AI for Medicine Course 3 Week 1 lecture notebook

Using BioC format and the NegBio Library

Welcome to this lecture notebook! You'll be exploring some of the uses of the `NegBio` library, a tool for biomedical text mining, which you will use in the graded assignment at the end of the week.

You'll be using the same dataset as in the assignment, so this is a good opportunity to become more familiar with it.

- This dataset consists of 1,000 X-ray reports that have been manually labeled by a board-certified radiologist.
- The reports indicate the presence or absence of several different pathologies.
- You'll also have access to the extracted "Report Impression" section of each report, which is the summary provided for each X-ray.

Import Pandas and Load Dataset

In [1]:
```
import pandas as pd

# Read the data from file
df = pd.read_csv("stanford_report_test.csv")

# Check the num of rows, columns
print(f"dataset has shape: {df.shape}")
df.head()
```

```
dataset has shape: (1000, 18)
```

Out[1]:

	SimpleTestReportID	Report Impression	No Finding	Enlarged Cardiomediastinum	Cardiomegaly	Lung Lesion	Airspace Opacity	Edema	Consolidation	Pneumonia	Atelectasis	F
0	1.0	\n \n1.mild pulmonary edema, and cardiomegaly....	NaN	NaN	1.0	NaN	NaN	1.0	-1.0	NaN	1.0	
1	2.0	\n \n1.unremarkable cardiomediastinal silhouet...	NaN	0.0	NaN	NaN	1.0	NaN	0.0	-1.0	NaN	
2	3.0	\n1. lines and tubes are unchanged in position...	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN	
		\n1.										

In [2]:
```
# Get a better view of the report impression column
for i in range(3):
    print("################################")
    print(f"Report number: {i+1}")
    print(df.loc[i, 'Report Impression'])
```

```
################################
Report number: 1


1.mild pulmonary edema, and cardiomegaly.  trace pleural fluid
effusions.

2.low lung volumes with minimal basilar atelectasis.

3.no new focal consolidation.

4.interval placement of defibrillation pads.


################################
Report number: 2


1.unremarkable cardiomediastinal silhouette

2.diffuse reticular pattern, which can be seen with an atypical
infection or chronic fibrotic change.  no focal consolidation.

3.no pleural effusion or pneumothorax

4.mild degenerative changes in the lumbar spine and old right rib
fractures.


################################
Report number: 3

1. lines and tubes are unchanged in position.
2. increasing retrocardiac opacity and left midlung zone opacity.
3. there is a deep left costophrenic sulcus which is increased when
compared with prior films. no definite evidence of left
pneumothorax. clinical correlation is recommended. if clinically
indicated, consider film in expiration or decubitus views.
4. the icu team was informed of these results at 10 am on 05_02_2005.
```

Introducing BioC

Let's get started by looking at the `BioC` module. You'll be using `BioC` to convert your clinical data into a standard format that can be leveraged on more specialized libraries. This module is used for many other NLP tasks as well, such as serialization or deserialization of data. You can read more about it here.

For your purposes, you're interested in the `BioCCollection` object, which represents a collection of documents for a project. The collection might be an entire corpus, or a partial one.

In [3]:
```
import bioc
```

```
collection = bioc.BioCCollection()
print(f"attributes with value: \n\n{collection.__dict__}\n")
print(f"methods and attributes: \n\n{dir(collection)}\n")
print(f"documents within collection: {collection.documents}")
```

```
attributes with value:

{'encoding': 'utf-8', 'version': '1.0', 'standalone': True, 'source': '', 'date': '2021-04-09', 'key': '', 'infons': {}, 'do
cuments': []}

methods and attributes:

['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt_
_', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__red
uce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'add_document', 'clear_inf
ons', 'date', 'documents', 'encoding', 'infons', 'key', 'source', 'standalone', 'version']

documents within collection: []
```

Preparing the Text for BioC

When working with collections, you're mostly interested in the documents attribute and the `add_document()` method.

The `BioC` module gives you a standard format that allows you to apply other, more specialized libraries. Before seeing `BioC` in action, let's introduce `NegBio`, a tool that distinguishes negative or uncertain findings in radiology reports. It accomplishes this by using patterns on universal dependencies, instead of using rule-based methods. If you'd like to know more, check out the official github repo, or the official documentation.

You'll be using the `NegBioSSplitter` object to split your text into sentences. However, in order to do this, you'll first need to convert your text into a format that `BioC` supports. For this you'll use the `text2bioc()` function, which transforms the text into a `BioC` XML file. You can go even further and convert the text into documents with the `text2document()` function.

