AI IN HEALTHCARE

Assignment 3: MIMIC NLP

Greg White

Spring 2024

Foreword

- I conducted a named entity recognition (NER) task using natural language processing (NLP) tools on select notes from the MIMIC III dataset.
- I restricted my efforts to a single disease, 'Multiple sclerosis', and further restricted the notes associated with these admissions to discharge notes only. I restricted the population in this manner to (1) reduce the number of records processed due to Google Colab memory and processor constraints and (2) have more consistent record types so peculiarities of other note types and diseases wouldn't muddy results.
- I used pre-trained models for NLP and used Spacy, SciSpacy, and MedSpacy to perform the NER.
- Code is adapted from lecture, example, and repository material relevant to the course and this work is consistent with applicable permissions and licensing requirements.
- After performing the NER and building a corpus for each of the three modules/models I created a word2vec vector representation of each entity occurring more than 50 times and created a dimensional reduction and graphical representation in a tSNE plot with matplotlib.

CONFIG SECTION

Config Globals

- Local flag indicates whether the notebook will run on a local environment or in Colab . This was run on Colab and is set to false.
- Mac flag indicated whether mac-specific directory paths should be substituted for hard-coded windows paths. This was run on Colab and is set to false.
- Min count sets the number of times a word must appear in the corpus to be added to word2vec through its configuration arguments. This is set to 50 to reduce crowding in the tSNE plots and to focus on relationships among more frequently occurring entities.
- The disease variable sets the short name to search for in the icd9 tables. Once the disease is found, the admission records are retrieved for all admissions associated with this diagnosis. Multiple sclerosis was selected as it was a disease with sufficient records to be interesting and challenging to process but not overburdensome to the Colab hardware.

```
# flag for developing on a connected local runtime
# also add mac dir path for dev on macbook pro
local = False
mac = False

# charts can be re-sized for different monitor sizes here
# measurements are in pixel for the size of the whole chart w/ whitespace
CHART_HEIGHT = 500
CHART_WIDTH = 500

# change min word count for word2vec
MIN_COUNT = 50

# set disease to get notes for 'Multiple sclerosis'
# MS was selected as the population was small enough to run on CoLab pro instance
# query builder was used to check admission counts for each disease and progressivley
# rarer diseases were selected until performance was deemed acceptable
disease = 'Multiple sclerosis'
```

Config Packages

- Google Colab uses antiquated packages and infrequently updates them, so all required packages are forced to update so package dependencies can be resolved.
- The –q –q argument suppresses most of the extemporaneous output of the install process, limiting output to warnings and errors.
- While pip may report some dependency issues as Colab has certain requirements specified to pip in its requirements file, they can be ignored as the code runs well as written.
- IF RUNNING ON COLAB, THE RUNTIME MUST BE RESTARTED FOR THE INSTALLATION TO COMPLETE. Use the Runtime menu and select Restart session.

```
# -U arg forces update to most current version on package manager
# -q -q supresses all build and install messages below WARNING level

# update package manager and build versions
%pip install pip -U -q -q
%pip install setuptools -U -q -q
%pip install wheel -U -q -q

# install and update data and nlp packages
%pip install pandas -U -q -q
%pip install numpy -U -q -q
%pip install gensim -U -q -q
%pip install spacy -U -q -q
%pip install scispacy -U -q -q
%pip install medspacy -U -q -q
%pip install spacy-cleaner -U -q -q
%pip install spacy-huggingface-pipelines -U -q -q
%pip install nlp_preprocessor -q -q
```

Config Packages Continued

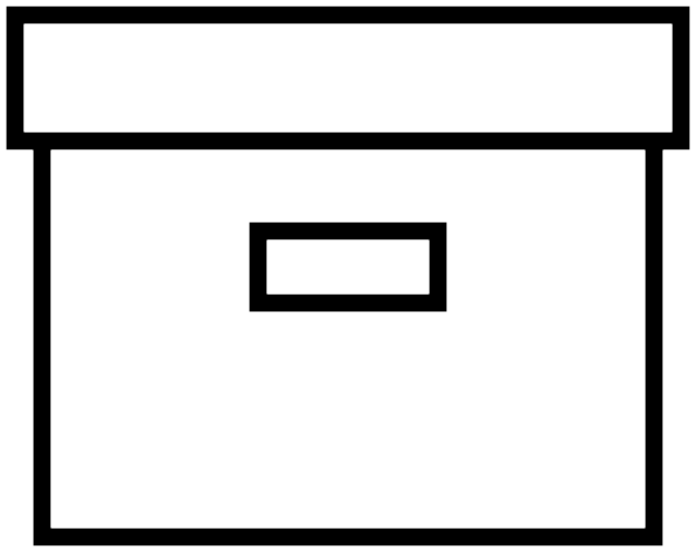
- Required packages are loaded for data processing, including NLP, file, and visualization packages.
- Abbreviations are used for most packages with 'as' clause to make code more readable and easier to write.
- Some packages are selectively loaded from larger packages for performance reasons. The entire package could be loaded without loss of function but not all pieces were needed for this assignment.

```
# import all installed packages required for running assignment
# where possible, use abbrv for readability
import os
import pandas as pd
import numpy as np
import re
import spacy as sp
import scispacy as ssp
import medspacy as ms
import matplotlib.pyplot as plt

# to improve perf, some packages were loaded selectivley
from spacy_cleaner import processing, Cleaner
from thinc.api import PyTorchWrapper as ptw
from gensim.models import Word2Vec as wv
from sklearn.manifold import TSNE as ts
from medspacy.preprocess import PreprocessingRule
from nlp_preprocessor import Preprocessor
```

