would like on the particular free text medical corpus you want to apply the vocabulary to. Nevertheless, they might still be useful to kickstart the vocabulary curation process

if you are interested in extracting many different medical concepts and willing to manually clean up the noisy terms.

Word2vec is likely the most basic NLP technique that can predict terms that occur in similar neighboring contexts. More sophisticated tools, such as the “Domain Learning Assistant” tool first published by Coden et al. (2012), integrate a user interface that allows more efficient ways of displaying and adjudicating candidate terms. Using this tool, which also uses other unsupervised NLP algorithms that perform better at capturing longer candidate phrases and abbreviations, a clinician is able to curate the following variations for pneumothorax in less than 5 minutes.

```

python
ptx = ['pneumothorax', 'ptx', 'pneumothoraces', 'pnuemothorax', 'pnumothorax', 'pntx',
'penumothorax', 'pneomothorax', 'pneumonthorax', 'pnemothorax', 'pneumothoraxes',
'pneumpthorax', 'pneumonthorax', 'pneumothorx', 'pneumothrax', 'pneumothroax', 'pneu-
mothraces', 'pneunothorax', 'enlarging pneumo', 'pneumothoroax', 'pneuothorax']

```

Pause for thought

Now we can spot mentions of relevant terms, but there are still some other edge cases you should think about when matching terms in free text:

1. Are spaces before and/or after a term important? Could they alter the meaning of the spot? (e.g. should [pneumothorax] and hydro[pneumothorax] be treated the same?)
2. Is punctuation before and/or after a term going to matter?
3. Do upper or lower cases matter for a valid match? (The above simple spotter turns all input text into lower letter case so in effect ignores letter cases when searching for a match.)

What could you do to handle edge cases?

1. Use regular expression when spotting the terms. You can pick what characters are allowed on either ends of a valid matched term, as well as upper or lower letter cases.
2. Add some common acceptable character variations, such as punctuation or spaces on either end for each term in the lexicon (e.g., “ptx/”).

14.3.3.3 NLP Exercise: Negation Detection at Its Simplest

Obviously, not all these reports that mention pneumothorax signify that the patients have the condition. Often times, if a term is negated, then it occurs in the same sentence as some negation indication words, such as “no”, “not”, etc. Negation at its simplest would be to detect such co-occurrence in the same sentence.

```

python
# e.g. Pneumothorax mentioned in text but negated, a simple spotter would still return 1 (yes)
sent3 = 'Pneumothorax has resolved.'
spotter(sent3, ptx)
python
# e.g. Simply spotting negation words in the same sentence:

```

```
neg = ['no','never','not','removed', 'ruled out', 'resolved']
spotter(sent3, neg)
...
```

However, there would be other edge cases. For example, what if “no” is followed by a “but” in a sentence? e.g. “There is no tension, but the pneumothorax is still present.”

Luckily, smarter NLP folks have already written some negation libraries to spot negated mentions of terms for us that work on these more complicated cases. However, first, we will need to learn how to pre-process the input text document into sentences (i.e. sentence tokenization).

14.3.3.4 NLP Exercise: Sentence Tokenization with NLTK

Splitting up the sentence before running negation is usually required with most negation libraries. Here is a link to instructions for installing NLTK: <https://www.nltk.org/install.html>.

```
...python
# Lets print a random report from df_cxr
report = df_cxr.text[random.randint(0,100)]
print(report)
...
```

There are two main ways to tokenize sentences with NLTK. If you do not need to save the sentence offsets (i.e., where the sentence started and ended in the original report), then you can just use “sent_tokenize”.

```
...python
# Simplest: Tokenize the sentences with sent_tokenize from NLTK
from nltk.tokenize import sent_tokenize
sents = sent_tokenize(report.replace('\n', ' ')) # removing new line breaks
# Print out list of sentences:
sent_count = 0
for s in sents:
    print("Sentence " + str(sent_count) + ":")
    print(s)
    print()
    sent_count = sent_count + 1
...
```

Alternatively, tokenize with “PunktSentenceTokenizer” from NLTK if you want to keep track of character offsets of sentences.

```
...python
from nltk.tokenize.punkt import PunktSentenceTokenizer
sent_count = 0
for s_start, s_finish in PunktSentenceTokenizer().span_tokenize(report):
    print("Sentence " + str(sent_count) + ": " + str([s_start, s_finish]))
    #important not to accidentally alter the character offsets with .replace()
    print(report[s_start:s_finish].replace('\n', ' '))
    print()
    sent_count = sent_count + 1
...
```

14.3.3.5 NLP Exercise: Using an Open-Source Python Library for Negation—Negex

Next, let us finally introduce “Negex”, an open source Python tool for detecting negation. It has limitations, but it would be easier to build and improve on top of it than to write something from scratch. You can download negex.python from: <https://github.com/mongoose54/negex/tree/master/negex.python>.

