**Samtla-Char-NER Report**

**Implementation of Character-based Named Entity Recognition into the Samtla System**

*Student: Matthew Ralph,*

*MSc Computer Science project report, Department of Computer Science and Information, Birkbeck College University of London*

[*mralph02@dcs.bbk.ac.uk*](mailto:mralph02@dcs.bbk.ac.uk)

*Supervisor: Dr Dell Zhang*

*With thanks to Dr Martyn Harris*

*This report is substantially the result of my own work, expressed in my own words, except where explicitly indicated in the text. I give my permission for it to be submitted to the JISC Plagiarism Detection Service. The report may be freely copied and distributed provided the source is explicitly acknowledged.*

# Abstract

Recent approaches to Named Entity Recognition (NER), such as that of (Kuru, Arkan Can and Deniz, 2016), demonstrate that a character-level representation of textual data can yield good results when training a deep learning model. In this project, a set of Hansard debates is aggregated, processed and labelled for use in a Bidirectional Long Short-Term Memory (BLSTM) neural network. The trained model, and the original dataset, is submitted for integration with Birkbeck’s Samtla digital humanities text archiving system, such that the Hansard texts can be browsed in the interface, and Named Entities previously unseen by the model, are recognised using word-internal (character clusters) and word-external (language context) clues and annotated to the user in the user interface. As part of this project, another simple graphical frontend, not coupled with Samtla, is built just to demonstrate the Named Entity annotations.

# Acknowledgements

I would like to express my gratitude to the people who taught me to program in Python by working on real problems, particularly to Ali Lotia and Ogonna Iwunze, whose expertise is matched only by their patience and compassion. I would also like to thank Sergio Gutierrez-Santos, whose course in the Java programming language was carefully designed and helped to open up a world of structured code for me, as well as demystifying unit testing.

I am grateful to Dr Martyn Harris for his help and encouragement when exploring this project and its potential integration with Samtla, and to Dr Dell Zhang for his ideas, advice on the academic landscape surrounding Named Entity Recognition, and quick responses to my queries. Petar Konovski’s quick and repeated assistance in setting up a server in Birkbeck with the right library dependencies greatly simplified the process of moving my code onto a suitable sized server, and I am grateful to Systems Group for the use of such a beefy machine.

Finally, I would like to thank my wife for all her help throughout this Master’s programme, while she worked on her own Master’s and continued to support so many people.

*Ad Maiorem Dei Gloriam*.

# Lists of figures, tables and code snippets

## Figures

[Figure 1 pipeline data processing model 8](#_Toc523753103)

[Figure 2 chunking process 11](#_Toc523753104)

[Figure 3 Interpolation algorithm 14](#_Toc523753105)

[Figure 4 Hansard debate, XML format 21](#_Toc523753106)

[Figure 5 Hansard debate, processed TXT format 21](#_Toc523753107)

[Figure 6 Mini dataset accuracy 23](#_Toc523753108)

[Figure 7 Mini dataset loss 24](#_Toc523753109)

[Figure 8 Mini dataset non-null label accuracy 24](#_Toc523753110)

[Figure 9 Toy dataset NaN validation accuracy 25](#_Toc523753111)

[Figure 10 Toy dataset NaN validation loss 25](#_Toc523753112)

[Figure 11 Toy Dataset NaN Validation Non-Null Label Accuracy 25](#_Toc523753113)

[Figure 12 ToyV2 dataset accuracy 27](#_Toc523753114)

[Figure 13 ToyV2 dataset loss 27](#_Toc523753115)

[Figure 14 ToyV2 dataset non-null label accuracy 27](#_Toc523753116)

[Figure 15 ToyV3 dataset accuracy 28](#_Toc523753117)

[Figure 16 ToyV3 dataset loss 28](#_Toc523753118)

[Figure 17 ToyV3 dataset non-null label accuracy 29](#_Toc523753119)

[Figure 18 Example Hansard text before NE annotations - SimpleGUI 33](#_Toc523753120)

[Figure 19 Example Hansard text after NE annotations - SimpleGUI 33](#_Toc523753121)

[Figure 20 Simple-GUI basic design 34](#_Toc523753122)

[Figure 21 Actual time spent on each work package of project 36](#_Toc523753123)

[Figure 22 Simple-GUI index page 39](#_Toc523753124)

[Figure 23 Simple-GUI date page 40](#_Toc523753125)

[Figure 24 Simple-GUI debate page with an annotated paragraph. 40](#_Toc523753126)

## Tables

[Table 1 % of NE data from DBPedia 9](#_Toc523753127)

[Table 2 X tensor dimensions 17](#_Toc523753128)

[Table 3 Y tensor dimensions 17](#_Toc523753129)

[Table 4 DBPedia post-processing tasks on Named Entities 18](#_Toc523753130)

[Table 5 Model datasets used 22](#_Toc523753131)

[Table 6 Baseline accuracy 22](#_Toc523753132)

[Table 7 Evaluations of trained models 22](#_Toc523753133)

[Table 8 k-fold cross-validation results 31](#_Toc523753134)

[Table 9 Original work plan given in proposal 35](#_Toc523753135)

[Table 10 List of Invoke tasks used to drive the pipeline 41](#_Toc523753136)

[Table 11 List of source code with author 44](#_Toc523753137)

## Code snippets

[Code Snippet 1 NLTK Punkt tokenizer prepared with some common abbreviations. 10](#_Toc523753138)

[Code Snippet 2 Interpolated Hansard text sample 12](#_Toc523753139)

[Code Snippet 3 Converting bucket numbers to datasets 16](#_Toc523753140)

[Code Snippet 4 logic to avoid re-interpolating overlapping NEs 19](#_Toc523753141)

[Code Snippet 5 Training on the Full dataset 30](#_Toc523753142)

[Code Snippet 6 k-fold cross validation 30](#_Toc523753143)

[Code Snippet 7 Python subclassing to add model evaluate 31](#_Toc523753144)

[Code Snippet 8 a regular Pytest unit test 32](#_Toc523753145)

[Code Snippet 9 A Pyfakefs Pytest unit test 32](#_Toc523753146)

Table of Contents

[0](#_Toc523753597)

[1. Abstract 1](#_Toc523753598)

[2. Acknowledgements 1](#_Toc523753599)

[3. Lists of figures, tables and code snippets 2](#_Toc523753600)

[3.1. Figures 2](#_Toc523753601)

[3.2. Tables 2](#_Toc523753602)

[3.3. Code snippets 2](#_Toc523753603)

[5. Introduction 5](#_Toc523753604)

[6. Overall Results 6](#_Toc523753605)

[7. Software Architecture 7](#_Toc523753606)

[7.1. The pipeline of tasks 7](#_Toc523753607)

[7.2. Python “Invoke” framework 7](#_Toc523753608)

[7.3. Named Entity downloading 10](#_Toc523753609)

[7.4. Raw Hansard downloading 10](#_Toc523753610)

[7.5. Hansard processing 11](#_Toc523753611)

[7.6. Hansard chunking 11](#_Toc523753612)

[7.7. Hansard interpolation 13](#_Toc523753613)

[7.8. Hansard numerification 14](#_Toc523753614)

[7.9. Partition into datasets and sizes 16](#_Toc523753615)

[7.10. Formation of tensors 18](#_Toc523753616)

[8. Implementation issues 18](#_Toc523753617)

[8.1. Wikipedia data cleanliness 18](#_Toc523753618)

[8.2. Interpolation overlaps 20](#_Toc523753619)

[8.3. NLTK Treebank word span\_tokenize bugs 20](#_Toc523753620)

[8.4. Toy dataset model – tensor formation 21](#_Toc523753621)

[8.5. Hansard presentation issues 21](#_Toc523753622)

[9. Evaluation and Testing 22](#_Toc523753623)

[9.1. Model evaluation 22](#_Toc523753624)

[9.1.1. Baseline results 23](#_Toc523753625)

[9.1.2. The ‘mini’ dataset 24](#_Toc523753626)

[9.1.3. The ‘toy’ dataset, version 1 25](#_Toc523753627)

[9.1.4. The ‘toy’ dataset, versions 2 and 3 27](#_Toc523753628)

[9.1.5. The full dataset 30](#_Toc523753629)

[9.1.6. Cross-validation 31](#_Toc523753630)

[9.2. Unit testing 32](#_Toc523753631)

[9.3. Overall evaluation 33](#_Toc523753632)

[10. Graphical User Interface 34](#_Toc523753633)

[10.1. Simple-GUI 34](#_Toc523753634)

[10.2. Samtla Integration 36](#_Toc523753635)

[11. Planning 36](#_Toc523753636)

[12. Summary and Conclusions 37](#_Toc523753637)

[12.1. Pre-processing is hard 37](#_Toc523753638)

[12.2. Labelling is hard 38](#_Toc523753639)

[12.3. Sentence tokenization is hard 38](#_Toc523753640)

[12.4. Neural networks are slow and opaque 38](#_Toc523753641)

[12.5. Future work 39](#_Toc523753642)

[13. References 39](#_Toc523753643)

[14. User Manual 39](#_Toc523753644)

[14.1. Data pipeline manual 40](#_Toc523753645)

[14.2. Simple-GUI Manual 40](#_Toc523753646)

[15. Appendix A: List of Invoke tasks 41](#_Toc523753647)

[16. Appendix B: What’s My Work 45](#_Toc523753648)

[17. Appendix C: Code 48](#_Toc523753649)

[17.1. tasks.py 48](#_Toc523753650)

[17.2. config\_util/config\_parser.py 54](#_Toc523753651)

[17.3. hansard\_gathering/chunk.py 54](#_Toc523753652)

[17.4. hansard\_gathering/driver.py 57](#_Toc523753653)

[17.5. hansard\_gathering/filesystem.py 59](#_Toc523753654)

[17.6. hansard\_gathering/interpolate.py 60](#_Toc523753655)

[17.7. hansard\_gathering/numerify.py 65](#_Toc523753656)

[17.8. hansard\_gathering/preprocessing.py 66](#_Toc523753657)

[17.9. keras\_character\_based\_ner/src/matt/alphabet\_management.py 67](#_Toc523753658)

[17.10. keras\_character\_based\_ner/src/matt/dataset\_hashing.py 68](#_Toc523753659)

[17.11. keras\_character\_based\_ner/src/matt/eval.py 70](#_Toc523753660)

[17.12. keras\_character\_based\_ner/src/matt/file\_management.py 73](#_Toc523753661)

[17.13. keras\_character\_based\_ner/src/matt/history.py 77](#_Toc523753662)

[17.14. keras\_character\_based\_ner/src/matt/minify\_dataset.py 78](#_Toc523753663)

[17.15. keras\_character\_based\_ner/src/matt/model\_integration.py 79](#_Toc523753664)

[17.16. keras\_character\_based\_ner/src/matt/persist.py 84](#_Toc523753665)

[17.17. keras\_character\_based\_ner/src/matt/predict.py 86](#_Toc523753666)

[17.18. keras\_character\_based\_ner/src/matt/save.py 88](#_Toc523753667)

[17.19. keras\_character\_based\_ner/src/matt/train.py 88](#_Toc523753668)

[17.20. ne\_data\_gathering/companies.py 90](#_Toc523753669)

[17.21. ne\_data\_gathering/people.py 93](#_Toc523753670)

[17.22. ne\_data\_gathering/places.py 95](#_Toc523753671)

[17.23. ne\_data\_gathering/util.py 97](#_Toc523753672)

[17.24. simple\_gui/simple\_gui.py 100](#_Toc523753673)

[17.25. simple\_gui/util.py 101](#_Toc523753674)

[17.26. simple\_gui/static/char-ner.js 102](#_Toc523753675)

[17.27. simple\_gui/static/style.css 102](#_Toc523753676)

[17.28. simple\_gui/templates/date.html 103](#_Toc523753677)

[17.29. simple\_gui/templates/debate.html 103](#_Toc523753678)

[17.30. simple\_gui/templates/index.html 103](#_Toc523753679)

[17.31. test/test\_companies.py 104](#_Toc523753680)

[17.32. test/test\_interpolate.py 104](#_Toc523753681)

[17.33. test/test\_matt.py 105](#_Toc523753682)

[17.34. test/test\_model\_integration.py 105](#_Toc523753683)

[17.35. test/test\_simple\_gui\_util.py 105](#_Toc523753684)

[17.36. test/test\_util.py 106](#_Toc523753685)

# Introduction

The brief for this project was to demonstrate Named Entity Recognition, using the approach cited in (Kuru, Arkan Can and Deniz, 2016), and the Keras implementation of this provided by GitHub user 0xnurl.[[1]](#footnote-1) The target dataset was the Hansard, the record of debates in both of the houses of Parliament in the United Kingdom.[[2]](#footnote-2) This dataset is now available via the Parliament UK Data API,[[3]](#footnote-3) however this API is largely undocumented and was not available at the start of this project. Instead, I used the They Work For You API,[[4]](#footnote-4) which has all debates from 1919 onwards available for download in a parsed XML format annotated with metadata about the speaker. I did not have time to use this high-quality metadata during the project. However, I am grateful for free use of this API which certainly made data preparation easier for me.

In order to implement the model, I had to produce labelled Hansard data. To manually label a few thousand debate documents, required to train even a very basic model, would have been too time-consuming for a project of a few months. So, I used a form of automatic labelling I refer to henceforth as ‘interpolation’, the algorithm for which is explained in section 7.7. Interpolation relied on me having a very large set of Named Entities in my chosen categories or locations, organizations and people. I used the DBPedia SPARQL endpoint[[5]](#footnote-5) and Python’s SPARQLWrapper library[[6]](#footnote-6) to download all the Named Entities on Wikipedia in these categories. There were some data cleanliness issues with this dataset that I never fully overcame, which are detailed in section 8.1.

I used the interpolated (labelled) Hansards to generate Y tensors for training. The processed Hansard debates themselves were segmented (or, as I refer to it below, “chunked”) into sentences, and then each character was converted to a number, to create X tensors. I then used the 0xnurl implementation to train the BLSTM model. An overview of results is given in section 6.

I chose this project because of my interest in linguistics and in humanities texts. In my first degree, Classics, I was fortunate to study linguistic change from Classical Greek to *koine*, the language of the New Testament. I also studied some phenomena of Latin that are markers of a particular gender or class. This project is a tiny step, greatly helped by the labours of TheyWorkForYou and 0xnurl, towards exploiting the value of the Hansard records.

This project is also in part politically motivated. It is vitally important that democratic citizens re-engage with the task of using factual analysis and solid statistics to make important decisions, rather than being emotionally stirred by the language of tyrants. This is a far older problem than the Romans and Ancient Greeks. Learning which companies, places and people we spend most energy talking about as a democracy seems to me, in its own small way, a part of this enormous and essential task.