Config File Paths

- File paths for the MIMIC III data are runtime specific.
- Mac and PC directory structures are hardcoded for efficient lookup without user intervention.
- Files were loaded to the same Google Drive holding the Colab notebook and are loaded with appropriate permissions.
- Google Drive is a requirement for loading and processing the large data files in the MIMIC database. Performance is improved by several orders of magnitude.
- It is critical that the user accepts ALL permissions for the connection to Drive when prompted. Any fewer than ALL results in cryptic error messages.

```
# pathing arguments for loading
if not local:
  # google colab is not installable on local runtime due to out of date software
  # YOU MUST ALLOW ALL PERMISSIONS FOR DRIVE, ANY UNCHECKED BOX RESULTS IN FAILURE
  from google.colab import drive
  drive.mount('/content/drive')
else:
  # path to folder containing unzipped mimic iii data
  # files were unzipped via 7zip and are in folders named after file name in all caps
  if not mac:
    path = 'C:\\Users\\bigre\\OneDrive\\Desktop\\AI HW\\mimic-iii-clinical-database-1.4'
  else:
    path = '/Users/gregwhite/Desktop/mimic-iii-clinical-database-1.4'
```



FILE LOADING SECTION

File Loading Code

- File path directories are pulled from variables set in the CONFIG section.
- The relevant files for this task include the NOTEVENTS file (notes associated with all admissions), the DIAGNOSES_ICD file (all diagnoses in the EHR from the ICD9 specification), and the D_ICD_DIAGNOSES file (specific information for all ICD9 codes).
- Files are loaded into panda dataframes for further manipulation.
- Files are read in their entirety.
- In this notebook, files were loaded from the 'else' group pointing to specific co-located Google Drive files.

```
# use panda csv reader for both local and colab runtimes
# local runtimes accesses files with hardcoded paths
# colab runtimes accesses files with google colab lib and file selection via gui
# load notes and d_icd_diagnoses data
if local:
  if mac:
    notes = pd.read_csv(path+ '/NOTEVENTS.csv.gz')
    diag = pd.read_csv(path+ '/DIAGNOSES_ICD.csv.gz')
    d_diag = pd.read_csv(path+ '/D_ICD_DIAGNOSES.csv.gz')
  else:
    notes = pd.read_csv(path+ '\\NOTEVENTS.csv\\NOTEVENTS.csv')
    diag = pd.read_csv(path+ '\\DIAGNOSES_ICD.csv\\DIAGNOSES_ICD.csv')
    d_diag = pd.read_csv(path+ '\\D_ICD_DIAGNOSES.csv\\D_ICD_DIAGNOSES.csv')
else:
  # files must be in MIMIC folder and store in .gz format with the following file names
  notes = pd.read_csv('/content/drive/MyDrive/MIMIC/NOTEEVENTS.csv.gz')
  diag = pd.read_csv('/content/drive/MyDrive/MIMIC/DIAGNOSES_ICD.csv.gz')
  d_diag = pd.read_csv('/content/drive/MyDrive/MIMIC/D_ICD_DIAGNOSES.csv.gz')
```

DATA CLEANUP SECTION

Find Relevant Notes

- The D_ICD_DIAGNOSES dataframe is queried for the ICD9 code related to the disease specified in the CONFIG section.
- All diagnostic entries are found for that ICD9 code in the DIAGNOSES_ICD dataframe.
- The diagnoses for the given disease are INNER JOINED with the hospital admission records in the ADMISSIONS dataframe.
- The notes in the resulting dataframe are filtered to only provide discharge summaries.
- The dataframe is further cleaned by removal of all N/A or blank notes.

```
# get all discharge notes for admissions associated with disease set in CONFIG
# get icd9 row_id for target disease
icd9_code = d_diag.loc[d_diag['SHORT_TITLE']==disease]['ICD9_CODE'].values[0]

# get all icd9 diags for given diseases
filtered_diag = diag.loc[diag['ICD9_CODE'] == icd9_code]

# get all discharge summaries for given diags
all_notes = pd.merge(filtered_diag, notes, on = "HADM_ID", how = "inner")

# reduce set to discharge summaries only
filtered_notes = all_notes.loc[all_notes['CATEGORY'] == 'Discharge summary']

# remove notes with null text
filtered_notes.dropna(subset = ['TEXT'], inplace = True)
```