To run Negex in a Jupyter Notebook, the required “negex.py” and “negex_triggers.txt” files are already in this chapter’s Github repository. Run the following Python code to import Negex to your notebook environment:

```
“python
import negex
# Read the trigger negation rule file that comes with negex
rfile = open(r'negex_triggers.txt')
irules = negex.sortRules(rfile.readlines())
rfile.close()
”
```

Again, let’s start with a simple example using Negex to show its basic function.

```
“python
sent = “There is no evidence of ptx.”
ptx = ['pneumothorax', 'ptx', 'pneumothoraces', 'pnuemothorax', 'pnumothorax', 'pntx',
'penumothorax', 'pneomothorax', 'pneumonthorax', 'pnemothorax', 'pneumothoraxes',
'pneumpthorax', 'pneuomthorax', 'pneumothorx', 'pneumothrax', 'pneumothroax', 'pneu-
mothraces', 'pneunothorax', 'enlarging pneumo', 'pneumothoroax', 'pneuothorax']
tagger = negex.negTagger(sentence = sent, phrases = ptx, rules = irules, negP=False)
negation = tagger.getNegationFlag()
negation
”
```

Now, we will try Negex on a CXR report that mentions pneumothorax. We have to tokenize the sentences first and see whether a given sentence mentions pneumothorax or not before we apply Negex for negation detection. If you apply Negex to a sentence that does not mention the term of interest, then it will return “affirmed”, which is definitely not the desired output.

```
“python
# Subset reports from df_cxr that mention pneumothorax:
df_ptx = df_cxr.loc[df_cxr['row_id'].isin(rowids)].copy()
# Grab the first CXR report in the df_ptx dataset as an example:
note = df_ptx.text[0]
# Show the relevant CXR report for the analysis:
print(note)
”
“python
# Tokenize the sentences in the note:
sents = sent_tokenize(note.replace('\n', ' ')) # replacing new line breaks (not essential)
# Applying spotter function to each sentence:
neg_output = []
count = 0
for sent in sents:
    # Apply Negex if a term in the ptx lexicon is spotted
    if spotter(sent,ptx) == 1:
```

```

tagger = negex.negTagger(sentence = sent, phrases = ptx, rules = irules, negP=False)
negation = tagger.getNegationFlag()
neg_output.append(negation)
print("Sentence " + str(count) + ":\n" + sent + "\nNegex output: " + negation + "\n")
count = count + 1
...

```

However, sometimes, multiple sentences from a note can mention a concept of interest. In the case of pneumothorax, a sentence at the start of the report could mention that the patient has a history of pneumothorax. Then the radiologist could write that it has resolved in another sentence near the end of the report. One way to deal with this is to store the negation results for all sentences that mention pneumothorax in a list and do some post-processing with it later.

```

“python
# Example: Now loop through the first 1000 notes in df_ptx
# (otherwise it would take a while to run on all)
results_ptx = df_ptx[:1000].copy()
for i in results_ptx.index:
    note = results_ptx.text[i]
    sents = sent_tokenize(note.replace('\n', ' '))
    neg_output = []
    rel_sents = []
    for sent in sents:
        # If a sentence mentions pneumothorax
        if spotter(sent,ptx) == 1:
            tagger = negex.negTagger(sentence = sent, phrases = ptx, rules = irules,
negP=False)
            negation = tagger.getNegationFlag()
            neg_output.append(negation)
            rel_sents.append(sent)
            print("Sentence: " + sent + " | " + "Negex output: " + negation + "\n")
    # Add a column in the df_ptx dataframe to "structure" the extracted ptx data
    results_ptx.loc[i, 'ptx_prediction' ] = ' | '.join(neg_output)
    # Add a column in the df_ptx dataframe to store the relevant sentences
    # that mentioned ptx
    results_ptx.loc[i, 'ptx_sentences' ] = ' | '.join(rel_sents)
# Don't forget to export your now "structured" results!!!
# tab delimited
results_ptx.to_csv("ptx_results.txt", sep = '\t', encoding='utf-8', index=False)
# as csv:
df_ptx.to_csv("ptx_results.csv", index=False)
# Show a few rows in the results dataframe:
results_ptx.head(10)
...