# Overall Results

The sequence of models described in section 9.1 show a gradual improvement in the model’s capability to recognise Named Entities, peaking at a non-null label accuracy (accuracy for Named Entity labels excluding the null label) of 0.5532. While a far cry from the F1 scores of 0.7-0.8 seen on various languages in (Kuru, Arkan Can and Deniz, 2016),[[7]](#footnote-7) both categorical accuracy and non-null label accuracy in trained models ToyV2 and ToyV3 comfortably beat the ‘baseline’ evaluation scores of always guessing ‘null’, or always guessing one particular NE, as shown in section 9.1.1 (section 9.1 contains a full description of all the trained models). My model beats the baseline by 5.6% for categorical accuracy,[[8]](#footnote-8) and by 9.7% for non-null label accuracy.[[9]](#footnote-9) These results are also noteworthy as they depend purely on an automated labelling algorithm which has no human tagging or gold data.

As set out in the proposal, this project also achieves automated downloading, processing and labelling of the whole Hansard corpus in an automated, code-driven fashion,[[10]](#footnote-10) and the loading of this dataset into the Birkbeck Samtla system. Finally, the approach to Graphical User Interface (GUI) integration is proved in a small sample interface called Simple-GUI, which provides and demonstrates the necessary client-side Javascript and backend API for integration into Birkbeck’s Samtla interface.

# Software Architecture

## The pipeline of tasks

The preparation of Hansard documents is, in essence, a data pipeline. Hansard data is sourced from the They Work For You API and Named Entities are sourced from a few different sources. They are combined using a variety of algorithms, and then stored in a format that can then be used to predict unseen named entities. As such, it is best visualised using a pipeline flow (see Figure 1). Each element of the pipeline is introduced in more detail in the sections below, along with details of the algorithms and data storage mechanisms used. Implementation difficulties are discussed in section 8.

The outputs of each pipeline step were persisted to disk, either as simple text files, Python “pickle” objects in binary format, or Keras’ H5 binary output format. Such persisting is essential when working with a large amount of data, firstly to allow each stage of the output to be validated and checked, and also to ensure the whole pipeline would not need to be run (which takes several days) every time a bug is discovered. As most of the functions in the pipeline do not return pure values but write their results out to disk (using ‘print’ statements only to inform the user of their progress), a unit testing approach was needed that could fake a UNIX filesystem in order to validate the functions were working as expected. This is explained in section 9.2.

Were this system to be ‘productionised’, then all the stages of this flow would ideally be run through a Continuous Integration system such as Jenkins or GoCD which would run the different stages in the correct sequence; in such an arrangement, new Named Entity data and newly produced Hansard records could be fed into the pipeline and used to continuously update the model to account for new data, taking advantage of Keras’ facility to load in a model from disk, and train it with new data.

## Python “Invoke” framework

As so much of this project’s effort was in collecting data for pre-processing, a command-line driven front-end was preferred over building a Graphical User Interface (GUI) just for the internal tasks of gathering, processing and aggregating data. Given the pipeline structure of the project, it was essential to have a tool that would allow code execution to start at any point in the pipeline, with all the correct dependencies in place, having run any prerequisite tasks required.

Invoke[[11]](#footnote-11) was chosen, after some experimentation with Argh,[[12]](#footnote-12) Shovel[[13]](#footnote-13) and Doit.[[14]](#footnote-14) Invoke was found to support arbitrary library imports from the Python global library and the current project, whereas Doit manipulated the user’s PYTHONPATH and so could not be integrated with a project structured into modules. Invoke also supports the basic features for which one might use a Makefile – a simple command line front-end providing many possible entrypoints into an application, with listed prerequisite tasks which could be called with specified, or default, arguments. In contrast to using Make, the task file itself (tasks.py in the code listing, given in section 17) is in pure python and does not require tab characters for delineation. Most calls in this project’s “tasks.py” are simply Python library imports and function executions, but some required separate command-line invocations e.g. to start PyTest or MyPy (for unit testing and static type analysis, respectively). Invoke natively supports this much more elegantly than Argh or Shovel. A full list of Invoke tasks and their descriptions is found in section 15.

Step numbers in the sections below refer to the blue numbers in Figure 1.



Figure 1 pipeline data processing model

## Named Entity downloading

**Step 1.** Firstly, named entities must be accrued. This is a prerequisite for any automated labelling approach. For locations, the CONLL2003 English dataset was used, together with DBPedia resources of type ‘dbo:Place’. For Organizations, the Amex, Nasdaq and NYSE Stock Exchange company listings were downloaded in Comma Separated Value (CSV) format, as was similar data from the London Stock Exchange, the CONLL2003 English dataset, and DBPedia’s ‘dbo:Organisation’ type. For people, the CONLL2003 English dataset and DBPedia ‘dbo:Person’ type were used, and the New York City Most Popular Baby Names data from Kaggle.[[15]](#footnote-15) Biography-center.com, which was suggested as a naming source by (Klein *et al.*, 2003), no longer has lists of names in an easily-parseable format. As Table 1 shows, the size of the DBPedia datasets dwarf the other datasets for all three Named Entity (NE) types.

Table 1 % of NE data from DBPedia

|  |  |
| --- | --- |
| **Dataset** | **% from DBPedia** |
| Locations | 99.8 |
| Organizations | 96.8 |
| People | 99.7 |

**Step 2.** The resulting data had to be cleaned to remove stopwords and some of the more obvious junk data. The data quality issues with the NE datasets are discussed in section 8.1.

**Step 3.** Simple UNIX utilities ‘cat’ and ‘sort’ were used to deduplicate the aggregated NE lists, and sort them into a large text file for each NE type. UNIX utilities (compiled from C) were preferred to Python because of their superior performance.

## Raw Hansard downloading

**Step 4.** To download the Hansards in a programmatic manner, the TheyWorkForYou API was chosen. This was on the basis of its high-quality documentation, and the availability of Hansard debates in XML format, with enriched metadata tags naming each speaker and detailing their constituency and party. Unfortunately, there was not time in this project to make use of this extra metadata.

Python’s concurrent.futures.ThreadPoolExecutor implementation was chosen to increase the speed of downloads as this activity is mainly bound by network I/O. Python’s concurrent library modules simply need to be invoked in a loop, and passed a Python Callable object, along with parameters – no manual thread handling code is needed.

## Hansard processing

The files downloaded from TheyWorkForYou are XML files with a lot of markup and metadata which would confuse a model aiming to learn NEs. After failed attempts with bleach.clean,[[16]](#footnote-16) which fails to remove nested HTML tags, the lxml library’s etree module[[17]](#footnote-17) was successfully used to remove all markup and preserve just the text of the debates. In order for lxml to accept the XML files and process them, the encoding of the files and the lxml library’s config had to be set to use UTF-8. Hansard debates use a wide range of characters, including accented letters like é as well as abbreviations like ¾, so it makes sense to pick the most widely-used Unicode encoding standard. This is one of several encodings supported by Samtla, so compatibility with that system was preserved.

## Hansard chunking

**Step 6.** I use the term ‘chunking’ throughout this report and the codebase, to refer to the process of sentence-segmentation. This avoids any confusion with the word-segmentation tokenizer, which is used in the interpolation algorithm (see section 7.7). The ‘chunker’ used is the NLTK Punkt sentence tokenizer. However, early testing showed that it struggled with the abbreviations used in Hansard, in particular ‘hon.’, which occurs frequently as a shortening of ‘honourable’. The Punkt tokenizer would view this as the end of a sentence, particularly as it often occurred in the context of ‘the hon. Gentleman’, with the following word capitalised.

Training a sentence segmenter on Hansard data with sentence markers would be a project in itself, so I merely passed several common abbreviations to the chunker, as shown in Code Snippet 1.



Code Snippet 1 NLTK Punkt tokenizer prepared with some common abbreviations.

The format originally chosen was simply to write out a new file for each sentence of each debate, and to auto-generate file numbers such that, if a debate was called ‘Public Sector Pay.txt’, the generated sentences would occupy files called ‘Public Sector Pay-chunk-0.txt’, ‘Public Sector Pay-chunk-1.txt’, etc. This format was just as quick to generate but used a huge amount of disk space. Indeed, I had only processed debates as far as May 1966 when the 200GB of space allocated for this project on my laptop ran out. On further investigation, it was noted that the Mac OS HFS+ filesystem will allocate 4k for any new file, as this is its minimum block size. Hence, the sentence segmenting algorithm was creating a large number of very small files (there are millions of sentences in the total dataset, see Table 5). The minimum size of these files was 4KB each, but the maximum was as large as the longest sentence.

I changed approach to use a single file to store just spans, named the same as the original debate file but ending in -spans.txt. This file contained new-line separated tuples of character-offsets for each sentence start and finish, as Figure 2 shows.



Figure 2 chunking process

## Hansard interpolation

**Step 7.** The interpolation algorithm is detailed in Figure 3. Even though the deep learning model we are using is character-based and has no knowledge of word-boundaries, for the interpolation the NLTK Treebank word-tokenizer was used. The reason for this was performance.

The challenge for interpolation was to find an algorithm that could match against a Python set object (to take advantage of the hashing-based implementation of sets in Python and avoid the full scan that a list would require), while also making the longest possible match. For example, as “Tonbridge” and “Tonbridge Wells” are different locations, we want to ensure that the longer match is found even if the shorter match would be found first with a simple scan through the text. Similarly, even though “Paris” is a location, “Paris Hilton” is a person and should be identified first, in order to annotate the text with the correct NE label.

An n-grams approach is taken. For each text, all n-grams are generated using Treebank’s span tokenizer (there were bugs found in this approach – see section 8.3). The default value used for n was 4, and as such 4-word NEs are the longest that we can interpolate. The n-grams are right-padded with the Python nil object, ‘None’ (see the Figure 3), so that Named Entities that are less than n words from the end of a sentence can still be matched. We then take each possible suffix of the n-gram, starting with the longest, and attempt to match it against all Named Entity lists – first locations, then organizations, then people.

The end result looks like Code Snippet 2. Note how ‘Railtrack’ has been identified as an Organization (represented by integer ‘2’), while the rest of the phrase is assigned the ‘NULL’ label 0. The text is rendered with its interpolated labels underneath it, using a helper function “hansard-display-interpolated-file” to line up the characters. For a complete list of all Invoke tasks written for this project, see Section 15.



Code Snippet 2 Interpolated Hansard text sample

The main problem encountered with interpolation was that of overlapping Named Entities. This is described, along with its solution, in section 8.2. The interpolation algorithm was designed solely to provide better-than-nothing labelling. For high quality labelling, human work would be required. The problems with this automatic labelling approach are discussed more generally in section 12.2.

## Hansard numerification

**Step 8.** To generate the X tensor, the debate texts themselves had to be converted to Numpy arrays of integers. A Python “CharBasedNERAlphabet” object was generated from the debate texts using hashing buckets as described in section 7.9, and then a simple lookup against this alphabet was used to convert individual Unicode codepoints into integers from the alphabet. This results in much smaller tensors than simply using the Unicode codepoint value directly, as Unicode has a total of 137,374 characters,[[18]](#footnote-18) requiring a 3-byte integer to store. The CharBasedNERAlphabet had only 160 characters so was much more compact. Any characters seen in numerification of Hansard, which were not seen in creation of the alphabet, were assigned the number for <UNK>, the unknown character.



Figure 3 Interpolation algorithm

## Partition into datasets and sizes

Any machine learning model requires dataset for training, for configuration of hyperparameters and for evaluation. In this project, these datasets were named ‘train’, ‘dev’ and ‘test’ respectively. Note that in 0xnurl’s Keras model, the ‘dev’ dataset was used to provide accuracy and loss scores at the end of each epoch, used to determine whether training should continue. I used the ‘test’ set solely for evaluation of models once their training was completed.

It is important that the divisions used for these datasets are fairly distributed and do not contain any biases. For instance, if the whole dataset of debates were treated as one linear list from 1919 to the present day, with contiguous segments used for each dataset, this would be a biased distribution, because the use of language changes over time. The model could be trained on early 20th Century English but then validated on 21st Century English, skewing the results.

To avoid this, all the debates’ file names were hashed using Python’s built-in hashing implementation. The file names include the date on which the debate was spoken in parliament, and the subject of the debate. These file names are strings, and Python’s built-in hash function takes each character, converts to an integer, and then uses exponentiation and addition to combine them. The modulo of the resulting integer was taken with respect to the number of buckets (which was set at 320), resulting in buckets of equal numbers of debates which are evenly distributed with respect to time. The contents of each bucket are saved in bucket-list files in the project, so the same datasets can be used consistently (for example, we never want to use data in the “test buckets”, even for manual validation).

In order to convert the 320 numbered buckets into datasets, a simple function was used, which is shown in its entirety in Code Snippet 3. Note that, while we hashed the whole interpolated dataset (all debates from 1919 to August 2018) into 320 buckets, the function in the code snippet only uses the first 8 buckets. This is referred to as the “Toy” dataset (see section 9.1.3 for a detailed description of all the datasets). This is because using the full dataset resulted in out-of-memory errors from Keras, as explained in section 9.1.5.



Code Snippet 3 Converting bucket numbers to datasets

Note also that the hash-bucketing technique is also used to build an alphabet for the model. When training is started, a set of debates is read in from disk, and the Unicode characters used in those debates are unioned together to make a set which initializes the CharBasedNERAlphabet object used to convert debate texts into a stream of integers for the X tensor (**Step 8** in Figure 1, and referred to as ‘numerification’ in the codebase. See section 7.8 above). To read in all 66,459 debate files to generate such an alphabet is wasteful – characters that are not part of the standard Roman alphabet or common English punctuation occur very rarely and give us very few clues about Named Entities. The characters in use, unlike the lexicon, change very little over the decades, so originally the alphabet object was simply built off all the debates from an arbitrary year (1949 to begin with). However, a more principled approach was to use all the debates from one bucket to create the alphabet object. All characters encountered that are not in the alphabet are given the integer for <UNK>, the unknown symbol.

As hoped for, each bucket contains roughly 205 debates, with a standard deviation across all the buckets of 15.5. One consequence of this approach is that all the ‘chunks’ (sentences) in a given debate are placed in the same dataset. That is, for each Hansard debate *h* that exists in a dataset *d* (be it train, test or dev), *all* of *h*’s sentences are found in dataset *d* and none of them are found in a different dataset. This does not seem to present a problem – the main motivation of the bucket-hashing approach was to ensure the datasets’ textual data is distributed across time.

One risk, however, is that longer debates have more sentences – so datasets which happen to have longer debates in, will have more data in them. During the course of this project, this did not present itself as a problem.

## Formation of tensors

**Step 10.** The X and Y tensors are generated from the numerified and interpolated Hansard data respectively. Both are constructed as native Python nested lists by reading from their respective data sources, and then processed using Keras’ pad\_sequences helper function. This accomplishes three things: it left-pads sequences shorter than sentence\_maxlen with 0s, it truncates any sequence longer than sentence\_maxlen, and it converts the nested list structure to Numpy nested arrays of the correct datatypes. The dimension contents for the X and Y tensors are shown in Table 2 and Table 3.