In [4]:
```
from negbio.pipeline.ssplit import NegBioSSplitter
from negbio.pipeline import text2bioc

splitter = NegBioSSplitter()
for i, report in enumerate(df["Report Impression"]):
        document = text2bioc.text2document(str(i), report)
        document = splitter.split_doc(document)
        collection.add_document(document)
```

In [5]:
```
collection.documents
```

Out[5]:
```
[BioCDocument[id=0,infons=[],passages=[BioCPassage[offset=0,text='\n \n1.mild pulmona ... lation pads.  \n \n',infons=[],
sentences=[BioCSentence[offset=0,text='\n \n1.mild pulmona ... and cardiomegaly.',infons=[],annotations=[],relations=
[],],BioCSentence[offset=46,text='trace pleural fluid \neffusions.',infons=[],annotations=[],relations=[],],BioCSentence
[offset=80,text='2.low lung volume ... ilar atelectasis.',infons=[],annotations=[],relations=[],],BioCSentence[offset=13
5,text='3.no new focal consolidation.',infons=[],annotations=[],relations=[],],BioCSentence[offset=168,text='4.interval p
lacem ... ibrillation pads.',infons=[],annotations=[],relations=[],]],annotations=[],relations=[],]],annotations=[],relat
ions=[],],
 BioCDocument[id=1,infons=[],passages=[BioCPassage[offset=0,text='\n \n1.unremarkable ... b \nfractures. \n \n',infons=
[],sentences=[BioCSentence[offset=0,text='\n \n1.unremarkable ...  fibrotic change.',infons=[],annotations=[],relations=
[],],BioCSentence[offset=153,text='no focal consolidation.',infons=[],annotations=[],relations=[],],BioCSentence[offset=1
79,text='3.no pleural effu ... t rib \nfractures.',infons=[],annotations=[],relations=[],]],annotations=[],relations=
[],]],annotations=[],relations=[],],
 BioCDocument[id=2,infons=[],passages=[BioCPassage[offset=0,text='\n1. lines and tub ... m on 05_02_2005.\n',infons=[],se
ntences=[BioCSentence[offset=0,text='\n1. lines and tub ... nged in position.',infons=[],annotations=[],relations=[],],Bi
oCSentence[offset=47,text='2. increasing ret ... ung zone opacity.',infons=[],annotations=[],relations=[],],BioCSentence
[offset=113,text='3. there is a dee ... with prior films.',infons=[],annotations=[],relations=[],],BioCSentence[offset=20
8,text='no definite evide ... eft\npneumothorax.',infons=[],annotations=[],relations=[],],BioCSentence[offset=251,text='c
linical correlation is recommended.',infons=[],annotations=[],relations=[],],BioCSentence[offset=288,text='if clinically
\nind ...  decubitus views.',infons=[],annotations=[],relations=[],],BioCSentence[offset=361,text='4. the icu team w ...
am on 05_02_2005.',infons=[],annotations=[],relations=[],]],annotations=[],relations=[],],
```

Interpreting the Documents

Now your `BioC` collection has been filled with documents, but the output is very hard to read. Let's break it down a little more.

In [6]:
```
len(collection.documents)
```

Out[6]:
```
1000
```

Looks like you have a document for each report impression. But what's stored inside each document? Let's check the first one.

In [7]:
```
collection.documents[0]
```

Out[7]:
```
BioCDocument[id=0,infons=[],passages=[BioCPassage[offset=0,text='\n \n1.mild pulmona ... lation pads.  \n \n',infons=[],sent
ences=[BioCSentence[offset=0,text='\n \n1.mild pulmona ... and cardiomegaly.',infons=[],annotations=[],relations=[],],BioCSe
ntence[offset=46,text='trace pleural fluid \neffusions.',infons=[],annotations=[],relations=[],],BioCSentence[offset=80,text
='2.low lung volume ... ilar atelectasis.',infons=[],annotations=[],relations=[],],BioCSentence[offset=135,text='3.no new fo
cal consolidation.',infons=[],annotations=[],relations=[],],BioCSentence[offset=168,text='4.interval placem ... ibrillation
pads.',infons=[],annotations=[],relations=[],]],annotations=[],relations=[],]],annotations=[],relations=[],]
```

Each document has an attribute called "passages" in which the sentences are stored. Notice that `passages` is a list, but for this case it will only have one element:

In [8]:
```
collection.documents[0].passages[0].sentences
```

Out[8]:
```
[BioCSentence[offset=0,text='\n \n1.mild pulmona ... and cardiomegaly.',infons=[],annotations=[],relations=[],],
 BioCSentence[offset=46,text='trace pleural fluid \neffusions.',infons=[],annotations=[],relations=[],],
 BioCSentence[offset=80,text='2.low lung volume ... ilar atelectasis.',infons=[],annotations=[],relations=[],],
 BioCSentence[offset=135,text='3.no new focal consolidation.',infons=[],annotations=[],relations=[],],
 BioCSentence[offset=168,text='4.interval placem ... ibrillation pads.',infons=[],annotations=[],relations=[],]]
```