```

Some observations

You can see that even Negex is not perfect at its single sentence level prediction. Here, it does not pick up hypothetical mentions of pneumothorax; it interpreted “t/o ptx” as affirmed. However, at the whole report level, later sentences might give a more correct negated prediction.

14.3.4 Putting It All Together—Obesity Challenge

See Jupyter Notebook: Part D—Obesity challenge

Let's consider a quick real-world challenge to test what we have learned. Unlike many medical concepts, obesity is one that has a fairly well-established definition. It may not be always correct (Ahima and Lazar 2013), but the definition is clear and objective: If a patient's BMI is above 30.0, they are considered obese.

However, it is worthwhile to be aware that many other clinical attributes in medical notes that are not as clear cut. For example, consider the i2b2 challenge on smoking detection (I2B2 2006). How does one define “is smoker”? Is a patient in a hospital who quit smoking three days ago on admission considered a non-smoker? What about a patient in primary care clinic who quit smoking a few weeks ago? Similarly, how does one define “has back pain”, “has, non-adherence”, and so on? In all of these cases, the notes may prove to be the best source of information to determine the cohort inclusion criteria for the particular clinical study. The NLP techniques you have learned in this chapter should go a long way to help to structure the “qualitative” information in the notes into quantitative tabular data.

The goal of the obesity challenge is to see how accurately you can identify patients who are obese from their clinical notes. In the interest of an easy-to-compute gold standard for our test (i.e. instead of manually annotating a gold standard data for e.g. “has back pain” ourselves), we picked “obesity” so that we can just calculate the patient's BMI from the height and weight information in MIMIC's structured data.

For the Obesity Challenge exercise:

1. We will generate a list of 50 patients who are obese and 50 who are not.
2. Then, we are going to pull all the notes for those patients.
3. Using the notes, you need to figure out which patients are obese or not.
4. At the end, the results will be compared with the gold standard to see how well you did.

Accessing notes data

The SQL query for this exercise is fairly long so it is saved in a separate text file called “part_d_query.txt” in this chapter's Github repository.

Copy the SQL command from the text file, then paste and run the command in Query Builder (<https://querybuilder-lcp.mit.edu>). Rename the downloaded file as “obese-gold.csv”. Make sure the file is saved in the same directory as the following notebook.

Setting up in Jupyter Notebook

As usual, we start with loading the libraries and dataset we need:

```
“python
# First off - load all the python libraries we are going to need
import pandas as pd
import numpy as np
```

```

...
"""python
notes_filename = 'replace this with your path to your downloaded .csv file'
obesity_challenge = pd.read_csv(notes_filename)
...

```

The “obesity_challenge” dataframe has one column, “obese”, that defines patients who are obese (1) or normal (0). The definition of obese is $BMI \geq 30$, overweight is $BMI \geq 25$ and < 30 , and normal is $BMI \geq 18.5$ and < 25 . We will create the notes and the gold standard data frames by subsetting “obesity_challenge”.

```

"""python
notes = obesity_challenge[['subject_id', 'text']]
gold = obesity_challenge[['subject_id', 'obese']]
...

```

NLP Exercise: Trivial term spotting as baseline

For this exercise, we are going to begin with trivial term spotting (which you have encountered in NLP exercise Part A) with only one obesity-related term at baseline. You, however, are going to work on editing and writing more complex, interesting and effective NLP code!

```

"""python
# Here is the list of terms we are going to consider “good” or associated with what we want to
find, obesity.
terms = ['obese']
...

```

Using the trivial term spotting approach, we’re going to quickly scan through our note subset and find people where the obesity-related term(s) appears.

```

"""python
# Now scan through all of the notes. Do any of the terms appear? If so stash the note
# id for future use
matches = []
for index, row in notes.iterrows():
    if any(x in row['text'] for x in terms):
        matches.append(row['subject_id'])
print("Found " + str(len(matches)) + " matching notes.")
...