Table 2 X tensor dimensions

|  |  |  |
| --- | --- | --- |
| **Dimension** | **Content** | **Length** |
| 1 | Text Samples | Batch-size length (varies depending on dataset) |
| 2 | Characters | Sentence\_maxlen (200) |

Table 3 Y tensor dimensions

|  |  |  |
| --- | --- | --- |
| **Dimension** | **Content** | **Length** |
| 1 | Text Samples | Batch-size length (varies depending on dataset) |
| 2 | Characters | Sentence\_maxlen (200) |
| 3 | One-hot array of labels | Number of labels (4) |

For the toy dataset, the Numpy arrays are pickled to disk. This is so they can be used in multiple model training runs with different hyperparameters, without having to regenerate the tensors. The Keras model used from 0xnurl is modified only slightly, so that the dataset’s get\_x\_y function calls a function in the ‘matt’ package, representing a package of library files added to the Keras model as part of this project. My contributions are mainly placed in this package in order to clarify exactly what is my contribution to 0xnurl’s model. A list of all files in this project, who authored them and what they achieve, is found in section 16.

# Implementation issues

## Wikipedia data cleanliness

The datasets downloaded using the DBPedia SPARQL API are a result of volunteer contributions to Wikipedia article content and metadata. As a result, the data is both voluminous and not very clean. Duplicates in the data, like the presence of “Ralph Allwood” and “Ralph Allwood MBE”, are not a problem, as they will improve the coverage of interpolation. However, the DBPedia API contains entries like the following which are wrongly listed as people:

* “(15 July 1914 – 8 November 1927)”
* “(1833-1905)”

And the following are listed as organizations:

* “I” (the first-person pronoun)
* “.” (a single period)

And the following as places:

* “And the”
* “the”

In the case of “(15 July 1914 – 8 November 1927)”, this would appear to be the birth and death dates of a person, wrongly classified as a person’s name. The other examples just seem to be mis-classifications of command words or characters in English.

At first, such oddities were manually removed, but it was clear that a more principled filtering approach was needed. A number of processing steps were then added to the Named Entity lists once they were downloaded from DBPedia. These are listed in Table 4.

Table 4 DBPedia post-processing tasks on Named Entities

|  |  |
| --- | --- |
| **Task** | **Regex (if applicable)** |
| Remove double quotes | N/A |
| Left-trim whitespace | N/A |
| Remove lines that are entirely numbers of symbols | ^[!@£$%^&\*()0-9 ]+$ |
| If whole line starts and ends with brackets, remove them | N/A |
| If line starts with more than one single quote, remove all single quotes at start of line | (.\*)'{2,}$ |
| If line starts with just whitespace or asterisks, remove them | ^[\* ]+(.\*) |
| Remove words shorter than 4 characters | N/A |

Finally, if after all processing, all the remaining words in a Named Entity are stop-words (taken from the NLTK Corpus of English stop-words), then whole entry is removed. Of course, this means that some perfectly valid Named Entities like ‘The Who’ cannot be recognised in the interpolation phase. This is a necessary trade-off of cleaning up the data in an automated fashion. Note that words shorter than 4 characters are also removed, before the stopwords step. These tend to be strange stub words like ‘ar’ which are low-value and hard to filter.

DBPedia contains a lot of Chinese, Russian and Arabic names in their respective scripts. This is not a data cleanliness problem, just a phenomenon of Wikipedia’s global reach. There is no principled reason to remove these names from the dataset, but it is unlikely that they would appear in Hansard in their native character-sets.

## Interpolation overlaps

One problem of the early incarnation of this algorithm is that earlier Named Entities could be overwritten by later ones. For example, in the phrase ‘The House’, the two-word phrase may be successfully interpolated as a place (referring to the house in which the debate takes place). However, when the algorithm moves on to the word ‘House’, it will label it as an organization (House is the name of two different companies listed on Wikipedia). The resulting labelling for “The House” is 111122222, with ‘The’ still labelled as a location even though ‘House’ is re-labelled as an organization. Aside from the ambiguity about what the correct labels for the whole phrase are, ‘The’ is now definitely labelled wrongly.

The fix for this problem was to arbitrarily choose the first-matched Named Entity as the correct one. In the case of ‘The House’, the 2-word phrase is labelled as a location. This labelling is then protected – as the n-grams window slides forward to recognise more Named Entities, we keep track of whether the phrase being examined overlaps with a previously recognised Named Entity (to keep track of this, we store recentest\_match\_end, the index of the character at the end of the most recent labelling). If it does overlap, we skip over this n-gram without searching for any more Named Entities. Code Snippet 4 shows the logic (“overlaps” is a helper function which simply compares the first index of the current ngram with recentest\_match\_end and returns True if overlap occurs).



Code Snippet 4 logic to avoid re-interpolating overlapping NEs

## NLTK Treebank word span\_tokenize bugs

NLTK’s span\_tokenize has two open issues on GitHub, one of which was opened in August while this project was being written.[[19]](#footnote-19) “Span\_tokenize” cannot handle some inputs with unbalanced or nested quotation marks. The issue seems to stem from the implementation, which first tokenizes the text into individual words (not offsets), then hunts through the text for each tokenized word individually, in order to generate the offsets. The NLTK community on GitHub has submitted a number of fixes to the code used to match the text, but the conversation on the issue as a whole concluded[[20]](#footnote-20) that the only robust solution was to remove the span\_tokenize method completely.

This project is fully reliant on NLTK’s Treebank’s word span\_tokenize to generate offsets used to create ngrams to scour for Named Entities, as described in detail in section 7.7. The character-position indices have to be preserved, in order to generate a Y tensor with NE labels in the same positions as the original characters. In order to side-step the NLTK span\_tokenize bug, I searched for all files that had failed interpolation on the first iteration, replaced all occurrences of double-quotes with single-quotes, and then re-interpolated them, as shown in the top-left of Figure 3.

This approach is not ideal as it involves changing the raw textual data; given more time, a robust solution to span-tokenizing should be investigated, and relevant code submitted to the NLTK project for review in a Pull Request. Another approach is to completely exclude word-tokenization from the interpolation process, using a sliding character-window over the text to find and label named entities. This option was excluded because of its poor performance and time constraints.

Finally, another library could be used to provide span\_tokenize functionality. This is discussed in section 12.3.

## Toy dataset model – tensor formation

Sentences, segmented by the NLTK Punkt sentence segmenter with some customisation, are used as the default unit for each tensor. This has the advantage that each tensor (if correctly ‘chunked’ into a sentence) is guaranteed to be a single, cohesive utterance, as opposed to tensors represented by a fixed number of characters. At the other extreme, it is much more tractable than using a whole debate as one tensor (the longest debate in the collection was 1.13m characters long).

Sentences in human language greatly vary in length, yet the BLSTM requires tensors of uniform length. A max sentence length of 200 was chosen for the model. Any sentences longer than this are truncated, even if truncation occurs mid-way through a word. Any sentences shorter than this are left-padded with null characters, using Keras’ pad\_sequences helper method, which also takes care of converting the python lists to Numpy arrays. The value of 200 was chosen by taking the median sentence length of the ‘ToyV1’ dataset, which was 111, and then rounding up. Of course, this still means that a majority of sentences will have padding – the decision to use variable-length sentences to form tensors necessitates choosing between sparseness in the tensors, and frequent truncation of sentences. I judged sparseness in the tensors to be the best way to preserve information for the model to learn from.

## Hansard presentation issues

When the debates were downloaded from the TheyWorkForYou API, all speaker information was retained in XML metadata tags. So as not to label or train on these tags, they were removed from the raw data. The difference is illustrated by comparison of Figure 4 and Figure 5, both taken from Hansard debate “Flying Bomb Attacks (Meetings with Ministers” from the 7th July 1944.



Figure 4 Hansard debate, XML format



Figure 5 Hansard debate, processed TXT format

Removing all XML tags presents a metadata problem, as all indication of the speaker, originally in the “speech” XML tag, has now been removed. A better processing pipeline would download the debates in two formats, stripping the tags to train the model, but then re-instating them to display the metadata to the end-user. The tags could be stripped again whenever the model is used to predict Named Entities, so that only the texts of the debates themselves are annotated with Named Entity prediction.

# Evaluation and Testing

## Model evaluation

The Keras model was evaluated using the Keras ‘evaluate’ method. Of course, the labelled data used to fit the model was both limited by the contents of its data sources, and by the non-human manner of the labelling – the labels were ‘interpolated’ using the algorithm described in detail in section 7.7.

The various different datasets, their sizes (number of sentences, which equals the length of the first dimension of the X and Y tensors), and the number of epochs of training used are listed in Table 5. Baseline accuracy is shown in Table 6 and described in section 9.1.1. Evaluation of all the models trained is shown in Table 7 and analysed in the rest of this section below.

Table 5 Model datasets used

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model name** | **Training dataset size** | **Dev dataset size** | **Test dataset size** | **Epochs**[[21]](#footnote-21) | **Notes** |
| Mini | 4000 | 4000 | 4000 | 2 | Takes first 4000 sentences from ToyV1. |
| ToyV1 | 2,323,451 | 1,233,720 | 1,157,309 | 7 | All validations on Dev and Test datasets returned NaN. |
| ToyV2 | 500,000 | 6000 | 6000 | 6 | Uses same data as ToyV1, but with arrays clipped at the limits shown. |
| ToyV3 | 1,000,000 | 60,000 | 60,000 | 7 | Uses same data as ToyV1, but with arrays clipped at the limits shown. |
| Full | 91,233,214 | 45,205,262 | 46,136,596 | 0 | This dataset was not trained on – see section 9.1.5. |

Table 6 Baseline accuracy

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Dataset used** | **Dataset size** | **Baseline prediction** | **Categorical accuracy** | **Non-Null Label Accuracy** |
| Test | Toyv2 | Always null | 0.9035 | 0.0 |
| Test | Toyv2 | Always “location” | 0.05775 | 0.05775 |

Table 7 Evaluations of trained models

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Trained model under evaluation** | **Dataset used for evaluation** | **Dataset size** | **Categorical accuracy** | **Non-Null Label Accuracy** | **Loss** |
| Mini | Test | Mini | 0.8792 | 0.0 | 0.3947 |
| ToyV1 | Test | Toyv1 | 0.9726 | NaN | 0.0885 |
| ToyV1 | Test | Mini[[22]](#footnote-22) | 0.9570 | 0.5270 | 0.1386 |
| ToyV2 | Test | ToyV2 | 0.9538 | 0.5532 | 0.1564 |
| ToyV3 | Test | ToyV3 | 0.9594 | 0.5454 | 0.1364 |

### Baseline results

When evaluating any model it makes sense to first calculate the score of the common-sense baseline. In a Named Entity task where Null is the most frequent label, the obvious baseline is to declare every word as Null, i.e. not a Named Entity. Of course, in practice such a classifier is useless. However, in terms of pure accuracy, it gains quite high results.

Fortunately, 0xnurl’s Keras implementation of the model defines a custom metric for non-null-label-accuracy, the accuracy of NE labels excluding the null label. For this metric, the only baseline is to pick one NE label and apply it all the time, say, ‘location’. In this case, the non-null label accuracy will match the categorical accuracy, as in each case we are picking a non-null label. Of course, if we guess the ‘null’ label each time, then the non-null label accuracy will be zero, as shown in Table 6.

### The ‘mini’ dataset

In order to test out a complete run of the Keras model and verify saving of its state and tracking of its loss scores across epochs, the ‘mini’ dataset was generated. The mini dataset is derived from the ‘toy’ dataset (for which see section 9.1.3), but in the first dimension of the X and Y tensors, only the first 4000 samples are taken for each dataset. So only 4000 sentences are used for each of train, dev and test. The figures above show the predictably appalling behaviour of this dataset.

Note that the ‘mini’ model was only run for two epochs. We can see that accuracy rose and loss decreased. However, the non-null label accuracy also decreased – it appears that in the early epochs of the model, the most efficient way to minimise loss is to label every character as 0, the null label. This observation was also borne out in the ‘toy’ dataset, where non-null label accuracy fell for the first 100k samples or so, before starting to rise.

Attempts to ‘predict’ using the mini dataset also demonstrated that this model has a marked preference for the NULL label. Indeed, all evaluation done using the mini model returned NULL labels for every character; see Table 7, where the mini model’s non-null label accuracy is 0.0.



Figure 6 Mini dataset accuracy



Figure 7 Mini dataset loss



Figure 8 Mini dataset non-null label accuracy

### The ‘toy’ dataset, version 1

The toy dataset was constructed with 8 buckets of the 320 in the dataset, roughly 1,600 debates in total. Half of these were used for training, and a quarter each for test and dev. After two epochs, the model stopped training due to a bug discussed below. After this, the model was trained for a further five epochs.



Figure 9 Toy dataset NaN validation accuracy



Figure 10 Toy dataset NaN validation loss



Figure 11 Toy Dataset NaN Validation Non-Null Label Accuracy

The graphs above do not show validation accuracy. This is because of a bug encountered with the Toy dataset, where after each epoch, evaluation done against the ‘dev’ dataset leads to a result of NaN (Not a Number). Because of this bug, it was not possible to track the model’s performance on a dataset other than ‘train’ during the fitting process, and hence impossible to detect and avoid overfitting. This was also the reason why the model originally stopped training after two epochs – the Keras EarlyStopping callback was called after two epochs with no improvement in the validation score. Figure 9, Figure 10 and Figure 11 show no validation scores because of these NaN return values. In order to train for more epochs, the EarlyStopping configuration had to be removed from the model.

While attempting to fix the NaN validation problem, the NumPy arrays used in all datasets were searched for NaN values, infinity values and other non-numeric values, without success. It was noted that this problem was not encountered in the mini dataset, which indicated that it was related to the batch-size used in the toy dataset. Its evaluation metrics are shown in Table 7 (as the ToyV1 non-null label-accuracy score was NaN, we also evaluated the model against the ‘mini’ dataset to provide some indicative score). As non-null label accuracy is so much lower than the categorical accuracy, it is no surprise that the model also had a marked preference for returning the Null label.

### The ‘toy’ dataset, versions 2 and 3

Comparing the Mini and ToyV1 datasets, it is clear that the Mini dataset at least returned validation data, a signal that could be used to detect overfitting during the training epochs. This understanding led to ToyV2, which caps the training data to 500,000 sentences, and test and dev to 6,000 sentences each. The values were picked empirically based on the successful validation feedback from the Mini dataset.

Restricting the size of the ‘dev’ dataset solved the problem of NaN scores during training, as the graphs below show. Thanks to the end-of-epoch validation data, it was possible to identify that the ToyV2 dataset started to overfit after the 5th epoch (see Figure 12, Figure 13 and Figure 14).



