NLP project proposal

Section	Points
Code	**25**
Raw	10
Unit tests	5
Package (backend)	5
CLI	5
Presentation	**10**
GitLab	**5**
Total	**40**

Aims

The primary goal of this project is to develop a search engine using NER technology that will find relevant papers from a corpus.

1. **Create a corpus of scientific abstracts** using the NCBI E-utilities API

   Collect the abstract from **NCBI E-utilities API** I saw the example of using Entrez package from Biopython or using **requests API**. Also make sure to have 1 sec delay in response to prevent yourself from getting block by the pubmed website. We can use 3 years data but for testing we can start with 3 months data (then increase it to 3/4 year data) and also we have to specify whether these abstract are biological process or related search so that we can limit our query using MESH Terms.

   After collection of data we can see how many abstracts are there and how it is stored in xml /json format in our CACHED FOLDER, later we can create database in the same folder then we can store this info in our database as table(PUBMED_ABSTRACT) with columns id, **PUBMEDID** and text.

For cli what we can do here is to create a function that takes pubmed id and return the text/abstract of the related pubmedid or it can return the link of the url and when we click with it takes us to the webpage where the article in stored in pubmed.

2. **Apply NER technology** to tag words within the abstracts with specific keywords

   For NER model we can use pre trained model **en_core_web_sm or BERT** only problem with Bert is that it require additional GPU which can be the problem for other machine. After using the model and fetching our entries and the specific Labels. What we can do is we can create another table in our database called "named_entity" which contain text abstract id , text, label as columns.

   CLI idea search for label get the nested dictionary of the pubmed id(from abstract id which is foreign key in named entity) and the abstract it is present in.

3. **Store the abstracts and their tagged metadata** for faster searching

   Define a new function which search for keyword or phrases in the database and give informative results for example which pubmed id it uses , abstract, webpage. We can modify our function to return the results from a specified date/range.

4. **Create a frontend** which allows one to search the corpus using a keyword or phrase
    1. This should include a box where users can input their keyword/phrase
    2. There should be a feature included to filter the results to those abstracts within a specific range of dates
    3. The returned results should include
        1. The PMID of the abstract (with a hyperlink to its PubMed page)
        2. The date of publication
        3. A collapsable field with the entire abstract for users to read
        4. A list of keywords/phrases in the abstract which were identified as being tagged with the user's search query
    4. Include a button so that user's can download this information (you can choose the available format(s))

   The above points should be returned in form table? We can use Flask which we used for hw but using the same template and changing the data source but we have to make sure it does not take any rubbish thing apart from Labels/keywords and the above list(point 3) must be stored in our database before doing the query .

   CLI use above list/dict and print it in the stdout.

We can create some basic files:
APi file for downloading info
Database .py
Model.py
Constant.py- not sure??
cli.py
For **unit tests** we can create a dummy data on which we can test are functions,
For **package** we can store everything just like we did in the hw.
Containerization must be same as what we did in hw: problem may occur by connecting frontend and backend

THE DETAILED ANSWERS ARE GIVEN BELOW.

Tasks

Task 1 - *Create a Corpus* (2 pts)

- Using tools available from NCBI, write code that downloads abstracts from PubMed from the last 3 years
    - Be sure to follow the guidelines of the API
    - Because we are only interested in biological publications, it may be helpful to filter the cached abstracts using their included metadata. There may be certain keywords or tags that can be used to select for relevant entries
    - If you find that the number of abstracts is too small to be useful, feel free to expand the number of abstracts collected from PubMed (e.g. from the last 4 years instead)

https://erilu.github.io/pubmed-abstract-compiler/

one example of using request lib to collect past 4 or 3 years abstracts

```
import requests
from bs4 import BeautifulSoup
from datetime import datetime, timedelta

# Define your API key and search terms
api_key = "your_api_key_here"
search_terms = "biological process[MeSH Terms]"

# Get the current date and subtract 4 years
current_date = datetime.now()
four_years_ago = current_date - timedelta(days=365*4)

# Define the date range to search
date_range = f"{four_years_ago.year}:{current_date.year}"

# Make the API call
url = f"https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?
db=pubmed&term={search_terms}&datetype=pdat&reldate=
{date_range}&retmax=1000&api_key={api_key}"
response = requests.get(url)

# Parse the XML response using BeautifulSoup
soup = BeautifulSoup(response.text, "lxml")

# Extract the abstracts
```

Or BioPython lib to extract data

```
from Bio import Entrez

Entrez.email = "your_email@example.com"  # Always tell NCBI who you are
```

```
# Search for articles

handle = Entrez.esearch(db="pubmed", term="cancer treatment", retmax=10)

record = Entrez.read(handle)

ids = record["IdList"]

# Retrieve the abstracts

handle = Entrez.efetch(db="pubmed", id=ids, rettype="abstract", retmode="text")

records = handle.read()

print(records)
```

In this step only we can create a catch folder of our database

Then we can store the results in our database

PubMed abstract: name of the database and we have id, text as column where text is our abstract.

```
from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.ext.declarative import declarative_base

# Create the database engine
engine = create_engine('sqlite:///abstracts.db')

# Define the abstracts table
Base = declarative_base()
class Abstract(Base):
    __tablename__ = 'abstracts'
    id = Column(Integer, primary_key=True)
    text = Column(String)

# Create the table
Base.metadata.create_all(engine)

# Create a session
from sqlalchemy.orm import sessionmaker
Session = sessionmaker(bind=engine)
session = Session()

# Add the abstracts to the table
for abstract in abstracts:
    session.add(Abstract(text=abstract))

# Commit the changes
```

Task 2 - *Tag the Abstracts using NER* (2 pts)