Define NER Extraction

- Two methods are written (due to some issues with Colab overloading) to perform the NER process, one using a pre-processing cleaning step and one without.
- The pre-processing cleaning was performed by spacy-cleaner and used arguments most appropriate to the notes text to remove non-sensical information.
- For each method, the NLP extracts the entities (along with related data) and then yields only the list of entities for each document.

```
# overloaded method to extract just the tokens from the nlp

# this method takes a cleaner from spacy-cleaner to remove unwanted chars or text
# get all tokens and return str list of text
def get_text_items_cleaner(cleaner, nlp, txt):
  cleaner.clean(txt)
  doc = nlp(txt)
  wlst = []
  for token in doc:
    wlst.append(token.text)
  return wlst

# this method does not use a spacy-cleaner and the cleaning is performed by med-spacy
# get all tokens and return str list of text
def get_text_items(nlp, txt):
  doc = nlp(txt)
  wlst = []
  for token in doc:
    wlst.append(token.text)
  return wlst
```

Define MedSpacy Cleaning

- The cleaning pre-processing functionality of MedSpacy was leveraged to clean up MIMIC notes text.
- These rules are used later when the MedSpacy model is loaded.
- These specific pre-processing rules are from examples provided by the MedSpacy team in documentation and in the code files of the respository.

```
# preprocessing rules, used by med-spacy to clean the mimic notes
# https://github.com/medspacy/medspacy/blob/master/notebooks/08-Preprocessing-Postprocessing.ipynb
preprocess_rules = [

    PreprocessingRule(
        r"\[\*\*[\d]{1,4}-[\d]{1,2}(-[\d]{1,2})?\*\*\]",
        repl="01-01-2010",
        desc="Replace MIMIC date brackets with a generic date."
    ),

    PreprocessingRule(
        r"\[\*\*[\d]{4}\*\*\]",
        repl="2010",
        desc="Replace MIMIC year brackets with a generic year."
    ),

    PreprocessingRule(
        r"dx'd",
        repl="Diagnosed",
        desc="Replace abbreviation"
    ),

    PreprocessingRule(
        r"tx'd",
        repl="Treated",
        desc="Replace abbreviation"
    ),

        PreprocessingRule(
        r"\[\*\*[^\]]+\]",
        desc="Remove all other bracketed placeholder text from MIMIC"
    )
]
```

SPACY

Spacy NER

- Spacy is used to perform the NER.
- A deep copy of the filtered notes dataframe is performed so any actions in this processing run do not affect later processing.
- On subsequent runs, the entities can be retrieved from a csv file that is saved as an intermediate step. This is a large performance enhancing step as the cleaning process for each doc is long.
- The pre-trained model en_core_web_lg is used to provide distinct NER from the other models used in future sections.
- The cleaner is configured with recommended values from spacy_cleaner's documentation.
- The entities are extracted in a lambda expression across the whole notes dataframe and is inserted into a new column in the dataframe.
- The first note's entities are displayed for visual inspection.

```
# if running again, skip entity extraction and load from processed csv
if os.path.isfile('/content/drive/MyDrive/NLP/spacy_core_web_lg_' + disease + '.csv'):
  sp_ent = pd.read_csv('/content/drive/MyDrive/NLP/spacy_core_web_lg_' + disease + '.csv')
else:
  # download desired spacy model (large)
  sp.cli.download('en_core_web_lg')

  # work against deep copy of df
  sp_ent = filtered_notes.copy(deep = True)

  # load model
  nlp = sp.load('en_core_web_lg')

  # https://spacy.io/universe/project/spacy-cleaner
  # clean up text to replace nums and punctuation and stop words
  cleaner = Cleaner(
      nlp,
      processing.remove_stopword_token,
      processing.replace_punctuation_token,
      processing.mutate_lemma_token,
  )

  # add new column with ents for each note to df
  sp_ent['ents'] = sp_ent['TEXT'].apply(lambda x: get_text_items_cleaner(cleaner,nlp,x))

  # save entities to csv so you dont have to run again
  sp_ent.to_csv('/content/drive/MyDrive/NLP/spacy_core_web_lg_' + disease + '.csv', encoding='utf-8', index=False)

# see entities for first note
sp_ent.at[0, 'ents']
```

SCISPACY

Scispacy NER

- SciSpacy is used to perform the NER.
- A deep copy of the filtered notes dataframe is performed so any actions in this processing run do not affect later processing.
- On subsequent runs, the entities can be retrieved from a csv file that is saved as an intermediate step. This is a large performance enhancing step as the cleaning process for each doc is long.
- The pre-trained model en_ner_bc5cdr_md is used to provide distinct NER from the other models used in future sections. This model is pre-trained on biomedical text.
- The model is only installed when this task is run as the dependencies conflict with other installs performed in CONFIG.
- The cleaner is configured with recommended values from spacy_cleaner's documentation.
- The entities are extracted in a lambda expression across the whole notes dataframe and is inserted into a new column in the dataframe.
- The first note's entities are displayed for visual inspection.

