COMP3225 + COMP6253 Natural Language Processing

Stuart E. Middleton, sem03@soton.ac.uk

Updated: 19th Jan 2023

Deliverables and deadlines

Deliverable	Deadline	Feedback	Weight
task1_submission.py	20 attempts (best taken) deadline module week 7	Automated email of F1 scores for each submission	5%
task2_submission.py	20 attempts (best taken) deadline module week 8	Automated email of F1 scores for each submission	5%
task3_submission.py	20 attempts (best taken) deadline module week 10	Automated email of F1 scores for each submission	10%
task4_submission.py	20 attempts (best taken) deadline module week 12	Automated email of F1 scores for each submission	5%
			Coursework total 25% of module mark

Tasks

This assignment is about implementing practical natural language processing (NLP) algorithms to perform information extraction and named entity recognition tasks. From a UTF-8 plain text serialized book, you will generate table of content, set of questions, set of character names and set of named entities (with types). Your code will be automatically run on a book hidden from you and evaluated against a manually curated ground truth also hidden from you.

Resources

You are provided (via links on module wiki) with the following:

1. nlp_submission.py >> template for python code file you will need to submit
2. ontonotes_parsed.json >> dataset that will be available to train NER model
3. eval_chapter.txt >> example test corpus (chapter) from David Copperfield
4. eval_book.txt >> example test corpus (book) from David Copperfield
5. gold_characters.txt >> example ground truth for characters
6. gold_questions.txt >> example ground truth for questions
7. gold_ne.json >> example ground truth for named entities
8. gold_toc.json >> example ground truth for table of contents

You should download many (at least 10 book styles) plain text books from the free to use Project Gutenberg webpage https://www.gutenberg.org to create a training set. The hidden evaluation book will be a classic novel, so ensure you have examples of these in your training set. Testing on

one book (David Copperfield provided in example package) will not be sufficient, as books vary in style and content and your code will very likely overfit to the single book.

For example, books by Charles Dickens can be downloaded from https://www.gutenberg.org/ebooks/author/37 and the book David Copperfield from https://www.gutenberg.org/ebooks/766

A hidden book will be selected by the lecturers from the Project Gutenberg collection (plain text UTF-8) and used for automated evaluation. A ground truth table of contents, set of questions, set of characters and set of named entities will be manually curated for this hidden text. Your code will be run using this hidden text and F1 scores computed against the ground truth.

Note that Project Gutenberg plain text books are serialized as lines of text with a limited width. This means you will need to rebuild the paragraphs of text to make a list of complete sentences without newlines (\r\n) embedded within them. Look at the eval_book.txt for an example (David Copperfield) and gold_questions.txt for examples of how ground truth sentences will not include any newlines.

The module labs (regex, CRF NER) provide an excellent resource to help you with writing the code for this coursework.

Tasks your submitted code must perform

Task 1 – Use regex to extract chapter headings from the book text and serialize a table of contents as a JSON file saved to current working dir.

If the book has its own table of contents and it disagrees with the book's chapter headings, then use the information from the book's chapter headings (e.g. TOC uses roman numerals but chapter headings use numbers). This might occur if the TOC at the start is in uppercase or shortens the real chapter headings.

You need to handle the various styles of books in Project Gutenberg. For example, a book might have a chapter title after a chapter index (e.g. 'Chapter 1: Some title') or it might have a chapter title on a new line below the chapter index using all caps to indicate it's a title and not a paragraph (e.g. 'Chapter 2\r\n\r\nSOME TITLE IN CAPS'). Download a variety of books to ensure you have a large enough training set.

If the book has parts, volumes or is made up of multiple books, then prefix each of these to the chapter number within parentheses e.g. { '(Book 1) IV' : 'some title for Book 1 Chapter IV' } or { '(Part A) 3' : 'some title for Part A Chapter 3' } or { '(Volume 1) (Part A) 4' : 'some title for vol 1 Part A Chapter 4' }.

> Input >> book
>
> Format of text extraction >> keep original case, strip whitespace from start & finish
>
> Output >> toc.json = { <chapter_number_text> : <chapter_title_text> }
>
> Eval >> F1 score based on exact text match at char level – anything else is a false positive

Task 2 – Use regex to extract every question from the chapter text and serialize as a text file saved to current working dir.