Figure 12 ToyV2 dataset accuracy



Figure 13 ToyV2 dataset loss



Figure 14 ToyV2 dataset non-null label accuracy

Note from Table 7 that the non-null label accuracy for ToyV2 is higher than that achieved from ToyV1 against the ‘mini’ test-set, which is in turn higher than the ‘mini’ dataset’s non-null label accuracy of 0.0.

Following these results, a ToyV3 dataset was trained, using more training data and test/dev sets of the same size, to see if this score could be further improved. As Table 7 shows, while it had smaller loss than ToyV2, its non-null label accuracy was in fact lower. Its graphs are given below for reference, in Figure 15, Figure 16 and Figure 17. On the assumption that non-null labels are those of most utility to the user, the ToyV2 model was used in the Simple-GUI described in section 10.



Figure 15 ToyV3 dataset accuracy



Figure 16 ToyV3 dataset loss



Figure 17 ToyV3 dataset non-null label accuracy

### The full dataset

To train the ToyV1 dataset for seven epochs on Birkbeck’s deep Machine Learning server took about 36 hours per epoch, or about eleven days. With 2,323,449 sentences of data trained every epoch, it is clear that the ToyV1 dataset actually represents a significant amount of data. As the ToyV1 dataset required 18GB just to load the tensors into memory, it is anticipated that the full dataset would require 720GB or thereabouts.

Clearly, to use all the interpolated Hansards into the model (divided into train, test and dev sets) requires using Keras’ fit-generator methods, to generate the tensors on the fly as they are needed, and a large amount of time. It is possible that the NaN validation problems described in 9.1.3 would recur, given the much larger dataset (182,575,072 sentences in total, see Table 5). Or, it is possible that, with a suitably chosen batch-size for the generator, the NaN validation problem could be avoided. It is not clear how long such a model could take to train – if the time taken scaled linearly from the ToyV1 dataset, which used 8 of the 320 buckets, it could take 1440 hours, or 60 days, per epoch. It is hoped that the time taken would not scale linearly, given that the batch size, and the required stochastic gradient descent calculations per batch, would be much smaller.

While the full dataset was not used for training as part of this project, the code to do so was added to model\_integration.py, train.py and tasks.py. The main addition is the nested loops required to generate the data in the batch-size requested. As Code Snippet 5 shows, the generators originally used to create the X and Y tensors are first created. Then, the X and Y generators are called together inside a pair of nested loops. The inner loop creates a tensor for each item in the current batch, while the outer loop handles the batch size and creation of a new list for each X and Y, for each batch. Once a given batch has been created, its X and Y tensors padded and converted to Numpy arrays, before being yielded out of the Python generator.



Code Snippet 5 Training on the Full dataset

Unfortunately, attempting to train the model using the full dataset resulted in out-of-memory errors from the underlying Tensorflow library. Completing training on the full dataset is a logical extension to this project and is discussed in section 12.5.

### Cross-validation

K-fold cross-validation was indicated as the preferred evaluation method for the machine learning model, in this project’s proposal.

Thanks to the hash-bucketing approach to datasets, the Hansard debates are already scrambled, so I could just use contiguous pieces of the NumPy arrays to generate segments. I used Scikit-Learn’s KFold class to generate ten folds. In each case, a fold was one tenth of the data, used for validation. The other nine tenths were used for training. Code Snippet 5 shows how the Scikit-Learn’s KFold class is used to provide indices for the dataset splits, a new Keras model is instantiated, and then methods manual\_fit and manual\_evaluate are called. These methods are added to the Keras model class using Python subclassing in my project’s Python package – this approach is required because 0xnurl’s Keras implementation provides its own implementation of the “fit” and “evaluate” methods, which are not designed for use in cross-validation. Code Snippet 6 shows an example of this approach.

Because of issues mentioned with using the full dataset explored in section 9.1.5, I used the same dataset originally used to train the ToyV2 model. As a fresh model was used for cross-validation evaluation, this was a fair evaluation on a manageable amount of data.



Code Snippet 6 k-fold cross validation



Code Snippet 7 Python subclassing to add model evaluate

**Cross-validation results when they’ve run – waiting on Birkbeck server**

Table 8 k-fold cross-validation results

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Dataset used** | **Dataset size** | **Number of folds (test = 1 fold)** | **Epochs per fold** | **Loss** | **Categorical accuracy** | **Non-Null Label Accuracy** |
| Cross-validation | ToyV2[[23]](#footnote-23) | 10 | 3 |  |  |  |

## Unit testing

For all the pre-processing code submitted for this project, the Pytest framework was used to run simple unit tests, validating that functions return expected outputs for given inputs. In this project, most of the main functions used did not return values directly to the caller, but wrote values out to disk, either as text files, as binary Python pickled data, or as Keras h5 database-files. Similarly, many functions expected their input to be a path to a file on disk, from which they would read either text or binary data to continue processing. This approach was taken to support the ‘pipeline’ concept outlined in section 7.1.

In order to validate that a function was writing out expected values to disk, Pyfakefs[[24]](#footnote-24) was used to create a fake filesystem, existing solely in memory, in the context of the unit test. Then the function was run, and the dummy file on the fake file-system was then examined to ensure it had the expected context. Given Pyfakefs’ native integration with Pytest, the written tests do not have to bear much complexity for this setup.

To illustrate this approach, Code Snippet 7 shows a regular unit test written in Pytest. A value is passed into the ‘onehot’ function, and its output is compared against an expected value in a simple equality assertion. Code Snippet 8, by contrast, shows a Pytest unit test with Pyfakefs. The argument passed into test\_interpolate\_one, “fs”, is the fake filesystem. The “fs.create\_file” call is used to create a dummy file in the fake filesystem to feed into the function under test, and an empty dummy file to accept the function’s written output. Once the function (interpolate\_one in this test) is called, the resulting output file is read from the dummy filesystem, and its contents compared to their expected result.



Code Snippet 8 a regular Pytest unit test



Code Snippet 9 A Pyfakefs Pytest unit test

Unit test coverage for this project is not extensive – most of the effort of evaluation was irected towards evaluating the machine learning model. However, the unit testing approach demonstrated here could be applied more thoroughly to the whole codebase.

## Overall evaluation

The user experience was manually validated using Simple-GUI (see section 10). It was noted that there are many errors in NE recognition, as the non-null label accuracy of just over 0.5 would suggest. However, given the low quality of the input labelled data, the NEs that are successfully recognised are impressive and useful.

One major problem found with the GUI experience was that it was easy to re-submit a paragraph for Named Entity recognition, which had been processed already. This would result in annotation tags such as ‘<loc>…</loc>’ being themselves submitted to the model to be predicted. To avoid this problem, the GUI was corrected so that on click, a paragraph had a ‘bounce’ effect so it was obvious to the user that it is being processed; paragraphs that had already been predicted were coloured in blue, so even if it had all null labels, a user could still see it had been processed; and the click handler was removed from processed paragraphs in the JQuery callback, so they could not be re-submitted for NE prediction. An example is given in Figure 15 of the view pre-NE annotation, and in Figure 16 of post-NE annotation.

****

Figure 18 Example Hansard text before NE annotations - SimpleGUI

****

Figure 19 Example Hansard text after NE annotations - SimpleGUI

# Graphical User Interface

## Simple-GUI

This project includes its own Graphical User Interface (GUI), which was originally in order to mitigate the risk of Birkbeck’s Samtla system not being available for integration. However, as section 10.2 shows, Simple-GUI’s implementation was complete enough to be integrated with Samtla wholesale, requiring no changes to the client-side Javascript code – hence it proved a useful part of the integration work. Preparing a simple GUI also had the advantage that the required Javascript could be developed and tested on a single-purpose system with no other dependencies.

The GUI created for this purpose was named Simple-GUI. It contains the following basic functionality: allow a user to see a list of all possible Hansard debates on a given date; allow a user to select a Hansard debate from a given date to display all of its text; The Hansard debate displayed should in some way annotate all Named Entities predicted by the Named Entity model which should be run on the server. Figure 20 shows the design of the HTTP endpoints to the server, and the role of client-side Javascript.



Figure 20 Simple-GUI basic design

Python Flask was chosen as the web-server to build a very basic site, as it supports Python 3 and does not bring the unneeded overhead of a persistence framework, unlike the more sophisticated Django web server used for Samtla. Flask’s only dependencies are Jinja2,[[25]](#footnote-25) which we use to template the HTML files for the site, and Werkzeug[[26]](#footnote-26) for managing the Web Server Gateway Interface (WSGI) HTTP routing.

In the first draft of the GUI, when the user clicked on an entire Hansard debate, the whole debate was first fed into the predict\_str() function of the model, to generate Named Entity labels for the whole text, before displaying the whole annotated text to the user. However, given that running a whole debate text through “predict\_str” takes about 8 minutes even on the Birkbeck machine learning server Venus, it is clear that this approach could never constitute an acceptable user experience. Of course, the values could all be pre-computed, but this seems at odds with the goal of using a machine-learning model which can be dynamically re-trained and re-run on the same source text.

A second approach was chosen, as shown in Figure 20, whereby the text is loaded into the browser unannotated, Text is split on newlines, with each newline-separated chunk residing in its own HTML paragraph element, within <p> tags. Whenever a paragraph is clicked on, a JQuery event handler sends an AJAX POST request containing the whole text of the paragraph, to a special “/predict/” endpoint on the web-server. The web-server passes this whole string to the Keras model (read in from disk when Flask starts, but thereafter held in memory and available to all Flask HTTP request threads) to predict NEs on it. The model returns predictions as a zipped Python list of tuples, such as “[('H', 'LOC'), ('u', 'LOC'), ('l', 'LOC'), ('l', 'LOC')]”. To make this user-friendly, we convert this zipped list into text surrounded by markup tags like “<loc>Hull</loc>”. The function that does this simply keeps track of the current and previous character label and adds NE markers where required.

The annotated text is then returned to the browser, calling a JQuery callback which replaces the paragraph contents in the page, with the annotated content. The effect is that whenever a user clicks on a paragraph, its NEs are annotated. This second approach was empirically found to have a much better user experience. Finally, the JQueryUI[[27]](#footnote-27) “bounce” effect was added to the paragraph text when it was clicked, to indicate to the user that the click was recognised, and Domain Object Model (DOM) element classes with Cascading Style Sheets (CSS) were used to change the text to a blue colour once NE prediction was done, and to remove the click event handler so the user could not predict NEs a second time on the same text.

As this GUI is very much a stub designed to enable development and demos of the project, no thought was given to non-functional requirements like performance or security. Performance in particular would be greatly improved with a relational database to index Hansard debates for a particular date, and cached results from the model predictions. Presentation of the UI would be enhanced by using some sort of animated, floating annotations on NEs in the browser, rather than just changing the HTML to add NE tags.

## Samtla Integration

The full processed Hansard dataset was given to Dr Martyn Harris on Thursday 23rd August. After some discovery, it was concluded that the dataset was too large for Samtla’s current storage array, so a subset of the Hansard data was prepared for loading. After discussion of the different integration options, it was agreed to run the Simple-GUI’s Flask server on the Samtla hardware, and to embed the Simple-GUI’s client-side Javascript in its current form. This would provide easy integration with Samtla without the need for any invasive changes to its other text corpora. For the Hansard datasets, Samtla’s native, gazetteer-based NER functionality would simply be suspended, so that this project’s code could be used instead.

As of this writing, on 30th August 2018, the loading of Hansard data into Samtla is in progress. A meeting is scheduled with Dr Martyn Harris on Thursday 6th September to discuss hosting of the Javascript and Flask server code, which is ready to be used.

# Planning

In retrospect, the plan submitted in the Proposal suffered from a critical lack of detail. The complete plan, with its original timelines, is given in Table 11. The original ‘risks and mitigations’ column has been removed as it is not relevant here.

Table 9 Original work plan given in proposal

|  |  |  |
| --- | --- | --- |
| **ID** | **Work package** | **Target date** |
| 1 | Sourcing and aggregation of data sources for Named Entities of companies, people and places.  Preparation of Hansard documents with NEs interpolated into the document using the NE lists aggregated above. | End April 2018 |
| 2 | Perform manual validation of above Hansard documents to complete semi-automatic labelled data.  Construction of NER learning model BLSTM; integration with and modification of https://github.com/0xnurl/keras\_character\_based\_ner implementation. | End May 2018 |
| 3 | Familiarisation with Samtla SLM. Conversion of Hansard into correct format and loading of Hansard data into Samtla. | Mid June 2018 |
| 4 | Familiarisation with Samtla back-end (Python Django) and integration of NER processing with departmental server. | End June 2018 |
| 5 | Familiarisation with Samtla front-end (Javascript jQuery) and integration of NER visualization. | End July 2018 |
| 6 | Evaluation of Samtla Char-NER system using k-fold cross-validation techniques. | End August 2018 |
| 7 | Finalise write-up into report. | Before 17th Sept 2018 |

The actual time spent on each work package up to and including 23rd August is shown in

Figure 21.

Figure 21 Actual time spent on each work package of project

There are some immediately obvious conclusions from the comparison. Firstly, the dots (which show actual days worked on the project) make it clear that work only back in earnest once the exam period was over. Planning to do a significant amount of work during exams was foolhardy.

Secondly, the plan had a lack of detail. In particular, Work Packages 1 and 2, concerning the preprocessing/labelling of data, and training of the model, comprised a huge amount of work and should have been several work packages. Steps 3 and 4, integration with Samtla, barely required any work once the heavy lifting of NE prediction was done – a 12-line Javascript file was all that was needed to get prediction working, and Dr Martyn Harris was able to load the Hansard texts into Samtla himself using his own scripts.

Finally, having a separate work package for writing up the report was not wise, as it turned out that the report had to be written in parallel with the work in order for the detail to be remembered. The graph clearly shows that the work packages were not sequential either, as had originally been envisaged – the different parts of the project had to be worked on in parallel to ensure the data was compatible between each stage.

# Summary and Conclusions

## Pre-processing is hard

The vast majority of the work in this project involved taking the Hansard data, pre-processing it without introducing data corruption, and then labelling it for the model. The project demonstrates that producing data in a usable format and with a reasonable distribution takes a lot of time from machine learning projects.

The work of the TheyWorkForYou API makes this much simpler for the Hansard dataset, without which the raw PDFs from the UK government website would have also needed parsing.

## Automated labelling is hard

In general terms, it is clear that the labelling done on the Hansard data is not ideal. All the graphs in section 9.1 show a marked divergence between training loss and validation loss, right from the first epoch – hence the model is struggling to generalise what it learns on the training data, to apply successfully to the dev or test data.

One reason may be that the interpolation algorithm detailed in section 7.7 has a major flaw: it is sequential. The algorithm first tries to identify a given set of words as a location, then an organization, then a person. There is no logic to ‘abandon’ one interpolation for a more likely one – indeed, there is no model to define, at interpolation-time, what ‘more likely’ even means.