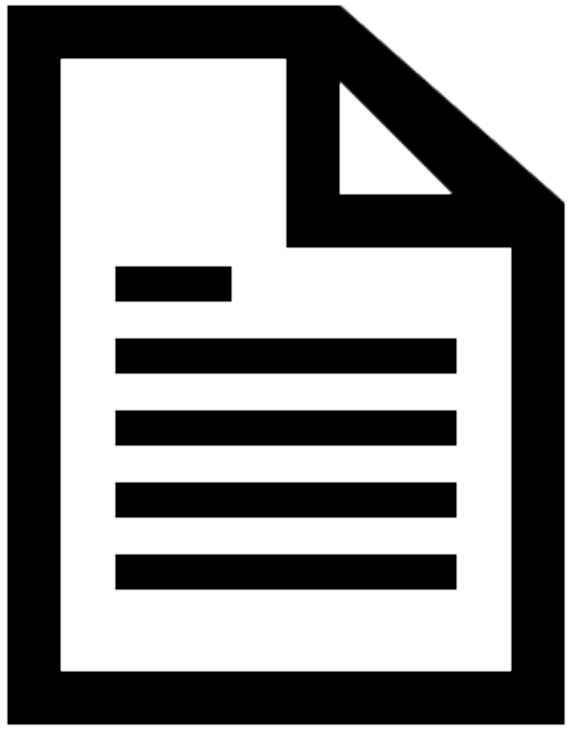
```
# if running again, skip entity extraction and load from processed csv
if os.path.isfile('/content/drive/MyDrive/NLP/scispacy_bc5cdr_md_' + disease + '.csv'):
  ssp_ent = pd.read_csv('/content/drive/MyDrive/NLP/scispacy_bc5cdr_md_' + disease + '.csv')
else:
  #download desired scispacy model (bc5cdr_md )
  %pip install https://s3-us-west-2.amazonaws.com/ai2-s2-scispacy/releases/v0.4.0/en_ner_bc5cdr_md-0.4.0.tar.gz

  # work against deep copy of df
  ssp_ent = filtered_notes.copy(deep = True)

  # load model
  nlp = sp.load('en_ner_bc5cdr_md')

  # https://spacy.io/universe/project/spacy-cleaner
  # clean up text to replace nums and punctuation and stop words
  cleaner = Cleaner(
      nlp,
      processing.remove_stopword_token,
      processing.replace_punctuation_token,
      processing.mutate_lemma_token,
  )

  # add new column with ents for each note
  ssp_ent['ents'] = ssp_ent['TEXT'].apply(lambda x: get_text_items_cleaner(cleaner, nlp, x))

  # save entities to csv so you dont have to run again
  ssp_ent.to_csv('/content/drive/MyDrive/NLP/scispacy_bc5cdr_md_' + disease + '.csv', encoding='utf-8', index=False)

# see entities for first note
ssp_ent.at[0, 'ents']
```

MEDSPACY

MedSpacy NER

- MedSpacy is used to perform the NER.
- A deep copy of the filtered notes dataframe is performed so any actions in this processing run do not affect later processing.
- On subsequent runs, the entities can be retrieved from a csv file that is saved as an intermediate step. This is a large performance enhancing step as the cleaning process for each doc is long.
- The pre-trained model albert-medical-ner-proj is used to provide distinct NER from the other models used in future sections. This model was found on huggingface and is a pre-trained BERT model using medical literature.
- The model is only installed when this task is run as the dependencies conflict with other installs performed in CONFIG.
- The preprocessor is configured with the settings specified in the DATA CLEANUP section.
- The entities are extracted in a lambda expression across the whole notes dataframe and is inserted into a new column in the dataframe.
- The first note's entities are displayed for visual inspection.

```
# if running again, skip entity extraction and load from processed csv
if os.path.isfile('/content/drive/MyDrive/NLP/albert-medical-ner-proj_' + disease + '.csv'):
  ms_ent = pd.read_csv('/content/drive/MyDrive/NLP/albert-medical-ner-proj_' + disease + '.csv')

else:
  # work against deep copy of df
  ms_ent = filtered_notes.copy(deep = True)

  # load blank model
  nlp = ms.load()
  # add pre-trained model as token pipe to nlp
  nlp.add_pipe("hf_token_pipe", config={"model": "medical-ner-proj/albert-medical-ner-proj"})

  # preprocess using rules in DATA CLEANUP to remove unwanted input from notes
  preprocessor = Preprocessor(nlp.tokenizer)
  nlp.tokenizer = preprocessor
  preprocessor.add(preprocess_rules)

  # add new column with ents for each note
  ms_ent['ents'] = ms_ent['TEXT'].apply(lambda x: get_text_items(nlp,x))

  # save entities to csv so you dont have to run again
  ms_ent.to_csv('/content/drive/MyDrive/NLP/albert-medical-ner-proj_' + disease + '.csv', encoding='utf-8', index=False)

# see entities for first note
ms_ent.at[0, 'ents']
```