Input >> chapter

Format of text extraction >> keep original case, strip whitespace from start & finish, strip any wrapping quotes from start and end of a question

Output >> questions.txt = each line is a separate extracted question

Eval >> F1 score based on exact text match at char level – anything else is a false positive

Task 3 – Train a CRF model to tag named entities using the ontonotes dataset as a training corpus. Run the CRF model and serialize as JSON a set of named entities (only return NE types DATE, CARDINAL, ORDINAL, NORP) for the chapter text to current working dir.

Input >> chapter, ontonotes dataset

Format of text extraction >> lower case, strip whitespace from start and finish of entries

Output >> ne.json = { <NE_type> : [<NE_phrase>, <NE_phrase>, …] }

Eval >> F1 score based on exact text match at char level – anything else is a false positive

Task 4 – Use CRF model and regex to extract every character name from the chapter text and serialize as a text file saved to current working dir.

Report all extracted forms of character names, so for text “When Mr Square went to the shops, Square’s face lightened up’ would provide “mr. square” and “square” as character names.

Input >> chapter, ontonotes dataset

Format of text extraction >> lower case, strip whitespace from start and finish of entries

Output >> characters.txt = each line is a separate extracted character name

Eval >> F1 score based on exact text match at char level – anything else is a false positive

Testing your code (to avoid wasted submission attempts)

The automated test harness which will run your code is a standard ECS VM running Red Hat Enterprise Linux 8 VM. The following commands were used to setup python 3.9 on the test harness VM. No other python packages can be used.

```
sudo yum -y install gcc gcc-c++ python39-devel

sudo yum install python39 python39-pip

sudo python3.9 -m pip install --upgrade pip

sudo python3.9 -m pip install -U nltk

sudo python3.9 -m pip install numpy

sudo python3.9 -m pip install scipy

sudo python3.9 -m pip install sklearn

sudo python3.9 -m pip install sklearn_crfsuite

sudo python3.9 -m pip install pandas

python3.9

      import nltk

      nltk.download('all')
```

It is strongly recommended you test with an **identical** setup otherwise your submissions may fail with errors. Any errors are your responsibility to fix, and lecture staff will **not** provide software support to you. An iSolutions guide to requesting an ECS VM is available here

http://ecs.gg/remote

The only exception to this no support rule is if the module test harness itself fails for some reason (preventing submissions from being evaluated). The test harness has been set to run on VM reboot, so any downtime is likely to be quickly resolved within 1 hour by iSolutions. Lecturer staff will investigate if downtime is longer than this at the earliest opportunity in the working day.

Your submitted code (same for all tasks) will be executed by the automated test harness using the below command line (with placeholders replaced with full paths to hidden resources located on the VM):

```
python3.9 task1_submission.py <ontonotes_parsed.json>
<eval_book.txt> <eval_chapter.txt>
```

You should locally check your code works prior to submission using the following command line:

```
python3.9 task1_submission.py ontonotes_parsed.json
eval_book.txt eval_chapter.txt
```

Memory limit for jobs is enforced using a linux bash shell to avoid crashing the VM. You can do this yourself on your test VM's to check it does not run out of memory. The evaluation VM shell script has a memory limit of 7Gbytes. Submission failures due to memory errors will decrease your submission attempts - it is your responsibility to test your models prior to submission.

Note: iSolutions for 2023 semester 2 are upgrading all student VM requests to be 8Gbytes, up from the 4GBytes they offered in 2022. Let your module leader know if this does not happen for some reason by week 5 (and use a 4Gbyte VM for testing whilst module leader investigates).

Create your own shell script my_shell_script.sh for testing:

```
#!/bin/bash

# memory limit in kilobytes (7340032 = 7G)

ulimit -m 7340032 -v 7340032

# run python code

python3.9 task1_submission.py ontonotes_parsed.json
eval_book.txt eval_chapter.txt
```

Baseline app indicative F1 scores for tasks

The evaluation harness will feedback your task F1 scores. When marking, which is manually done by module team at the end of task using a spreadsheet, your best F1 score is positioned against a baseline app F1 score and a gold F1 score (based on academic judgement around how difficult the book is each year) to calculate the final mark. Doing as well as the baseline app will result in a pass with around a 50% mark. Beating the gold F1 score target will result in a 100% mark. You should aim to do better than the baseline app F1 scores.