```

We will assume all patients are initially “unknown” and then for each of the true matches, we’ll flag them. Note: We are using 1 for obese, 0 for unknown and -1 for not-obese. For our code at baseline, we have not implemented any code that sets a note to -1 , which can be the first improvement that you make.

```

"""python
# For the patients in those notes, set “obese” true (1) in a the results
myscores = gold.copy()
myscores['obese'] = 0 # This sets them all to unknown
for subject_id in matches:
    myscores.loc[myscores["subject_id"] == subject_id, 'obese'] = 1
...

```

And finally, the following code would score the results:

```

"""python

```

```

# Compute your score
skipped = 0
truepositive = 0
falsepositive = 0
truenegative = 0
falsenegative = 0
for index, row in myscores.iterrows():
    if row['obese'] == 0:
        skipped = skipped + 1
    else:
        if row['obese'] == 1 and gold.loc[index]['obese'] == 1:
            truepositive = truepositive + 1
        elif row['obese'] == -1 and gold.loc[index]['obese'] == -1:
            truenegative = truenegative + 1
        elif row['obese'] == 1 and gold.loc[index]['obese'] == -1:
            falsepositive = falsepositive + 1
        elif row['obese'] == -1 and gold.loc[index]['obese'] == 1:
            falsenegative = falsenegative + 1
print ("Skipped:\t" + str(skipped))
print ("True Pos:\t" + str(truepositive))
print ("True Neg:\t" + str(truenegative))
print ("False Pos:\t" + str(falsepositive))
print ("False Neg:\t" + str(falsenegative))
print ("SCORE:\t" + str(truepositive + truenegative - falsepositive - falsenegative))
...

```

NLP Exercise: can you do better?

We got a score of 19 (out of a possible 100) at baseline. Can you do better?

Here are a few NLP ideas that can improve the score:

- Develop a better lexicon that captures the various ways in which obesity can be mentioned. For example, abbreviations are often used in clinical notes.
- Checking whether the mentioned term(s) for obesity is further invalidated or not. For example, if “obese” is mentioned in “past”, “negated”, “family history” or other clinical contexts.
- Use other related information from the notes, e.g. extract height and weight values with regular expressions and compute the patient’s BMI or directly extract the BMI value from the notes.
- Tweak the regular expressions to make sure additional cases of how terms can be mentioned in text are covered (e.g. plurals, past tenses (if they do not change the meaning of the match)).

14.4 Summary Points

1. Spotting a “name-entity” is as simple as writing code to do a search-and-find in raw text.

2. However, to identify a semantic concept of interest for clinicians, we need to account for variations through which the concept may be described in clinical notes. This may include misspellings, rewording, and acronyms; it may also require text pattern recognition, where use of regular expression can be useful.
3. In general, a more robust vocabulary that recognizes a concept of interest in many forms will help you spot the concept with higher sensitivity.
4. After spotting a term (i.e., name-entity) of interest in unstructured text, it may be important to interpret its context next to improve specificity.
5. Negation detection is one type of NLP context interpretation. There are many others and the importance of each depends on your task.
6. Negation detection at its simplest may be the detection of a negation-related term (e.g., “no”) in the same sentence. More complex NLP libraries, such as Negex and sPacy, can help you do a better job in more complicated cases (e.g., “but”).
7. At the whole document level, a term or concept may be mentioned in multiple sentences in different contexts. It is up to experts to determine how to put together all the information to give the best overall prediction for the patient.

14.5 Limitations

- We are not taking advantage of deep parses (i.e., using full computer generated “sentence diagrams”). With well-written, grammatically-correct text you may do better tracking the semantic assertions (e.g., direct statements of fact in the text) in the notes; however, this can break down quickly and fail easily in the presence of more informal language.
- The tools we are using depend on some understanding of word structure; thus, German agglutinative nouns can be a challenge for automated processing as assumptions about spaces separating tokens, as can languages that do not use spaces (e.g., many South East Asian language families).
- Very large collections of text can take a long time to run with these methods. Fortunately, clinical notes are not “large” in the way that other corpuses are (e.g., Twitter can run on the order of billions of tweets for a fairly small time frame), so most of these collections will run fine on modest hardware, but they may take several hours on a modern laptop.
- Regular expressions may be brittle; sets that work well on one dataset may fail on another due to different standards of punctuation, formatting, etc.
- We have not taken advantage of the structure of the clinical notes (e.g., a past medical history section) when available. This kind of context can make many tasks (such as identifying if a disease IS a family history mention) easier, but it can be a challenge identifying them especially in more free form notes such as the ones you find in an ICU.
- Lastly there are cases where substantial domain knowledge or judgement calls are required. For example, *“She denies insulin non-compliance but reports that her VNA asked her to take insulin today and she only drew air into the syringe without*

fluid” could be interpreted as non-compliant as the patient knowingly skipped doses (and subsequently was admitted to the ICU with diabetic ketoacidosis, a complication due to not getting insulin). Or, this sentence could be judged to be compliant as the patient “tried”. Such judgement calls are beyond the scope of any computer and depend on what the information is going to be used for in downstream analytics.