WORD2VEC

Word2Vec Encoding

- Word2vec is adapted from the example to create vectors for each word in the NER corpus that occurs at least MIN_COUNT times.
- A word, "performed" was selected to confirm the word2vec model had processed the data successfully.
- The vectors for the given word are displayed.
- The model is checked for the most similar words by vector. Words like done and obtained would make a lot of sense in consideration of performed as they are results of a task like performed.

```
# generate a word2vec encoding for each corpus
spacy_model = wv(sp_ent['ents'].to_list(), min_count = MIN_COUNT)
scispacy_model = wv(ssp_ent['ents'].to_list(), min_count = MIN_COUNT)
medspacy_model = wv(ms_ent['ents'].to_list(), min_count = MIN_COUNT)
```

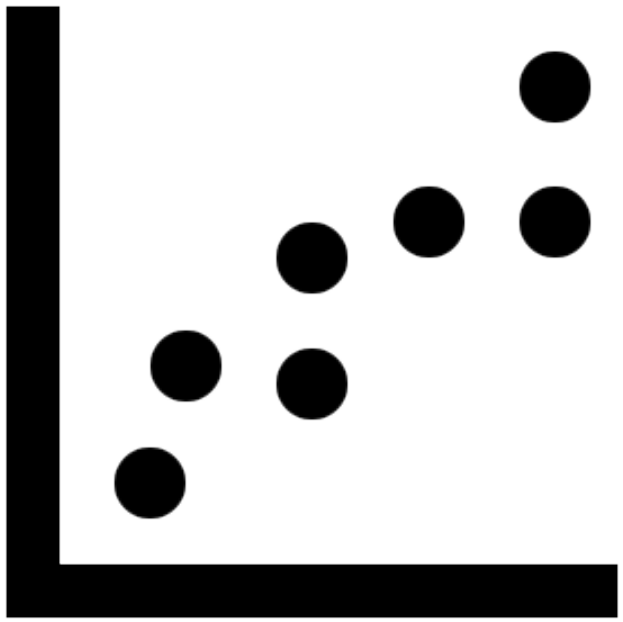
```
# config vars for w2v demo
word = "performed"
```

```
# see w2v vector for given word
spacy_model.wv[word]
```

```
array([ 0.40218985,  0.43970832,  0.80300564, -0.08110847, -0.7237941 ,
       -0.3519538 ,  0.30912942, -0.46563405, -0.20791994,  0.4076424 ,
        0.80176485,  0.20378162,  0.12214528, -0.55300295, -0.8195721 ,
       -1.077182  ,  0.01073126, -0.2331222 , -0.0469726 ,  0.05800621,
        0.36568022, -0.16920422,  0.12212812,  0.31828094, -0.29685363,
        0.19955283, -0.03828701, -0.6820717 ,  0.20017697, -0.1554183 ,
       -0.07943325,  0.2771323 ,  0.7207642 , -0.4823637 ,  0.10198066,
       -0.3032094 , -0.60200787, -0.15675233, -0.4061264 ,  0.00740666,
       -0.32988465, -0.9274994 ,  0.9262295 ,  0.1432677 , -0.17781062,
       -0.5584554 ,  0.3015411 , -0.8718876 ,  0.27109253,  0.7319253 ,
        0.47319013, -0.9891093 , -0.35075995, -0.11400718, -0.8145453 ,
       -0.8543735 ,  0.09114161,  0.6844283 , -1.1162522 ,  0.83541065,
        0.02917493,  0.09701664,  0.5049955 , -0.39570522, -0.21558566,
        0.5839581 ,  0.73145676,  0.1917951 , -0.545753  ,  0.6294057 ,
       -0.72281075,  0.63636744,  0.5449003 , -0.29246563,  1.2187313 ,
        0.73532563, -0.37376544,  0.02987189, -0.05644527, -0.03451042,
       -0.4481365 ,  0.5431746 , -0.85676503, -0.16589585,  0.38609672,
       -0.5255789 ,  0.42274266,  0.12989256,  0.04730193,  0.14890708,
       -0.12782681, -0.26063013,  0.40642548, -0.45877275, -0.04379699,
        0.1896607 ,  0.43536976, -0.00655844, -0.41728127, -0.48489946],
      dtype=float32)
```

```
# find most similar words in w2v for given word
spacy_model.wv.similar_by_word(word)
```

```
[('done', 0.8878586292266846),
 ('obtained', 0.8295945525169373),
 ('LP', 0.8178612589836121),
 ('An', 0.7729572653770447),
 ('EEG', 0.7346168756484985),
 ('repeat', 0.733252227306366),
 ('head', 0.73268723487854),
 ('scan', 0.7292380928993225),
 ('MRI', 0.7121354341506958),
 ('ultrasound', 0.7004503011703491)]
```