Hence, due to the NE overlap logic described in section 8.2, there is a “first-found” bias where, once the interpolation algorithm has started to identify a Named Entity, it can never change its mind, regardless of what further evidence becomes available as the sliding window moves through the text. The interpolation algorithm could attempt to identify Named Entities in the text probabilistically – such as considering the prior probability of that particular word-phrase being a particular NE type, as derived from some other corpus. As the window moved over the text, the algorithm would see more evidence, which it could consider as it performed its labelling.

## Sentence tokenization is hard

The NLTK Punkt sentence-tokenizer has to be taught specific abbreviations in order to achieve workable sentence division. In a machine learning project motivated by not needing to identify features, this seems unprincipled. The NLTK Punkt tokenizer’s span\_tokenize bugs, described in section 8.3, required the raw Hansard data to be altered. Spacy, another Python library, offers its own sentence segmentation solution, but instead of providing character indices of the starts and ends of sentences, it returns ‘Span’ objects which comprise tokens that enable the client to construct the original string from the response. All the Hansard chunking code would need to be rewritten in order for this library to be used.

## Neural networks are slow and opaque

As detailed in 9.1.5, training the full dataset could take 60 days, assuming a linear increase in training time. Also, using hundreds of thousands of tensors to validate the data produced NaN scores which rendered the training useless. Configuration of hyper-parameters is still something of a dark art; one approach is to simply run the model many thousands of times with different hyperparameters, and choose the best results. Given more time, this project would benefit from testing some different combinations of hyperparameters concerning sentence length and batch size, as well as dropout rate, learning rate and the Embedding size used in the first layer of 0xnurl’s model.[[28]](#footnote-28)

Gaining any further information from a Keras model during training, except for the metrics defined at model-compile time which are computed after each epoch, requires writing custom call-backs. A degree of pre-work must be done to identify information likely to be useful, and then Keras callbacks must be written, in order to expose this information during model training. For example, no precision or recall data is available from 0xnurl’s model, as this was removed from the Keras standard library.

## Future work

This project achieved its stated aims of performing Named Entity recognition using a trained neural network, but leaves plenty of room for further improvement. Adding Viterbi post-processing to the model, as detailed in (Kuru, Arkan Can and Deniz, 2016), could improve the non-null label accuracy scores. Given time, training on a larger dataset could also improve the score – the full dataset of Hansards have all been interpolated, so future work could involve using Keras’ “fit\_generator()” method to successfully train on this huge dataset. This would require overcoming the out-of-memory errors encountered in this project.

However, ultimately a ceiling will be reached, after which the labelling approach should be revisited to overcome the limitations summarised in 12.2. A manual labelling effort could greatly improve the quality of data to train from.

The genre itself may also be problematic; as Dr Martyn Harris notes, the Hansard debates cover a wide variety of topics and a wide space of time. Perhaps their content is simply too broad to be considered as a single genre. As such, better results could hypothetically be achieved by filtering Hansard debates down to those on similar topics, such as transportation, or international affairs.

Finally, the Keras model should have precision and recall metrics added, so that F1 can be calculated. These metrics were removed from Keras in version 2 of its standard library but could be added to the model as custom metrics. In general, the user interface could be improved by adding the percentage certainty of the label given by the model, or indeed the percentage certainty for each label given by the model, per character, although this would be a challenge to render clearly to the user.

# References

Klein, D. *et al.* (2003) ‘Named entity recognition with character-level models’, *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003 -*, 4, pp. 180–183. doi: 10.3115/1119176.1119204.

Kuru, O., Arkan Can, O. and Deniz, Y. (2016) ‘CharNER : Character-Level Named Entity Recognition’, *Coling*, pp. 911–921.

While not directly quoted, the following books were consulted for the preparation of the Simple-GUI and the setup of the Keras model:

Chollet, F. (2017) ‘Deep Learning with Python’, *Manning Publications*

Duckett, J., Ruppert, G. and Moore, J. (2014) ‘JavaScript and JQuery’, *John Wiley & Sons*

Grinberg, M. (2014) ‘Flask Web Development’, *O’Reilly*.

# User Manual

## Data pipeline manual

The data pipeline is run by using the invoke tasks listed in section 15, in the correct order. Firstly, the *ne-data* tasks all need to be run – these will download all Named Entities from DBPedia into their correct folders. Next, the *hansard-download-all* and *hansard-process-all tasks* should be run, which will distribute tasks to a Python thread pool to download and process the Hansard debates into text format. Then *hansard-chunk-all, hansard-interpolate-all* and *hansard-numerify-all* should be run to finalise processing of the data.

Once the data is present on the machine, *char-ner-create-x-toy* and *char-ner-create-y-toy* should be invoked to create the tensors for the model and save them on disk. *The model-minify-toy* task can optionally be run to create the mini dataset. Now the model can be trained with *model-train-toy* (or *model-train-mini*), which may take several days. Once finished, graphs of the metrics during the epochs of training can be generated in png format using the *model-history-toy* (and *model-history-mini*) tasks. Now the trained model can be formally evaluated with the *eval-model-manual task* and invoked from Simple-GUI by starting the Flask webserver using the *flask-start-server* task. The *eval\_k\_fold\_cross* task will train the model ten times for k-fold cross-validation, displaying results with mean and standard deviation at the end of the 10th training run.

## Simple-GUI Manual

Simple-GUI is deliberately straightforward. The index page presents a list of dates to the user (Figure 22). Selecting a date takes the user to the date page, with a list of all Hansard debates available for that date. Each Hansard debate is given a numerical ID (Figure 23). Clicking on a debate leads to the debate page (Figure 24), which shows the text of the debate. Clicking on a paragraph will cause the paragraph to ‘bounce’, an animation to show the user that its Named Entities are being fetched. Then the NEs will be annotated with simple tags, and the text colour will turn blue to indicate that the NEs for this paragraph are annotated already – see the figure for an example.



Figure 22 Simple-GUI index page



Figure 23 Simple-GUI date page



Figure 24 Simple-GUI debate page with an annotated paragraph.

# Appendix A: List of Invoke tasks

Table 10 comprises a list of Invoke tasks which can be started within the project, along with a description of the work that they do. The tasks are a combination of environment setup, running automated tests, and the ‘business logic’ of the code – the downloading and processing of Named Entities and Hansard debates. Having one clean, uniform interface for all these tasks greatly simplified the workflow when parts of the pipeline had to be re-run, without the overhead of creating a GUI or integrating with one. Also, having pre-requisites in code avoids the need for repetition. For example, the unit tests are not run unless the virtual environment is set up, and the static type-checker has run already. These tasks are both listed as pre-requisites of the ‘test’ task in Invoke’s “tasks.py” file.

Table 10 List of Invoke tasks used to drive the pipeline

|  |  |
| --- | --- |
| **Task** | **Description** |
| char-ner-create-x-toy | Create an X tensor of Numpy arrays from numerified Hansard data |
| char-ner-create-y-toy | Create a Y tensor of Numpy arrays from onehot vectors from interpolated (labelled) Hansard debates |
| char-ner-display-median-sentence-length | Get the median sentence length of a given dataset |
| char-ner-display-pickled-alphabet | Display the CharBasedNERAlphabet object pickled to disk by char-ner-pickle-alphabet |
| char-ner-pickle-alphabet | Use a small subset of the Hansard debates data to union together all characters used and create a CharBasedNERAlphabet object with a number-to-character mapping |
| char-ner-rehash-datasets | Rehash all the debate data into a different number of buckets – discussed in section 7.9. |
| compile | Run py\_compile on all python files to find compile-time static code problems |
| enable-venv | Enable the Virtual Environment (i.e. a segregated location for pip installs) for this project. A required prerequisite for several other tasks. |
| eval-baseline | Calculate and evaluation baseline for the project based on a provided y tensor, assuming the baseline guesses a given label for each character. |
| eval-k-fold-cross | Calculate the k-fold cross-validation score for the Toy dataset, given 10 folds in the data. |
| eval-model-manual | Take a trained model from disk and perform Keras evaluate() function on a tensor of test data, producing loss, accuracy and non-null label accuracy. |
| flask-start-server | Start the Python Flask web-server used to demonstrate Simple-GUI |
| hansard-chunk-all | Use the chunker on all Hansard debates in the collection – described in section 7.6 |
| hansard-chunk-one | Use the chunked on one Hansard debate, to allow manual validation |
| hansard-display-chunked | Display one Hansard debate, tagged with all its sentence-boundaries, to validate the sentence chunking algorithm. |
| hansard-display-interpolated-file | Display one Hansard debate, with every character tagged by the interpolator as 0 (null), 1 (location), 2 (organization) or 3 (person) |
| hansard-download-all | Concurrently download all Hansard debates from both houses, from a given starting date. |
| hansard-fix-uninterpolated | Find all Hansard debates which did not correctly interpolate due to an NLTK bug with span\_tokenize, and re-interpolate |
| hansard-interpolate-all | Interpolate (label) all Hansard debates using Named Entity data |
| hansard-interpolate-one | Interpolate (label) one Hansard debate for manual validation |
| hansard-numerify-one-to-file | Numerify one Hansard debate – i.e. replace each of its characters with the equivalent integer for this CharBasedNERAlphabet, and store in a file for manual validation |
| hansard-process-all | Do pre-processing steps on all Hansards – discussed in more detail in section 7.5. |
| hansard-process-one | Do pre-processing steps on one Hansard for manual validation. |
| hansard-write-total-number-of-sentences-to-file | Count how many sentences there are in each dataset and write out to file for easy retrieval. This information is used to estimate to the user how long the tensor creation will take. |
| model-history-mini | After training the mini model, produce graphs of its loss, accuracy and non-null label accuracy on the train and dev datasets as recorded after each epoch. |
| model-history-toy | As above, but for the model trained on the toy dataset. |
| model-mini-predict-file | Use the mini model to predict all Named Entities in a given text file |
| model-minify-toy | Take the toy dataset and truncate the 1st dimension of all the X and Y tensors to the first 4000 samples, to create a mini dataset |
| model-retrain-toy | Read in saved Keras toy model off disk, and then perform further epochs of training on it. |
| model-toy-predict-file | Read toy model off disk and use it to predict all Named entities in a given file |
| model-toy-predict-string | Read toy model off disk and use it to convert the given string to tensor, and predict Named Entities in that tensor |
| model-train-mini | Run model.fit() on the Keras BLSTM model, with a batch of the first 4000 samples from the toy dataset. This is to test the end-to-end process of saving and retrieving the model. |
| model-train-full | Run model.fit\_generator() on the Keras BLSTM model, which carves test, dev and train sets from all the available data. Memory issues prevented this task from running to completion during this project. |
| model-train-toy | Run model.fit() on the Keras BLSTM model, with a toy dataset of 1 320th of the Hansard debates. This is to get an initial indication of the model’s learning capability. |
| ne-data-companies-download-process | Both download and process companies data from DBPedia and other sources |
| ne-data-companies-process | Only do post-processing, data cleansing tasks on companies data, to assist with iteratively improving the cleaning algorithm |
| ne-data-people-download-process | Both download and process people data from DBPedia and other sources |
| ne-data-people-process | Only do post-processing, data cleansing tasks on people data, to assist with iteratively improving the cleaning algorithm |
| ne-data-places-download-process | Both download and process people data from DBPedia and other sources |
| ne-data-places-process | Only do post-processing, data cleansing tasks on places data, to assist with iteratively improving the cleaning algorithm |
| print-debate-titles | Both download and process people data from DBPedia and other sources |
| python-type-check | Run mypy, Python’s static type checker, over all files I wrote in the project which contain type annotations |
| test | First run python-type-check, then run pytest unit tests on the project |

Each task in invoke calls out to something else; most tasks invoke a library function from elsewhere in the code base, while some invoke shell commands, for example to de-duplicate and sort the Named Entity lists. In this case, it is faster for the shell to call a GNU C binary and use the ‘sort’ and ‘uniq’ commands, than to use similar functionality in Python.

# Appendix B: What’s My Work

As this project embeds Nur Lan’s Keras model, Table 11 lists who authored each code source file in the project, and a description of its purpose. My contribution is approximately 2,500 lines of code.