Each sentence stores information about the text, offset, relations and annotations. Let's check the sentences saved in the first document of our collection:

In [9]:
```
for i,s in enumerate(collection.documents[0].passages[0].sentences):
    print(f"sentence number {i + 1}: {s.text}\n")
    print("################################\n")
```

```
sentence number 1:

1.mild pulmonary edema, and cardiomegaly.

################################

sentence number 2: trace pleural fluid
effusions.

################################

sentence number 3: 2.low lung volumes with minimal basilar atelectasis.

################################

sentence number 4: 3.no new focal consolidation.
```

```
##############################

sentence number 5: 4.interval placement of defibrillation pads.

##############################
```

Cleaning up with the clean() function

Notice how the first report impression, which had two sentences, was split successfully. However, the newlines have not been trimmed. The `clean()` function from the previous lecture notebook will come in handy here. Let's bring it back out of the toolbox and apply it in this notebook!

In [10]:
```
import re
def clean(sentence):
    lower_sentence = sentence.lower()
    corrected_sentence = re.sub('and/or', 'or', lower_sentence)
    corrected_sentence = re.sub('(?<=[a-zA-Z])/(?=[a-zA-Z])', ' or ', corrected_sentence)
    clean_sentence = corrected_sentence.replace("..", ".")
    punctuation_spacer = str.maketrans({key: f"{key} " for key in ".,"})
    clean_sentence = clean_sentence.translate(punctuation_spacer)
    clean_sentence = ' '.join(clean_sentence.split())
    return clean_sentence
```

Exercise

Now that you've spent some time exploring how the `NegBio` library works, let's try it out on your data.

You'll determine whether a given report impression can tell you if a patient has an existing condition, while taking into account whether there was negation or uncertainty in the findings. For this task, you'll use these predetermined categories:

In [11]:
```
CATEGORIES = ["Cardiomegaly", "Lung Lesion", "Airspace Opacity", "Edema",
              "Consolidation", "Pneumonia", "Atelectasis", "Pneumothorax",
              "Pleural Effusion", "Pleural Other", "Fracture"]
```

Import NegBio Dependencies

Next you'll import everything you need for this task. Don't be alarmed by the declared paths below the imports! They're just mapping the path to various files that `NegBio` relies on.

In [12]:
```
from pathlib2 import Path
from negbio.main_chexpert import pipeline
from negbio.pipeline.parse import NegBioParser
from negbio.chexpert.stages.load import NegBioLoader
from negbio.chexpert.stages.extract import NegBioExtractor
from negbio.chexpert.stages.classify import ModifiedDetector
from negbio.chexpert.stages.aggregate import NegBioAggregator
from negbio.pipeline.ptb2ud import NegBioPtb2DepConverter, Lemmatizer

PARSING_MODEL_DIR = "~/.local/share/bllipparser/GENIA+PubMed"
CHEXPERT_PATH = "NegBio/negbio/chexpert/"
MENTION_PATH =f"{CHEXPERT_PATH}phrases/mention"
UNMENTION_PATH = f"{CHEXPERT_PATH}phrases/"
NEG_PATH = f'{CHEXPERT_PATH}patterns/negation.txt'
PRE_NEG_PATH = f'{CHEXPERT_PATH}patterns/pre_negation_uncertainty.txt'
POST_NEG_PATH = f'{CHEXPERT_PATH}patterns/post_negation_uncertainty.txt'
```

The encoding of information within these files is beyond the scope of this notebook, but if you're really curious about the contents you could do something like this to see more:

```
!cat $NEG_PATH
```

In [13]:
```
!cat $NEG_PATH
```

```
# No definite XXX
({} > {} {lemma:/definite/}) > {dependency:/neg/} {}

# No obvious XXX
({} > {} {lemma:/obvious/}) > {dependency:/neg/} {}


{} > {dependency:/amod|nsubj/} {lemma:/normal|unremarkable/}
{} < {dependency:/amod|nsubj/} {lemma:/normal|unremarkable/}
({} > {} {}) < {dependency:/nsubj|dobj/} {lemma:/unremarkable|normal/}
{} < {} ({} > {dependency:/amod/} {lemma:/normal|unremarkable/})
{} < {} ({} < {dependency:/nsubj/} {lemma:/normal|unremarkable/})
{} < {dependency:/conj:no/} {}
{} < {} ({} < {dependency:/conj:or/} ({} > {} {lemma:/no/}))
{} < {dependency:/nsubj/} ({lemma:/limit.*/} > {} {lemma:/upper/} & > {dependency:/nmod:of/} {lemma:/normal/} & > {depend
ency:/case/} {lemma:/at|within/})
{} < {} ({dependency:/exclude/} < {} ({} > {} {lemma:/no/}))


({lemma:/silhouette/} > {} {}) < {dependency:/dobj|nsubj/} {lemma:/obscure/}
```

Running this process for the entire dataset is very slow (~1.5 hr on a fast laptop!) so let's slice it to showcase how `NegBio` works. Let's start with 50 random observations.