TSNE PLOTS

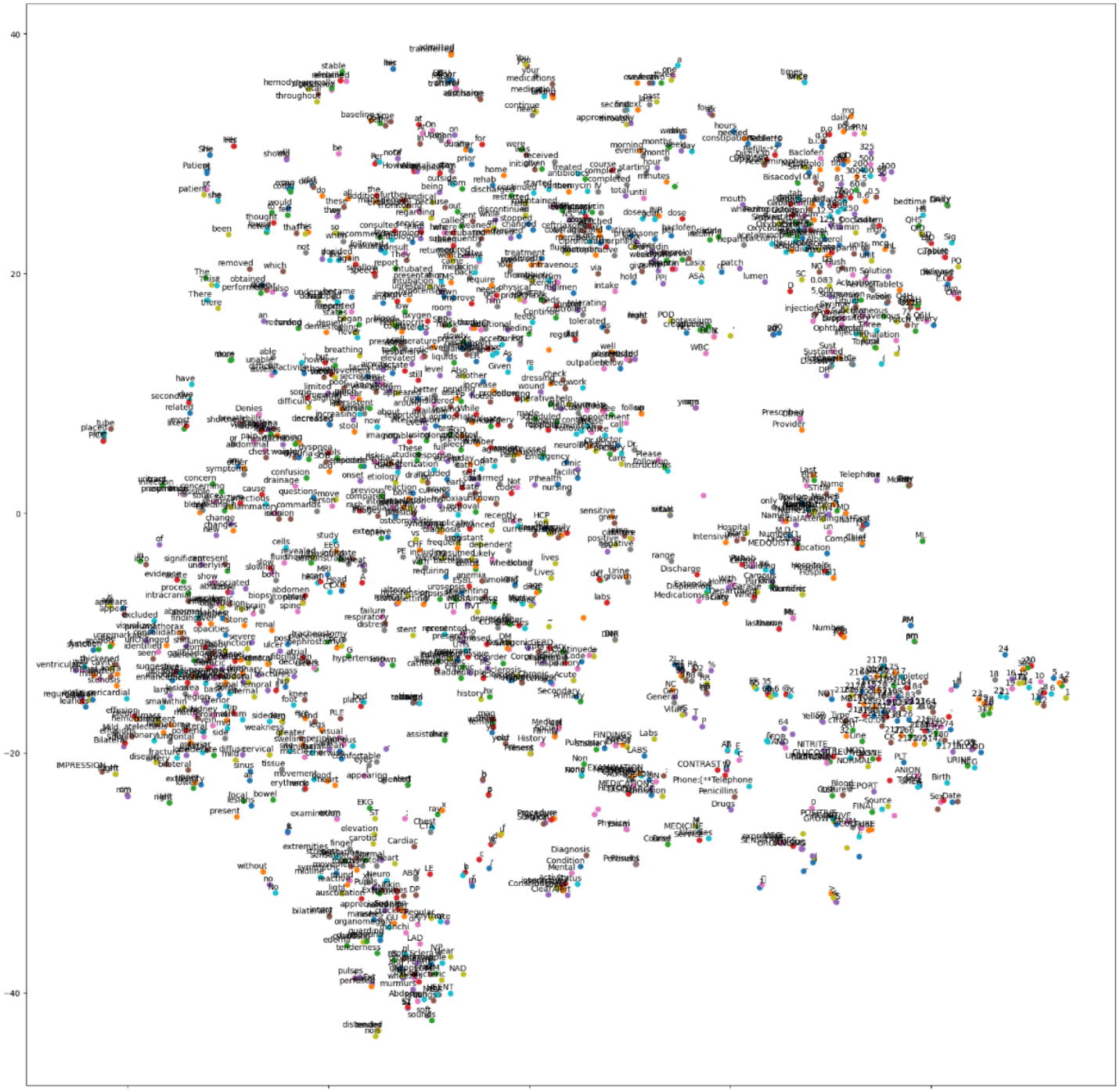
TSNE Plots

- Code to create dimensionally reduced 2d TSNE plots was re-used (with permission) from the example.
- The words in the word2vec model are reduced to 2 dimensions and plotted in line with matplotlib.
- The first function call is displayed in the picture but is identical for the other two NLP tools used, differing in model name argument only.
- The following plots are included to show the successful completion of this exercise. They are not easily readable in this slide format and are best viewed and interpreted in the notebook itself.
- Clustered words were taken to be of some closer relation in the discharge note summaries.