Table 11 List of source code with author

|  |  |  |
| --- | --- | --- |
| **File Path** | **Author** | **Description** |
| ./tasks.py | Matt Ralph | Invoke tasks manager |
| ./ne\_data\_gathering/places.py | Matt Ralph | Gather NE location data |
| ./ne\_data\_gathering/util.py | Matt Ralph | NE common functions |
| ./ne\_data\_gathering/\_\_init\_\_.py | Matt Ralph | Empty file for Python Package definition |
| ./ne\_data\_gathering/people.py | Matt Ralph | Gather NE people data |
| ./ne\_data\_gathering/companies.py | Matt Ralph | Gather NE organization data |
| ./test/test\_companies.py | Matt Ralph | Unit tests for companies |
| ./test/\_\_init\_\_.py | Matt Ralph | Empty file for Python Package definition |
| ./test/test\_util.py | Matt Ralph | Unit test NE processing functions |
| ./test/test\_model\_integration.py | Matt Ralph | Unit test for onehot function |
| ./test/test\_simple\_gui\_util.py | Matt Ralph | Unit test for prediction formatting |
| ./test/test\_matt.py | Matt Ralph | Unit test for file management |
| ./test/test\_interpolate.py | Matt Ralph | Unit tests for interpolation algorithms |
| ./hansard\_gathering/filesystem.py | Matt Ralph | Utilities to allow Simple-GUI to serve Hansards from disk |
| ./hansard\_gathering/\_\_init\_\_.py | Matt Ralph | Empty file for Python Package definition |
| ./hansard\_gathering/chunk.py | Matt Ralph | Code to perform sentence segmentation |
| ./hansard\_gathering/numerify.py | Matt Ralph | Code to create x tensor |
| ./hansard\_gathering/preprocessing.py | Matt Ralph | Preprocessing of raw Hansard data |
| ./hansard\_gathering/interpolate.py | Matt Ralph | Interpolation algorithm for Y tensor |
| ./hansard\_gathering/driver.py | Matt Ralph | kick off Hansard downloads in threadpool |
| ./config\_util/config\_parser.py | Matt Ralph | Parse config file for TheyWorkForYou API |
| ./config\_util/\_\_init\_\_.py | Matt Ralph | Empty file for Python Package definition |
| ./keras\_character\_based\_ner/\_\_init\_\_.py | Matt Ralph | Empty file for Python Package definition. Added to allow my code to import Nur Lan's functions |
| ./keras\_character\_based\_ner/src/config.py | Nur Lan (0xnurl) | Nur Lan's Keras model implementation - embedded in this project to allow imports. |
| ./keras\_character\_based\_ner/src/\_\_init\_\_.py | Nur Lan (0xnurl) | Nur Lan's Keras model implementation - embedded in this project to allow imports. |
| ./keras\_character\_based\_ner/src/matt/save.py | Matt Ralph | Subclass Keras model to add 'save' method so we can save model weights progress |
| ./keras\_character\_based\_ner/src/matt/model\_integration.py | Matt Ralph | Main point of integration with Nur Lan's model - provide X, Y tensors and alphabet for model |
| ./keras\_character\_based\_ner/src/matt/dataset\_hashing.py | Matt Ralph | Logic to hash Hansards into buckets so all datasets contain distributed data |
| ./keras\_character\_based\_ner/src/matt/alphabet\_management.py | Matt Ralph | Create, store and load alphabet objects needed by the model |
| ./keras\_character\_based\_ner/src/matt/persist.py | Matt Ralph | Subclass Nur Lan's Keras model to allow easy saving and loading from disk to train or predict |
| ./keras\_character\_based\_ner/src/matt/predict.py | Matt Ralph | Helper functions to make it simple to predict NEs in strings from trained models |
| ./keras\_character\_based\_ner/src/matt/\_\_init\_\_.py | Matt Ralph | Empty file for Python Package definition |
| ./keras\_character\_based\_ner/src/matt/train.py | Matt Ralph | Override Nur Lan's own train.py so I can control the epochs and config used for training |
| ./keras\_character\_based\_ner/src/matt/eval.py | Matt Ralph | Code for baseline evaluation of code, k-fold cross validation, and model evaluation on test set |
| ./keras\_character\_based\_ner/src/matt/file\_management.py | Matt Ralph | Write own functions for pickling to address pickle bug, manage Hansard chunk files |
| ./keras\_character\_based\_ner/src/matt/minify\_dataset.py | Matt Ralph | Functions to create 'mini' dataset |
| ./keras\_character\_based\_ner/src/matt/history.py | Matt Ralph | Functions to create png graphs to show metrics after model has trained |
| ./keras\_character\_based\_ner/src/model.py | Nur Lan (0xnurl) | Nur Lan's Keras model implementation - embedded in this project to allow imports. I made some changes, indicated by comments marked ‘MIR’, to add functionality where this could not be done with subclassing. |
| ./keras\_character\_based\_ner/src/dataset.py | Nur Lan (0xnurl) | Nur Lan's Keras model implementation - embedded in this project to allow imports. |
| ./keras\_character\_based\_ner/src/train.py | Nur Lan (0xnurl) | Nur Lan's Keras model implementation - embedded in this project to allow imports. |
| ./keras\_character\_based\_ner/src/alphabet.py | Nur Lan (0xnurl) | Nur Lan's Keras model implementation - embedded in this project to allow imports. |
| ./simple\_gui/util.py | Matt Ralph | Simple-GUI helper function to convert model predictions to rendered text |
| ./simple\_gui/static/char-ner.js | Matt Ralph | Client-side javascript embedded to allow client to request Named Entity prediction from browser |
| ./simple\_gui/static/jquery-ui.min.js | Open Source | Copy of JQuery-UI library so this project can use its 'effect' function |
| ./simple\_gui/static/jquery.min.js | Open Source | Copy of JQuery library so this project can use its 'post' function for AJAX calls |
| ./simple\_gui/simple\_gui.py | Matt Ralph | Flask routes for Simple-GUI web server, and logic to initialize the ML model in the web server |

# Appendix C: Code

All the code contributed for this project is listed below. Empty Python \_\_init\_\_.py files (which are required in folders which are Python packages) are omitted as they do not contain code.

## tasks.py

from \_\_future\_\_ import print\_function

from hansard\_gathering import driver

from hansard\_gathering import preprocessing

from hansard\_gathering import chunk

from hansard\_gathering import interpolate

from hansard\_gathering import numerify

from ne\_data\_gathering import places

from ne\_data\_gathering import people

from ne\_data\_gathering import companies

from ne\_data\_gathering import util

from invoke import task, call

from keras\_character\_based\_ner.src.matt import alphabet\_management, file\_management, \

model\_integration, dataset\_hashing, train, minify\_dataset, history, predict, eval

from keras\_character\_based\_ner.src.config import Config

from simple\_gui import simple\_gui

import pickle

@task

def print\_debate\_titles(ctx, datestring):

[print(title) for title in driver.get\_hansard\_titles(datestring, "Debates", "commons")]

[print(title) for title in driver.get\_hansard\_titles(datestring, "Debates", "lords")]

@task

def hansard\_download\_all(ctx, year=1919, month=1, day=1):

driver.get\_all\_hansards(year, month, day)

@task

def hansard\_process\_one(ctx, filepath):

preprocessing.process\_hansard\_file(filepath)

@task

def hansard\_process\_all(ctx):

preprocessing.process\_all\_hansard\_files()

@task

def hansard\_chunk\_one(ctx, filepath):

tokenizer = chunk.nltk\_get\_tokenizer()

chunk.chunk\_hansard\_debate\_file\_nltk(filepath, tokenizer)

@task

def hansard\_chunk\_all(ctx, starting\_date):

# e.g. --starting-date 1919-01-01

chunk.chunk\_all\_hansard\_files(starting\_date)

@task

def hansard\_display\_chunked(ctx, filepath):

chunk.display\_chunked\_hansard(filepath)

@task

def hansard\_display\_interpolated\_file(ctx, filepath):

interpolate.display\_one\_file\_with\_interpolations(filepath)

@task

def hansard\_interpolate\_one(ctx, filepath):

ne = interpolate.NamedEntityData()

interpolate.interpolate\_one\_wrapper(filepath, ne, "processed")

@task

def hansard\_interpolate\_all(ctx, starting\_date):

interpolate.interpolate\_all\_hansard\_files(starting\_date)

@task

def hansard\_fix\_uninterpolated(ctx, starting\_date):

interpolate.fix\_uninterpolated\_hansards(starting\_date)

@task

def hansard\_numerify\_one\_to\_file(cdx, filepath):

with open("keras\_character\_based\_ner/src/alphabet.p", "rb") as f:

alph = pickle.load(f)

numerify.numerify\_one\_to\_file(filepath, alph, maxlen=Config.sentence\_max\_length)

@task

def enable\_venv(ctx):

ctx.run("echo enabling venv...")

ctx.run("source ./masters\_venv/bin/activate && pip install -r requirements.txt >/dev/null")

@task

def hansard\_write\_total\_number\_of\_sentences\_to\_file(ctx, dataset\_name):

file\_management.write\_total\_number\_of\_hansard\_sentences\_to\_file(dataset\_name)

@task

def compile(ctx):

ctx.run("find . -name '\*.py' | grep -v masters\_venv | xargs python -m py\_compile")

@task

def ne\_data\_companies\_download\_process(ctx):

companies.download\_and\_process("raw\_ne\_data", "/companies/dbpedia.txt")

ctx.run("cd ne\_data\_gathering/processed\_ne\_data/companies && cat \* | sort > ALL.txt")

@task

def ne\_data\_companies\_process(ctx):

util.dbpedia\_post\_processing(

"{}{}".format("raw\_ne\_data", "/companies/dbpedia.txt"), "processed\_ne\_data{}".format(

"/companies/dbpedia.txt"))

ctx.run("cd ne\_data\_gathering/processed\_ne\_data/companies && cat \* | sort > ALL.txt")

@task

def ne\_data\_people\_download\_process(ctx):

people.download\_and\_process("raw\_ne\_data", "/people/dbpedia.txt")

ctx.run("cd ne\_data\_gathering/processed\_ne\_data/people && cat \* | sort > ALL.txt")

@task

def ne\_data\_people\_process(ctx):

util.dbpedia\_post\_processing(

"{}{}".format("raw\_ne\_data", "/people/dbpedia.txt"), "processed\_ne\_data{}".format(

"/people/dbpedia.txt"))

ctx.run("cd ne\_data\_gathering/processed\_ne\_data/people && cat \* | sort > ALL.txt")

@task

def ne\_data\_places\_download\_process(ctx):

places.download\_and\_process("raw\_ne\_data", "/places/dbpedia.txt")

ctx.run("cd ne\_data\_gathering/processed\_ne\_data/places && cat \* | sort > ALL.txt")

@task

def ne\_data\_places\_process(ctx):

util.dbpedia\_post\_processing(

"{}{}".format("raw\_ne\_data", "/places/dbpedia.txt"), "processed\_ne\_data{}".format(

"/places/dbpedia.txt"))

ctx.run("cd ne\_data\_gathering/processed\_ne\_data/places && cat \* | sort > ALL.txt")

@task

def char\_ner\_pickle\_alphabet(ctx):

alphabet\_management.pickle\_alphabet()

@task

def char\_ner\_display\_pickled\_alphabet(ctx):

alphabet\_management.display\_pickled\_alphabet()

@task(post=[call(hansard\_write\_total\_number\_of\_sentences\_to\_file, "train"),

call(hansard\_write\_total\_number\_of\_sentences\_to\_file, "dev"),

call(hansard\_write\_total\_number\_of\_sentences\_to\_file, "test"),

call(hansard\_write\_total\_number\_of\_sentences\_to\_file, "alphabet-sample"),

call(hansard\_write\_total\_number\_of\_sentences\_to\_file, "ALL"),

])

def char\_ner\_rehash\_datasets(ctx):

dataset\_hashing.rehash\_datasets()

@task

def char\_ner\_create\_x\_toy(ctx, dataset\_name):

model\_integration.create\_x\_toy(Config.sentence\_max\_length, dataset\_name)

@task

def char\_ner\_create\_y\_toy(ctx, dataset\_name):

model\_integration.create\_y\_toy(Config.sentence\_max\_length, dataset\_name)

@task

def char\_ner\_display\_median\_sentence\_length(ctx, dataset\_name):

print(model\_integration.get\_median\_sentence\_length(dataset\_name))

@task(enable\_venv)

def python\_type\_check(ctx):

ctx.run("echo mypy: checking Python static types...")

ctx.run("mypy hansard\_gathering")

ctx.run("mypy ne\_data\_gathering")

ctx.run("mypy keras\_character\_based\_ner/src/matt")

ctx.run("mypy test")

@task(python\_type\_check)

def test(ctx):

ctx.run("echo pytest: running tests...")

ctx.run("pytest test")

@task

def model\_minify\_toy(ctx):

minify\_dataset.minify\_all()

@task

def model\_train\_toy(ctx, regenerate\_tensors="no"):

if regenerate\_tensors == "yes":

print("Regenerating tensors")

model\_integration.create\_x\_toy(Config.sentence\_max\_length, "train")

model\_integration.create\_x\_toy(Config.sentence\_max\_length, "test")

model\_integration.create\_x\_toy(Config.sentence\_max\_length, "dev")

model\_integration.create\_y\_toy(Config.sentence\_max\_length, "train")

model\_integration.create\_y\_toy(Config.sentence\_max\_length, "test")

model\_integration.create\_y\_toy(Config.sentence\_max\_length, "dev")

train.toy\_dataset\_fit()

@task

def model\_train\_full(ctx):

train.full\_dataset\_fit\_generator()

@task

def model\_retrain\_toy(ctx):

train.toy\_dataset\_refit()

@task

def model\_train\_mini(ctx):

train.mini\_dataset\_fit()

@task

def model\_history\_mini(ctx):

history.graph\_model\_history("keras\_character\_based\_ner/src/mini\_dataset.history.p", "mini")

@task

def model\_history\_toy(ctx):

history.graph\_model\_history("keras\_character\_based\_ner/src/toy\_dataset.history.p", "toy")

@task

def model\_toy\_predict\_file(ctx, file):

print(predict.model\_toy\_predict\_file(file))

@task

def model\_toy\_predict\_str(ctx, string):

print(predict.model\_toy\_predict\_str(string))

@task

def model\_mini\_predict\_file(ctx, file):

print(predict.model\_mini\_predict\_file(file))

@task

def eval\_model\_manual(ctx, dataset\_name, dataset\_size):

eval.model\_data\_validation(dataset\_name, dataset\_size)

@task

def eval\_k\_fold\_cross(ctx):

eval.k\_fold\_cross\_validation()

@task

def eval\_baseline(ctx, dataset\_name, dataset\_size, guessed\_label):

eval.calc\_eval\_baseline(dataset\_name, dataset\_size, int(guessed\_label))

@task

def flask\_start\_server(ctx):

simple\_gui.main()

## config\_util/config\_parser.py

import yaml

def parse\_config():

with open('./config.yml') as config:

conf = yaml.load(config)

return conf

## hansard\_gathering/chunk.py

from datetime import datetime

from nltk.tokenize.punkt import PunktSentenceTokenizer, PunktParameters # type: ignore

from typing import Generator, Tuple

import concurrent.futures

import glob

import itertools

import os

def chunk\_hansard\_debate\_file\_textblob(file\_path):

"""

Try TextBlob to segment a Hansard debate into its constituent sentences.

file\_path e.g. "processed\_hansard\_data/1948-04-19/Oral Answers to Questions &#8212; Oyster Industry.txt"

:param file\_path:

:return:

"""

from textblob import TextBlob # type: ignore

with open(file\_path) as f:

debate\_text = f.read()

tb = TextBlob(debate\_text)

print("Chunking up file: {}".format(file\_path))

dest\_file\_path = file\_path\

.replace("processed\_hansard\_data", "chunked\_hansard\_data")\

.replace(".txt", "")

for sentence\_number, sentence in enumerate(tb.sentences):

os.makedirs(os.path.dirname(dest\_file\_path), exist\_ok=True)

with open("{}-chunk-{}.txt"

.format(dest\_file\_path, sentence\_number), "w+") as f:

f.write(sentence.raw)

def chunk\_hansard\_debate\_file\_nltk(file\_path, tokenizer):

"""

Try NLTK to segment a Hansard debate into its constituent sentences.

file\_path e.g. "processed\_hansard\_data/1948-04-19/Oral Answers to Questions &#8212; Oyster Industry.txt"

:param file\_path: path of file to split up

:param tokenizer: An NLTK tokenizer with customisations for Hansard

"""

dest\_file\_path = file\_path.replace(".txt", "-spans.txt")

with open(file\_path) as f:

debate\_text = f.read()

print("Chunking up file: {}".format(file\_path))

sent\_spans = tokenizer.span\_tokenize(debate\_text)

sent\_spans\_str = "\n".join("({},{})".format(

sent\_start, sent\_end) for sent\_start, sent\_end in sent\_spans)

os.makedirs(os.path.dirname(dest\_file\_path), exist\_ok=True)

with open(dest\_file\_path, "w+") as f:

f.write(sent\_spans\_str)

def list\_processed\_hansard\_files(starting\_date) -> Generator[str, None, None]:

"""

Provide a starting\_date as chunking takes a long time. This allows the process to be resumable.