In [14]:
```
sampled_df = df.sample(50)
```

Also, let's recreate the code from the beginning of the notebook as a function, including the `clean()` function as well.

In [15]:
```
def get_bioc_collection(df):
    collection = bioc.BioCCollection()
    splitter = NegBioSSplitter()
    for i, report in enumerate(df["Report Impression"]):
        document = text2bioc.text2document(str(i), clean(report))
        document = splitter.split_doc(document)
        collection.add_document(document)
    return collection
```

Here, you'll repeat your process from earlier by converting the report impression strings into a `BioC` XML format which `NegBio` can read.

In [16]:
```
collection = get_bioc_collection(sampled_df)
```

Now let's instantiate `NegBio` 's lemmatizer.

The process of lemmatization refers to returning the dictionary form of a word (or lemma) by removing inflectional endings. It's very cool and you can read more about it here.

In [17]:
```
lemmatizer = Lemmatizer()
```

Next you'll instantiate `NegBio`'s converter to convert from parse tree to universal dependencies. This is done using the Stanford converter, which you can find more information about here.

The parse tree used here is the Penn Treebank. In general terms, a treebank is an annotated text corpus that includes analysis beyond part-of-speech tagging. They've become very valuable resources to NLP research in recent years.

Universal dependencies, or UD, provide a powerful framework for annotating grammar across different languages. Read more about them here.

In [18]:
```
ptb2dep = NegBioPtb2DepConverter(lemmatizer, universal=True)
```

You've already seen the splitter in action before, so you can skip it.

In [19]:
```
ssplitter = NegBioSSplitter(newline=True)
```

Now you'll instantiate the parser and the loader.

Under the hood, you're using the BLIPP reranking parser, which is a statistical natural language parser.

The loader, as you might imagine, loads the reports into memory.

Over all of this, the chexpert-labeler is used. This labeler extracts observations from radiology reports specifically, and can provide a vocabulary appropriate to the clinical context.

In [20]:
```
parser = NegBioParser(model_dir=PARSING_MODEL_DIR)
loader = NegBioLoader()
```

The extractor is what extracts the observations from the report impressions.

In [21]:
```
extractor = NegBioExtractor(Path(MENTION_PATH), Path(UNMENTION_PATH))
```

The negator will determine whether negation or uncertainty exists in the context of the observations provided by the extractor.

In [22]:
```
neg_detector = ModifiedDetector(PRE_NEG_PATH, NEG_PATH, POST_NEG_PATH)
```

The aggregator then aggregates these observations if they belong to the same category.

In [23]:
```
aggregator = NegBioAggregator(CATEGORIES)
```

Putting it all together

Finally, you'll put everything together using the pipeline function, which takes as arguments all of the objects you've instantiated so far. Then you'll get a nice, clean DataFrame with your result:

In [24]:
```
collection = pipeline(collection, loader, ssplitter, extractor,
                      parser, ptb2dep, neg_detector, aggregator, verbose=True)
```

```
100%|██████████| 50/50 [01:09<00:00,  1.38s/it]
```

In [25]:
```
negbio_pred = pd.DataFrame()
for doc in collection.documents:
    dictionary = {}
    for key, val in doc.infons.items():
        dictionary[key[9:]] = val
    negbio_pred = negbio_pred.append(dictionary, ignore_index=True)
negbio_pred = negbio_pred.replace(
    "Positive", True).replace(
    "Negative", False).replace("Uncertain", False).fillna(False)
```

In [26]:
```
negbio_pred.head()
```

Out[26]:

	Airspace Opacity	Pleural Effusion	Atelectasis	Fracture	Pneumothorax	Consolidation	Cardiomegaly	Edema	Lung Lesion	Pneumonia	Pleural Other
0	True	True	False	False	False	False	False	False	False	False	False
1	True	False	True	True	False	False	False	False	False	False	False
2	False	False	False	False	False	True	False	False	False	False	False
3	True	True	False	False	False	False	True	True	False	False	False
4	False	False	False	False	True	False	False	False	False	False	False

Now you can check every entry in the report impressions for the presence of a condition, while knowing that negation has been taken into account. Really cool!

Congratulations on finishing this notebook!!! This was a very high-level explanation of everything that NegBio does and as you may have noticed, this library leverages many other great tools and libraries. Hopefully, it was a good introduction to how it works. **Nice work, keep it up!**