14.6 Conclusion

We provide an introduction to NLP basics in the above chapter. That being said, NLP is a field that has been actively researched for over half a century, and for well written notes, there are many options for code or libraries that can be used to identify and extract information.

A comprehensive overview of approaches used in every aspect of natural language processing can be found in Jurafsky and Martin (2009). Information extraction, including named-entity recognition and relation extraction from text, is one of the most-studied areas in NLP (Meystre et al. 2008), and the most recent work is often showcased in SemEval tasks (e.g., SemEval 2018).

For a focus on clinical decision support, Demner-Fushman et al. (2009) provides a broad discussion. Deep learning is an increasingly popular approach for extraction, and its application to electronic health records is addressed in Shickel et al. (2017).

Nonetheless, the basics outlined in this chapter can get you quite far. The text of medical notes gives you an opportunity to do more interesting data analytics and gain access to additional information. NLP techniques can help you systematically transform the qualitative unstructured textual descriptions into quantitative attributes for your medical analysis.

References

- Ahima, R. S., & Lazar, M. A. (2013). The health risk of obesity—better metrics imperative. *Science*, 341(6148), 856–858.
- Aho, A. V., & Ullman, J. D. (1995). *Foundations of computer science*. Chapter 10. Patterns, Automata, and Regular Expressions. Computer Science Press.
- Bodenreider, O. (2004). The unified medical language system (UMLS): Integrating biomedical terminology. *Nucleic Acids Research*, 32(suppl_1), D267–D270.
- Chapman, W. W., et al. (2001). A simple algorithm for identifying negated findings and diseases in discharge summaries. *Journal of Biomedical Informatics*, 34(5), 301–310.
- Coden, A., et al. (2009). Automatically extracting cancer disease characteristics from pathology reports into a Disease Knowledge Representation Model. *Journal of Biomedical Informatics*, 42(5), 937–949.
- Coden, A., et al. (2012). SPOT the drug! An unsupervised pattern matching method to extract drug names from very large clinical corpora. In *2012 IEEE second international conference on healthcare informatics, imaging and systems biology*. IEEE.

- Demner-Fushman, D., Chapman, W. W., & McDonald, C. J. (2009). What can natural language processing do for clinical decision support? *Journal of Biomedical Informatics*, 42(5), 760–772.
- Donnelly, K. (2006). SNOMED-CT: The advanced terminology and coding system for eHealth. *Studies in health technology and informatics*, 121, 279.
- Informatics for Integrating Biology & the Bedside. i2b2. (2006). www.i2b2.org/NLP/DataSets/Main.php. Smoking Challenge.
- Jurafsky, D., & Martin, J. H. (2009). *Speech and language processing* (3rd ed.). Upper Saddle River, NJ, USA: Prentice-Hall Inc.
- Kleene, S. C. (1951). Representation of events in nerve nets and finite automata. Technical report RM-704, RAND Corporation. RAND Research Memorandum.
- Meystre, S. M., Savova, G. K., Kipper-Schuler, K. C., & Hurdle, J. F. (2008). Extracting information from textual documents in the electronic health record: A review of recent research. *Yearbook of Medical Informatics*, 17(01), 128–144.
- Perkins, J. (2010). *Python text processing with NLTK 2.0 cookbook*. Packt Publishing Ltd.
- SemEval-2018. (2018). Tasks < SemEval-2018. <http://alt.qcri.org/semeval2018/index.php?id=tasks>.
- Shickel, B., Tighe, P. J., Bihorac, A., & Rashidi, P. (2017). Deep EHR: A survey of recent advances in deep learning techniques for electronic health record (EHR) analysis. *IEEE Journal of Biomedical and Health Informatics*, 22(5), 1589–1604.
- Talby, D. (2019, March 7). *Lessons learned building natural language processing systems in health care*. O'Reilly Media. www.oreilly.com/ideas/lessons-learned-building-natural-language-processing-systems-in-health-care.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.