Your mark is not dependant on other student performance, and if everyone gets a top F1 score everyone gets top marks. The marking spreadsheet will not be shared. The baseline app will not be shared. The baseline and gold F1 scores will not be shared.

Below are a set of task-based baseline app F1 scores for each task, taken from the 2022 evaluation book. Remember each year will have a different hidden eval book, and it might be easier or harder than the 2022 book for each task. These indicative scores are only there to give a rough indication of where you are compared to the baseline application. The marking scheme has details of how F1 scores compared to the baseline application F1 score are converted to marks. Put the hours in and do your best, as this is how you will develop your practical skills as a key module learning outcome.

Baseline app and gold F1 scores for 2022

Task1 baseline app F1 score 0.5, gold F1 score 0.95, top student F1 score 1.0

Task2 baseline app F1 score 0.77, gold F1 score 0.95, top student F1 score 1.0

Task3 baseline app F1 score 0.43, gold F1 score 0.60, top F1 score 0.71

Task4 baseline app F1 score 0.45, gold F1 score 0.70, top F1 score 0.88

Remember, each year it's a **different book** so the baseline and top F1 score will be a higher or lower than the 2022 scores (usually +/- 0.2 for task 3 and task 4 depending on how tricky the entities are in the hidden eval chapter).

Submission

You will submit your 'task<N>_submission.py' code files via handin. Up to 20 submission attempts are allowed per student per task. Failed submissions, for whatever reason, **will count** towards the 20 total submissions.

Each submission 'task<N>_submission.py' file will be automatically run using as input a text corpus which is hidden from students. The text corpus will be selected from one of the books/chapters that can be freely downloaded from https://www.gutenberg.org/

The submissions will generate task files {toc.json, questions.txt, ne.json, characters.txt} in the current working directory.

Your submission code **must** finish running within 30 minutes. If it overruns it will be automatically hard killed and the submission will be returned as 'timed out'. Timed out sessions count as failed submissions, and count towards your total submission count.

Task files will be automatically evaluated against a hidden manually labelled ground truth and an F1 score computed.

Submissions are evaluated in a queue. One submission is evaluated at a time. You will be emailed where you are in the queue. As each submission can take up to 30 minutes, if you were 10^{th} in the queue you can expect your submission to finish being evaluated after 10 x 30 minutes = 5 hours. It is strongly recommended you submit some attempts several weeks **before the deadline** to avoid being stuck in a long queue near the handin deadline date.

Late penalties are applied based on handin submissions time, NOT queued evaluation time, so if you do your final submit before the handin deadline you will not be labelled as late.

Late submissions

Late submissions will be penalised according to the standard rules.

The handin submission time is based on your last submission, so if you submit after the handin deadline you will incur a late penalty (even if you have submission attempts registered before the deadline).

Academic Integrity and playing fair

Your submitted code must be your own work. It can however re-use parts of the taught material (e.g. module lab code) as a starting point if this is helpful. Re-use of code from sources outside this module (e.g. github) is not permitted as code must be your own work. You can learn from online tutorials and github but you need to write your own code.

Your code must not download any additional resources or install any additional software packages on the text harness VM.

Any attempt to recover or email the hidden corpus (book and chapter) or ground truth evaluation files on the automated test harness VM is considered cheating. Printing extracts from the hidden book or chapter to stdout to inspect it via the automated evaluation results email is not allowed. If you do somehow accidentally print hidden book or chapter text to stdout then you must immediately email the module leader and self-report your mistake.

Your code must generate your models and results dynamically. It is not allowed to hardcode a lookup table of potential answers from a manual crawl of Gutenberg book store, for example. Regex patterns can contain hardcoded lexical terms but they should not be book specific.

Every code submission attempt is kept, along with the email result after evaluation. Code will be manually inspected and/or run past code checkers and attempts to cheat will trigger an academic integrity case.