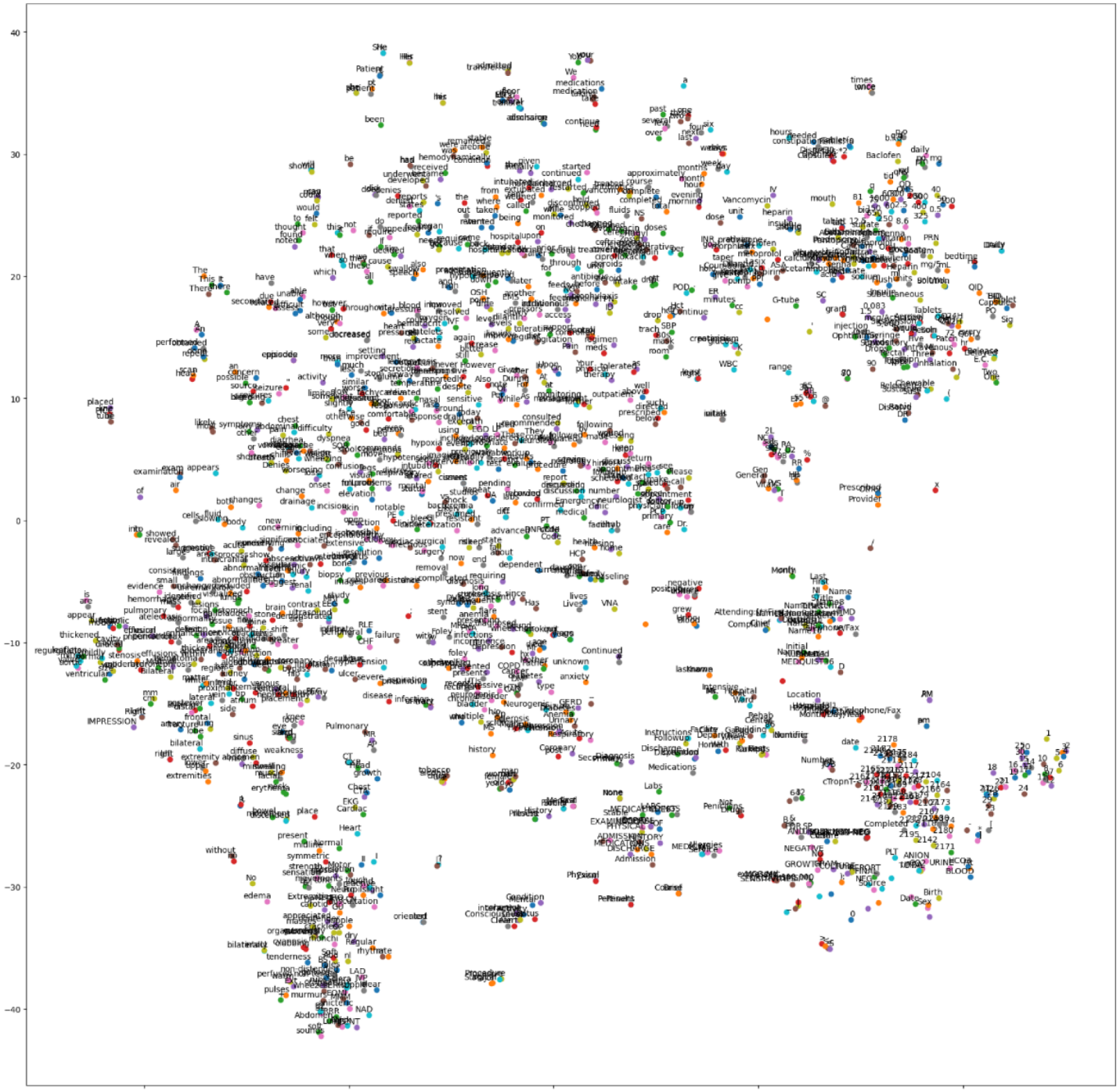
```
# set inline mode for matplotlib
%matplotlib inline

# define method for making tsne plot
def tsne_plot(model,words, preTrained=False):
    "Creates and TSNE model and plots it"
    labels = []
    tokens = []

    # get all tokens for corpus and add append to list
    for word in words:
      if preTrained:
          tokens.append(model[word])
      else:
          tokens.append(model.wv[word])
      labels.append(word)

    # make array of all tokens
    tokens = np.array(tokens)

    # config tsne plot, with args from lecture
    tsne_model = ts(perplexity=30, early_exaggeration=12, n_components=2, init='pca', n_iter=1000, random_state=23)
    new_values = tsne_model.fit_transform(tokens)

    # plot each word as a point in 2d space
    x = []
    y = []
    for value in new_values:
        x.append(value[0])
        y.append(value[1])

    plt.figure(figsize=(24, 24))
    for i in range(len(x)):
        plt.scatter(x[i],y[i])
        plt.annotate(labels[i],
                     xy=(x[i], y[i]),
                     xytext=(5, 2),
                     textcoords='offset points',
                     ha='right',
                     va='bottom')

    # reveal the plot
    plt.show()
```

```
# make and display a tsne plot for the word2vec encodings for the spacy NER model
# words plotted in clusters are assumed to be related per model
tsne_plot(spacy_model, np.array(list(spacy_model.wv.index_to_key)))
```