- Iterate through the abstracts and tag specific words or phrases using an NER model
    - While you are free to create your own model, it is recommended to find a pre-trained model or a library with a model included to use: what I saw usually for word tag pre trained model en_core_web_sm is used but we can also use other pre trained model like BERT

```
import spacy

import mysql.connector


# Load the pre-trained NER model

nlp = spacy.load("en_core_web_sm")


# Connect to the database

mydb = mysql.connector.connect(

  host="hostname",

  user="username",

  password="password",

  database="database_name"

)


# Create a cursor to interact with the database

cursor = mydb.cursor()


# Fetch the abstracts from the database

cursor.execute("SELECT abstract FROM abstracts")

abstracts = cursor.fetchall()
```

```
# Iterate through the abstracts

for abstract in abstracts:

  abstract = abstract[0]

  # Process the abstract with the NER model

  doc = nlp(abstract)

  for ent in doc.ents:

    print(ent.text, ent.label_)



# Close the cursor and the connection

cursor.close()

mydb.close()
```

- Create a way to store this information so that one can quickly access it
    - This can be done using cached files or a DB

We can create a new table in our database called named_entities and store the text and its Tag labels in our newly created table and commit those changes in the database

NOTE: everything which we are downloading is stored in our cached folder

```
import spacy

import mysql.connector

# Load the pre-trained NER model

nlp = spacy.load("en_core_web_sm")

# Connect to the database

mydb = mysql.connector.connect(

 host="hostname",

 user="username",

 password="password",
```

```
  database="database_name"
)
# Create a cursor to interact with the database
cursor = mydb.cursor()
# Fetch the abstracts from the database
cursor.execute("SELECT abstract FROM abstracts")
abstracts = cursor.fetchall()
# Iterate through the abstracts
for abstract in abstracts:
    abstract = abstract[0]
    # Process the abstract with the NER model
    doc = nlp(abstract)
    for ent in doc.ents:
        # Insert the named entity and its label into the database
        sql = "INSERT INTO named_entities (abstract_id, text, label) VALUES (%s, %s, %s)"
        val = (abstract_id, ent.text, ent.label_)
        cursor.execute(sql, val)
# Commit the changes to the database
mydb.commit()
# Close the cursor and the connection
cursor.close()
mydb.close()
```

using bert model in below example:

```
import torch
from transformers import BertForTokenClassification, BertTokenizer
```

```
# Load the pre-trained BERT model and tokenizer

model = BertForTokenClassification.from_pretrained("bert-base-cased")

tokenizer = BertTokenizer.from_pretrained("bert-base-cased")


# Iterate through the abstracts

for abstract in abstracts:

  # Tokenize the abstract

  input_ids = torch.tensor([tokenizer.encode(abstract, add_special_tokens=True)])

  # Get the predicted named entity labels

  with torch.no_grad():

    output = model(input_ids)[0]

  labels = output.argmax(-1)

  # Extract the named entities and their labels

  entities = []

  for i, label in enumerate(labels[0]):

    if label != 0:

      entities.append((tokenizer.convert_ids_to_tokens[input_ids[0][i]], model.config.id2label[label]))

  print(entities)
```

Task 3 - *Develop the Search Functionality* (2 pts)

- Generate functions that allows one to query your tagged corpus using a keyword or phrases and get informative results (see Task 3)
    - As an example, one should be able to input specific gene symbols (e.g. IL2, IFNA) or phrases (e.g. "cell cycle" or "apoptosis")\

```
import spacy

# Load the pre-trained NER model
nlp = spacy.load("en_core_web_sm")
```

```
def query_corpus(keyword):
    result = []
    for abstract in abstracts:
        # Process the abstract with the NER model
        doc = nlp(abstract)
        for ent in doc.ents:
            if keyword in ent.text:
                result.append((ent.text, ent.label_))
    return result
```

- Write/modify methods for restricting the searches to a range of dates i.e. only search through abstracts within a certain range of time

```
import spacy
from datetime import datetime

# Load the pre-trained NER model
nlp = spacy.load("en_core_web_sm")

def query_corpus(keyword, start_date, end_date):
    result = []
    for abstract in abstracts:
        # Extract the date of the abstract
        date_str = extract_date(abstract)
        date = datetime.strptime(date_str, "%Y-%m-%d").date()
        # Check if the date is within the specified range
        if date >= start_date and date <= end_date:
            # Process the abstract with the NER model
            doc = nlp(abstract)
            for ent in doc.ents:
                if keyword in ent.text:
                    result.append((ent.text, ent.label_))
    return result
```

Task 4 - *GUI* (3 pts)

- Construct a web interface that allows one to search your corpus of abstracts using a specific keyword or phrase
    - This should include a box where users can input their keyword/phrase
    - There should be a feature included to filter the results to those abstracts within a specific range of dates
    - The returned results should include
        - The PMID of the abstract (with a hyperlink to its PubMed page)

        - The date of publication
        - A collapsable field with the entire abstract for users to read
        - A list of keywords/phrases in the abstract which were identified as being tagged with the user's search query
    - Include a button so that user's can download this information (you can choose the available format(s))

Task 5 - *Containerize the Application* (1 pt)

- Create a `Dockerfile` in your project's root directory that successfully compiles your application into a Docker image
    - It should bundle your backend and frontend code together
    - You may choose any base image
    - All members of group should be in the image metadata using `LABEL`
    - The `ENTRYPOINT` should start your web application and the web app should be accessible using port mappings

  This needs to be done in the end when our database and frontend is ready.

My Preferences:

Task1 or 2 that is fetching data and creating model.

What I think is that we have to focus on main task apart from CLI, UNIT Tests, containerization and even package.

Because these will be assessed later on.

DIVISION: each person should take one task but if someone is taking task3 then they can also help the person doing task2

TASK1

TASK2

TASK3

TASK4