:param starting\_date: e.g. 1919-01-01

:return:

"""

print("Listing processed Hansard files...")

files = sorted(glob.glob("hansard\_gathering/processed\_hansard\_data/\*\*/\*.txt", recursive=True))

# With thanks to

# https://stackoverflow.com/questions/33895760/python-idiomatic-way-to-drop-items-from-a-list-until-an-item-matches-a-conditio

def date\_is\_less\_than\_starting\_date(file\_path):

file\_path\_date = file\_path.split("/")[2]

file\_path\_dt = datetime.strptime(file\_path\_date, "%Y-%M-%d")

starting\_dt = datetime.strptime(starting\_date, "%Y-%M-%d")

return file\_path\_dt < starting\_dt

filtered\_files = list(itertools.dropwhile(date\_is\_less\_than\_starting\_date, files))

for \_file in filtered\_files:

yield \_file

def nltk\_get\_tokenizer():

"""

Return a tokenizer with some customization for Hansard

:return: a Punkt tokenizer

"""

# With thanks to

# https://stackoverflow.com/questions/34805790/how-to-avoid-nltks-sentence-tokenizer-spliting-on-abbreviations

punkt\_param = PunktParameters()

# 'hon. Gentleman' is very common in Hansard!

abbreviation = ['hon', 'mr', 'mrs', 'no']

punkt\_param.abbrev\_types = set(abbreviation)

return PunktSentenceTokenizer(punkt\_param)

def chunk\_all\_hansard\_files(starting\_date):

tokenizer = nltk\_get\_tokenizer()

pool\_implementation = concurrent.futures.ProcessPoolExecutor

# pool\_implementation = concurrent.futures.ThreadPoolExecutor

with pool\_implementation(max\_workers=16) as executor:

for \_file in list\_processed\_hansard\_files(starting\_date):

# TODO try other chunking approaches: fixed-length

executor.submit(chunk\_hansard\_debate\_file\_nltk, \_file, tokenizer)

def get\_sentence\_spans(filepath) -> Generator[Tuple[int, int], None, None]:

"""

Given a filepath, yield the first and last position of each sentence in that filepath

:param filepath:

:return:

"""

debug: bool = False

with open("{}.txt".format(filepath.replace(".txt", "-spans"))) as f:

sent\_spans = f.read()

if debug:

print("DEBUG: spans file is {}".format(filepath))

# Some debates have no content, and hence no sentences. Seems to be a TWFY bug.

# TODO investigate, if there's time.

if len(sent\_spans) == 0:

return

for sent\_span in sent\_spans.split("\n"):

sent\_start, sent\_end = sent\_span.replace("(", "").replace(")", "").split(",")

yield int(sent\_start), int(sent\_end)

def display\_chunked\_hansard(filepath):

assert "processed\_hansard\_data" in filepath, \

"We only allow processed hansards to be displayed in chunks"

with open(filepath) as f:

debate = f.read()

for sent\_start, sent\_end in get\_sentence\_spans(filepath):

print(debate[int(sent\_start):int(sent\_end)])

print("@@@")

## hansard\_gathering/driver.py

from config\_util.config\_parser import parse\_config

from datetime import datetime, timedelta

import concurrent.futures

import json

import os

import requests

# Prefixes used for each content type by TWFY

# 'Content-Type': ['url-prefix', 'file-prefix']

prefixes = {'Wrans': ['wrans', 'answers'],

'WMS': ['wms', 'ministerial'],

'Debates': ['debates', 'debates']}

def download\_all\_debates(datestring, debates\_list):

"""

Given a list of debate titles, download all of them into files.

"""

for debate in debates\_list:

print("Data for {}: {}".format(datestring, debate))

title = debate[0]

xml\_url = debate[2]

if xml\_url == "N/A":

continue

xml\_data = requests.get(xml\_url).text

os.makedirs("hansard\_gathering/raw\_hansard\_data/{datestring}".format(datestring=datestring), exist\_ok=True)

with open("hansard\_gathering/raw\_hansard\_data/{datestring}/{title}.xml".format(

datestring=datestring, title=title), "w") as f:

f.write(xml\_data)

def get\_titles\_and\_download(datestring, content\_type):

commons\_titles = get\_hansard\_titles(datestring, content\_type, "commons")

lords\_titles = get\_hansard\_titles(datestring, "Debates", "lords")

download\_all\_debates(datestring, commons\_titles)

download\_all\_debates(datestring, lords\_titles)

def get\_all\_hansards(start\_year=1919, start\_month=1, start\_day=1):

"""

Generate all datestrings from now back to March 29, 1803 (when Hansard started).

Get all available debates for each.

"""

def date\_gen():

year = start\_year

month = start\_month

day = start\_day

now\_dt = datetime.now()

then\_dt = datetime(year, month, day)

# While it's less than today

while then\_dt < now\_dt:

\_datestring = "{}-{}-{}".format(

str(then\_dt.year),

str(then\_dt.month).zfill(2),

str(then\_dt.day).zfill(2))

yield \_datestring

then\_dt += timedelta(days=1)

dg = date\_gen()

with concurrent.futures.ThreadPoolExecutor(max\_workers=8) as executor:

for datestring in dg:

executor.submit(get\_titles\_and\_download, datestring, "Debates")

def get\_hansard\_titles(datestring, content\_type, house="commons"):

"""

Given a date, download the Hansard xml of specified content for the specified date

:param datestring: e.g. '2017-12-04'

:param content\_type: Wrans, WMS or Debates

:param house: commons or lords

:return List of titles extracted from the json, as well as their HTML and XML urls

"""

twfy\_key = parse\_config()['api\_key']

request\_url = 'https://www.theyworkforyou.com/api/get{}?date={}&key={}&output=json'\

.format(content\_type, datestring, twfy\_key)

if content\_type == 'Debates':

request\_url += '&type={}'.format(house)

resp = requests.get(request\_url)

resp\_data = json.loads(resp.text)

if type(resp\_data) == dict and resp\_data.get("error", "") == "No data to display":

return [("No data to display for this date", "N/A", "N/A")]

else:

titles = []

for elem in resp\_data:

entry = elem["entry"] # my dear Watson

if "listurl" in entry and "body" in entry:

titles.append((entry["body"], make\_twfy\_html\_url(entry["listurl"]),

make\_twfy\_xml\_url(entry["listurl"], content\_type)))

return titles

def make\_twfy\_html\_url(text):

return 'https://www.theyworkforyou.com{}'.format(text)

def make\_twfy\_xml\_url(text, content\_type):

return 'https://www.theyworkforyou.com/pwdata/scrapedxml/{}/{}{}.xml' \

.format(prefixes[content\_type][0],

prefixes[content\_type][1],

text.split('=')[1].split('.')[0])

## hansard\_gathering/filesystem.py

from typing import Generator, List, Tuple

from os import listdir

"""

A file for manipulating Hansard files on the filesystem, mainly to power the simple\_gui

website.

"""

def get\_dates\_list() -> List[str]:

"""

return a list of all Hansard dates available on this machine, from the filesystem.

:return:

"""

dates = listdir("hansard\_gathering/processed\_hansard\_data")

return sorted([\_file for \_file in dates if not \_file.endswith("\_num")])

def get\_debates\_by\_date(date: str) -> Generator[Tuple[int, str], None, None]:

"""

Returns a list of all debates on a particular date, according to the filesystem on this machine.

:param date:

:return:

"""

debates = sorted(listdir("hansard\_gathering/processed\_hansard\_data/{}".format(date)))

filtered\_debates: List[str] = [\_file for \_file in debates if not \_file.endswith("-spans.txt")]

for idx, debate in enumerate(filtered\_debates):

if not debate.endswith("-spans.txt"):

yield (idx, debate)

def view\_hansard(date :str, debate\_title: str) -> str:

with open("hansard\_gathering/processed\_hansard\_data/{date}/{debate\_title}".format(

date=date, debate\_title=debate\_title)) as f:

debate = f.read()

return debate

## hansard\_gathering/interpolate.py

from datetime import datetime

from nltk.tokenize import TreebankWordTokenizer # type: ignore

from nltk import ngrams # type: ignore

from typing import Set, Generator, Tuple

import concurrent.futures

import glob

import itertools

import os

# 0 = NULL

# 1 = LOC

# 2 = ORG

# 3 = PER

class NamedEntityData:

def \_\_init\_\_(self):

self.places, self.companies, self.people = self.read\_in\_all\_ne\_data()

@staticmethod

def read\_in\_all\_ne\_data() -> Tuple[Set[str], Set[str], Set[str]]:

print("Gathering all Named Entity data")

with open("ne\_data\_gathering/processed\_ne\_data/places/ALL.txt") as f:

all\_places = [line.rstrip() for line in f]

with open("ne\_data\_gathering/processed\_ne\_data/companies/ALL.txt") as f:

all\_companies = [line.rstrip() for line in f]

with open("ne\_data\_gathering/processed\_ne\_data/people/ALL.txt") as f:

all\_people = [line.rstrip() for line in f]

return set(all\_places), set(all\_companies), set(all\_people)

def get\_all(self):

return self.places, self.companies, self.people

def ngram\_span\_search\_named\_entities(ngram\_span\_window, text: str, all\_places: Set[str],

all\_companies: Set[str], all\_people: Set[str]):

"""

Take a window e.g.((0, 1), (2, 6), (7, 15), (16, 19)) from a text. Starting with the longest

suffix (0-19 here), and working back via middle (e.g. 0-15) to the first (0-1),

check all NE lists for the text bounded by these indices.

If matches, return where the match started and ended, and which NE it is.

Note that because we pad\_right, later elements in the tuple might be None, e.g.:

((98, 102), (102, 103), None, None)

:param ngram\_span\_window: As shown in example above, taken from span\_tokenize.

:param text: The debate text we are examining

:param all\_places: NE list

:param all\_companies: NE list

:param all\_people: NE list

:return: match\_start where match starts, match\_end where match ends (half-open?), ne\_type as int

where 1 = LOC, 2 = ORG, 3 = PER, 0 = null

"""

start\_index = ngram\_span\_window[0][0]

for end\_index in reversed([tup[-1] for tup in ngram\_span\_window if tup is not None]):

if text[start\_index:end\_index] in all\_places:

return start\_index, end\_index, 1

elif text[start\_index:end\_index] in all\_companies:

return start\_index, end\_index, 2

elif text[start\_index:end\_index] in all\_people:

return start\_index, end\_index, 3

return 0, 0, 0

def overlaps(ngram\_span\_window, recentest\_match\_end: int):

"""

See if the current span window already has a matched NE ending in it.

:param ngram\_span\_window: e.g.((0, 1), (2, 6), (7, 15), (16, 19))

Note that because we pad\_right, later elements in the tuple might be None, e.g.:

((98, 102), (102, 103), None, None)

:param recentest\_match\_end:

:return: True if there would be an overlap

"""

ngram\_span\_window\_no\_nones = [x for x in ngram\_span\_window if x is not None]

return ngram\_span\_window\_no\_nones[0][0] <= recentest\_match\_end

def interpolate\_one(file\_path: str, tokenizer, stage, all\_places: Set[str],

all\_companies: Set[str], all\_people: Set[str], n=4):

"""

file\_path e.g. hansard\_gathering/processed\_hansard\_data/1943-09-21/Deaths of Members-chunk-1979.txt

:param file\_path: path to file to do interpolation on

:param tokenizer: an NLTK tokenizer with span\_tokenize method

:param stage: Whether to use source files from chunked or processed stage.

:param all\_places: files with lists of \_all\_ collected examples of that NE type, \n-separated

:param all\_companies: files with lists of \_all\_ collected examples of that NE type, \n-separated

:param all\_people: files with lists of \_all\_ collected examples of that NE type, \n-separated

:param n: number to use for ngramming

:return: None (we write out to disk)

"""

assert stage in file\_path, "{} must be present in file\_path".format(stage)

print("Interpolating file {}".format(file\_path))

with open(file\_path) as f:

text = f.read()

interpolated\_text\_list = [0 for \_ in range(len(text))]

# ngrams for the text that capture their starting and ending indices.

# We pad right because we take the first word of the ngram and all its possible suffixes

# when looking for NEs.

text\_span\_ngrams = ngrams(tokenizer.span\_tokenize(text), n, pad\_right=True)

# Returns ngrams of text\_spans e.g. [((0, 1), (2, 6), (7, 15), (16, 19)), ...]

# To solve Overlapping problem, we need to know when the end of the most recent match is

recentest\_match\_end = 0

# For each ngram set, we want to try all possible suffixes against the NE lists,

# from longest to shortest so we don't miss matches.

# Once we find a match, move on to the next ngram.

for ngram\_span\_window in text\_span\_ngrams:

if overlaps(ngram\_span\_window, recentest\_match\_end):

continue

ne\_type = 0 # 1 = LOC, 2 = ORG, 3 = PER, 0 = null

match\_start, match\_end, ne\_type = ngram\_span\_search\_named\_entities(

ngram\_span\_window, text, all\_places, all\_companies, all\_people)

if ne\_type is not 0:

# This is the recentest match

recentest\_match\_end = match\_end

# Build new interpolated text by adding NE markers using list slicing

match\_len = match\_end - match\_start

interpolated\_text\_list[match\_start:match\_end] = [ne\_type for \_ in range(match\_len)]

interpolated\_file\_path = file\_path.replace("{}\_hansard\_data".format(stage), "interpolated\_hansard\_data")

interpolated\_text = "".join([str(elem) for elem in interpolated\_text\_list]).rstrip()

print("Writing out to {}".format(interpolated\_file\_path))

os.makedirs(os.path.dirname(interpolated\_file\_path), exist\_ok=True)

with open(interpolated\_file\_path, "w") as f:

f.write(interpolated\_text)

def interpolate\_one\_wrapper(file\_path, ne, stage="processed"):

"""

:param file\_path:

:param stage:

:param ne: a NamedEntityData object

:return:

"""

t = TreebankWordTokenizer()

interpolate\_one(file\_path, t, stage, \*ne.get\_all())

def list\_hansard\_files(starting\_date, stage) -> Generator[str, None, None]:

"""

stage is chunked or processed

"""

print("Listing {} Hansard files...".format(stage))

files = sorted(glob.glob("hansard\_gathering/{}\_hansard\_data/\*\*/\*.txt".format(stage), recursive=True))

# Don't interpolate our spans (chunking) files

files = list(filter(lambda elem: not elem.endswith("-spans.txt"), files))

# With thanks to

# https://stackoverflow.com/questions/33895760/python-idiomatic-way-to-drop-items-from-a-list-until-an-item-matches-a-conditio

def date\_is\_less\_than\_starting\_date(file\_path):

file\_path\_date = file\_path.split("/")[2]

file\_path\_dt = datetime.strptime(file\_path\_date, "%Y-%m-%d")

starting\_dt = datetime.strptime(starting\_date, "%Y-%m-%d")

return file\_path\_dt < starting\_dt

filtered\_files = list(itertools.dropwhile(date\_is\_less\_than\_starting\_date, files))

for \_file in filtered\_files:

yield \_file

def interpolate\_all\_hansard\_files(starting\_date):

ne = NamedEntityData()

with concurrent.futures.ThreadPoolExecutor(max\_workers=16) as executor:

for \_file in list\_hansard\_files(starting\_date, "processed"):

executor.submit(interpolate\_one\_wrapper, \_file, ne, "processed")

def display\_one\_file\_with\_interpolations(file\_path):

assert "processed\_hansard\_data" in file\_path, \

"We only support displaying interpolations on processed Hansard data"

with open(file\_path) as f:

text = f.readlines()

with open(file\_path.replace("processed\_hansard\_data", "interpolated\_hansard\_data")) as f:

interpolation\_digits = f.read()

so\_far = 0

for line in text:

length = len(line)

print(line, end='')

print(interpolation\_digits[so\_far:so\_far + length], end='\n')

so\_far += length

def fix\_uninterpolated\_hansards(starting\_date):

"""

Fix a bug in the Hansard interpolations - Hansards with unbalanced double quotes cannot be span\_tokenized...