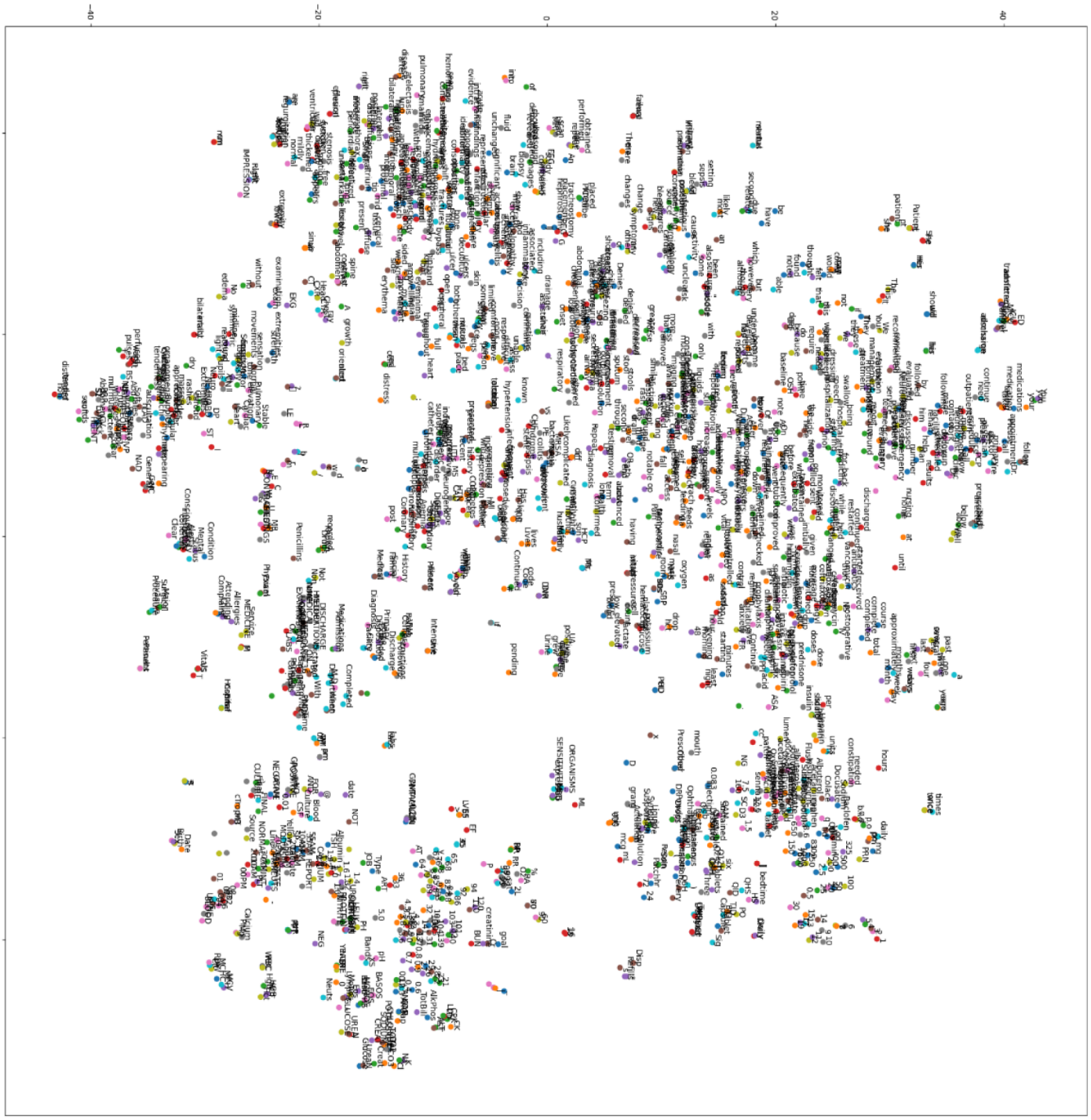
TSNE Plot for Spacy



TSNE Plot for SciSpacy



TSNE Plot for MedSpacy



CONCLUSIONS

Assignment 3 Conclusions

- Pre-trained models specifically trained for the field in which they will be used (medical, specifically charting terminology) extract more reasonable and useful entities in NER.
- MIMIC data, specifically the free-form text of notes, is very noisy with lots of shorthand, misspellings, and lots of cruft like dates and value measurements that have low value to or interfere with NER.
- NER makes quick work of processing large volumes of text for salient details and NLP can demonstrate relatedness between identified terms. Processing this data manually would be an insurmountable task for this dataset.
- NER still feels immature and while it will be a powerful tool for researchers and developers, the tools used in performing this analysis have too many shortcomings (or the author's usage is just to amature) to be incorporated as-is into any production workflows like medical care quality checks.

ACKNOWLEDGMENTS

Acknowledgments and Sources Used

- The code is original code processing and displaying information from the files provided by MIT in the MIMIC III dataset. While it is my own it may include code excerpts from or based on the spacy, scispacy, medspacy, and spacy-cleaner repositories. Pretrained models were sourced from other entities as exemplified in example code. Code and examples were also used from lecture and example materials provided by Prof. Ding with permission. The slides were made in PowerPoint and the colab logo was sourced from Google. I am not confident in citing the pretrained models as source appears ambiguous but none of the pretrained models are my own.
- https://edstem.org/us/courses/52089/discussion/4436192
- https://github.com/medspacy/medspacy
- https://ce11an.github.io/spacy-cleaner/
- https://allenai.github.io/scispacy/
- https://spacy.io/
- https://utexas.instructure.com/courses/1379131/files/75857310?module_item_id=13543104
- https://physionet.org/content/mimiciii/1.4/