:param starting\_date:

:return:

"""

debug = True

ne = NamedEntityData()

for \_file in list\_hansard\_files(starting\_date, "processed"):

interpolated\_file\_path = \_file.replace("processed\_hansard\_data", "interpolated\_hansard\_data")

if not os.path.exists(interpolated\_file\_path):

if debug:

print("Found uninterpolated file: {}".format(\_file))

with open(\_file, "w+") as f:

text = f.read()

f.write(text.replace('"', "'"))

interpolate\_one\_wrapper(\_file, ne, "processed")

## hansard\_gathering/numerify.py

from typing import List

import os

def numerify\_one\_to\_file(filepath, alphabet, maxlen):

"""

Convert a chunked hansard file's alphabet into numberical indices as required by the Keras implementation

for char-ner

:param filepath: path to the chunked Hansard file (a single sentence from a Hansard debate)

e.g. "hansard\_gathering/chunked\_hansard\_data/1938-10-04/Oral Answers to Questions &#8212; Anti-Aircraft Defence, London.-chunk-0.txt"

:param alphabet: a CharBasedNERAlphabet object containing the alphabet in use

PLEASE NOTE this function does not do any padding - it is envisaged that padding should be done later, closer

to into Keras. Otherwise, if sentence\_maxlen changed, the numerifying would all have to be revisited.

"""

assert "processed\_hansard\_data" in filepath, \

"We only numerify processed Hansard debates"

dest\_filepath = filepath.replace("processed\_hansard\_data", "numerified\_hansard\_data")

print("Converting file {} to numbers".format(filepath))

with open(filepath, "r") as f:

text = f.read()

numerified\_text\_list = numerify\_text(text, alphabet, maxlen)

numerified\_text = ",".join([str(elem) for elem in numerified\_text\_list])

os.makedirs(os.path.dirname(dest\_filepath), exist\_ok=True)

with open(dest\_filepath, "w") as f:

f.write(numerified\_text)

def numerify\_text(text, alphabet, maxlen) -> List[int]:

"""

Take a text and return its numerical representation as numbers in a List.

:param text:

:param alphabet:

:return:

"""

numerified\_text\_list = []

for idx, char in enumerate(text):

if idx > maxlen:

break

index = alphabet.get\_char\_index(char)

numerified\_text\_list.append(index)

return numerified\_text\_list

## hansard\_gathering/preprocessing.py

from lxml import etree # type: ignore

from typing import Generator, List

import glob

import os

def unxml\_hansard\_document(document\_text):

"""

Do preprocessing on a hansard doc expressed in text-xml. This includes html-unescaping and

removing tags. It could change in future.

:param document\_text:

:return:

"""

# Declare that strings are Unicode-encoded

parser = etree.XMLParser(ns\_clean=True, recover=True, encoding='utf-8')

tree = etree.fromstring(document\_text.encode('utf-8'), parser=parser)

notags = etree.tostring(tree, encoding='utf8', method='text')

return notags

def process\_hansard\_file(file\_path):

"""

file\_path e.g. "hansard\_gathering/raw\_hansard\_data/1919-02-04/MyDebate.xml"

"""

print("Processing {}".format(file\_path))

dest\_path = file\_path.replace("raw\_hansard\_data", "processed\_hansard\_data").replace(".xml", ".txt")

with open(file\_path) as f:

document\_text = f.read()

processed\_document\_text = unxml\_hansard\_document(document\_text)

os.makedirs(os.path.dirname(dest\_path), exist\_ok=True)

with open(dest\_path, 'wb+') as f:

f.write(processed\_document\_text)

# Clean up as we go along to save Matt's Hard Drive!

os.remove(file\_path)

def list\_raw\_hansard\_files() -> Generator[str, None, None]:

for \_file in glob.glob("hansard\_gathering/raw\_hansard\_data/\*\*/\*.xml", recursive=True):

yield \_file

def process\_all\_hansard\_files():

for hansard\_file in list\_raw\_hansard\_files():

process\_hansard\_file(hansard\_file)

## keras\_character\_based\_ner/src/matt/alphabet\_management.py

from keras\_character\_based\_ner.src.alphabet import CharBasedNERAlphabet

from keras\_character\_based\_ner.src.matt.file\_management import get\_texts

import pickle

def generate\_alphabet():

return CharBasedNERAlphabet(get\_texts())

def pickle\_alphabet():

alph = generate\_alphabet()

with open("keras\_character\_based\_ner/src/alphabet.p", "wb") as f:

pickle.dump(alph, f)

def display\_pickled\_alphabet():

alph = get\_pickled\_alphabet()

print(alph)

for i, ch in enumerate(alph):

print("{}: {}".format(i, ch))

def get\_pickled\_alphabet():

with open("keras\_character\_based\_ner/src/alphabet.p", "rb") as f:

return pickle.load(f)

## keras\_character\_based\_ner/src/matt/dataset\_hashing.py

import os

import glob

from collections import defaultdict

from typing import List, Set

def get\_total\_number\_of\_buckets() -> int:

return 320

def get\_bucket\_numbers\_for\_dataset\_name(dataset\_name: str) -> List[int]:

"""

Function to control bucket quantities and relative sizes of datasets

:param dataset\_name: ALL, train, dev or test

:return: a list of ints for the bucket numbers containing file lists

which, when unioned together, comprise that dataset.

"""

if dataset\_name == "ALL":

return list(range(320))

elif dataset\_name == "train":

return list(range(0, 160))

elif dataset\_name == "dev":

return list(range(160, 240))

elif dataset\_name == "test":

return list(range(240, 320))

# Small set of debates to build an alphabet off

elif dataset\_name == "alphabet-sample":

return [0]

else:

return []

def archive\_old\_bucket\_allocations():

"""

Move old bucket files in hansard\_gathering/data\_buckets to an archive so they're not lost

"""

os.makedirs("hansard\_gathering/data\_buckets\_archive", exist\_ok=True)

file\_list = sorted(glob.glob("hansard\_gathering/data\_buckets/\*.txt"))

for \_file in file\_list:

new\_dest = \_file.replace("data\_buckets", "data\_buckets\_archive")

os.rename(\_file, new\_dest)

def rehash\_datasets():

"""

Hash all Hansard debates into 3 datasets:

train

test

dev

(ALL)

We take a hash of the date-and-debate-name part of each filepath, then use modulo to

bucket this.

"""

archive\_old\_bucket\_allocations()

# bucket allocations: 4 for train, 2 for dev, 2 for test

num\_of\_buckets = get\_total\_number\_of\_buckets()

debug = False

os.makedirs("hansard\_gathering/data\_buckets", exist\_ok=True)

files\_by\_bucket = defaultdict(lambda: set())

file\_list = sorted(glob.glob(

"hansard\_gathering/processed\_hansard\_data/\*\*/\*.txt", recursive=True))

file\_list = list(filter(lambda elem: not elem.endswith("-spans.txt"), file\_list))

for \_file in file\_list:

date\_filename\_path = "/".join(\_file.split("/")[2:])

hash\_val = hash(date\_filename\_path)

bucket\_num = hash\_val % num\_of\_buckets

files\_by\_bucket[bucket\_num].add(\_file)

print("hashed {} into bucket {}".format(\_file, bucket\_num)) if debug else None

for bucket\_num in files\_by\_bucket.keys():

with open("hansard\_gathering/data\_buckets/{}.txt".format(bucket\_num), "w") as f:

filepaths = sorted(files\_by\_bucket[bucket\_num])

for filepath in filepaths:

f.write(filepath + "\n")

## keras\_character\_based\_ner/src/matt/eval.py

"""

Evaluate the trained Keras model

"""

from sklearn.model\_selection import KFold # type: ignore

from keras\_character\_based\_ner.src.matt.file\_management import unpickle\_large\_file

from keras\_character\_based\_ner.src.config import Config

from keras\_character\_based\_ner.src.matt.persist import AlphabetPreloadedCharBasedNERDataset,\

LoadedToyModel, SavedCharacterBasedLSTMModel

from typing import List

import numpy as np # type: ignore

def init\_config\_dataset():

"""

Create a vanilla Config and CharBasedNERDataset object, required for constructing a model.

We don't actually use the dataset at all in evaluation - we just use the model weights.

:return:

"""

config = Config()

dataset = AlphabetPreloadedCharBasedNERDataset()

return config, dataset

def k\_fold\_cross\_validation():

"""

Train and validate a new model using k-fold cross validation.

With thanks to

https://datascience.stackexchange.com/questions/27212/stratifiedkfold-valueerror-supported-target-types-are-binary-multiclass

for the guide on implementing k-fold in keras

:return:

"""

x = unpickle\_large\_file("keras\_character\_based\_ner/src/x\_np-train-toy.p")

y = unpickle\_large\_file("keras\_character\_based\_ner/src/y\_np-train-toy.p")

loss\_scores: List = []

categorical\_accuracy\_scores: List = []

non\_null\_label\_accuracy\_scores: List = []

# Use a new CharBasedLSTMModel with saving capabilities, and manual evaluation and fit methods

kf = KFold(n\_splits=10)

for train, test in kf.split(x):

model = SavedCharacterBasedLSTMModel(\*init\_config\_dataset())

model.manual\_fit(x\_train=x[train], y\_train=y[train], batch\_size=Config.batch\_size,

epochs=3)

loss, categorical\_accuracy, non\_null\_label\_accuracy = model.manual\_evaluate(

x\_test=x[test], y\_test=y[test], batch\_size=Config.batch\_size)

loss\_scores.append(loss)

categorical\_accuracy\_scores.append(categorical\_accuracy)

non\_null\_label\_accuracy\_scores.append(non\_null\_label\_accuracy)

scores\_dict = {

"loss\_scores": loss\_scores,

"categorical\_accuracy\_scores": categorical\_accuracy\_scores,

"non\_null\_label\_accuracy": non\_null\_label\_accuracy\_scores,

}

print(scores\_dict)

for title, scores in scores\_dict.items():

# With thanks to

# https://machinelearningmastery.com/evaluate-performance-deep-learning-models-keras/

print("Mean for {} is {}".format(title, np.mean(scores)))

print("Standard deviation for {} is {}".format(title, np.std(scores)))

def model\_data\_validation(dataset\_name, dataset\_size):

"""

Validate toy model on a bucket of text it hasn't been trained on (train) or validated on (dev) yet.

This is because the 'test' dataset used to train the model gave NaN for validation loss.

:param dataset\_name: train, test, dev

:param dataset\_size: toy or mini

:return:

"""

x = unpickle\_large\_file("keras\_character\_based\_ner/src/x\_np-{}-{}.p".format(

dataset\_name, dataset\_size))

y = unpickle\_large\_file("keras\_character\_based\_ner/src/y\_np-{}-{}.p".format(

dataset\_name, dataset\_size))

# Load in the pre-trained Toy model off disk

model = LoadedToyModel(\*init\_config\_dataset())

metrics = model.manual\_evaluate(x, y, Config.batch\_size)

print("On dataset {dataset\_size}-{dataset\_name}; ".format(

dataset\_size=dataset\_size, dataset\_name=dataset\_name))

print("loss: {}".format(metrics[0]))

print("categorical accuracy: {}".format(metrics[1]))

print("non null label accuracy: {}".format(metrics[2]))

if len(metrics) > 3: # If extended metrics have been added...

print("precision\_null: {}".format(metrics[3]))

print("precision\_loc: {}".format(metrics[4]))

print("precision\_org: {}".format(metrics[5]))

print("precision\_per: {}".format(metrics[6]))

print("recall\_null: {}".format(metrics[7]))

print("recall\_loc: {}".format(metrics[8]))

print("recall\_org: {}".format(metrics[9]))

print("recall\_per: {}".format(metrics[10]))

print("f1\_null: {}".format(metrics[11]))

print("f1\_loc: {}".format(metrics[12]))

print("f1\_org: {}".format(metrics[13]))

print("f1\_per: {}".format(metrics[14]))

def calc\_eval\_baseline(dataset\_name, dataset\_size, baseline\_label=0):

"""

Calculate a basic evaluation baseline, of assuming all labels are NULL

:param dataset\_name: train, test or dev

:param dataset\_size: toy or mini

:param baseline\_label: the label the baseline should always try to guess

:return:

"""

def all\_zeros(\_char\_onehot):

return all(elem == 0 for elem in \_char\_onehot)

def not\_null(\_char\_onehot):

return \_char\_onehot[0] == 0 and 1 in \_char\_onehot[1:]

def un\_one\_hot(\_char\_onehot):

for pos, val in enumerate(\_char\_onehot):

if val == 1:

return pos

y = unpickle\_large\_file("keras\_character\_based\_ner/src/y\_np-{}-{}.p".format(

dataset\_name, dataset\_size))

num\_of\_chars = 0

num\_of\_not\_nulls = 0

num\_correctly\_guessed = 0

for sample in y:

for char\_onehot in sample:

if all\_zeros(char\_onehot):

pass # This is padding, ignore

else:

num\_of\_chars += 1

if not\_null(char\_onehot):

num\_of\_not\_nulls += 1

if un\_one\_hot(char\_onehot) == baseline\_label:

num\_correctly\_guessed += 1

# The baseline will be wrong for every not-null in the chars

baseline\_inaccuracy = float(num\_of\_not\_nulls) / float(num\_of\_chars)

baseline\_accuracy = 1 - baseline\_inaccuracy

baseline\_guessed\_accuracy = float(num\_correctly\_guessed) / float(num\_of\_chars)

print(baseline\_accuracy)

print(baseline\_guessed\_accuracy)

## keras\_character\_based\_ner/src/matt/file\_management.py

from typing import List, Generator, Any

import os

import pickle

from keras\_character\_based\_ner.src.matt.dataset\_hashing import get\_bucket\_numbers\_for\_dataset\_name

from hansard\_gathering import chunk

def get\_all\_hansard\_files(dataset\_name: str) -> Generator[str, None, None]:

"""

Return generator of all file names in a given dataset.

:param dataset\_name: train, dev, test or ALL

:return:

"""

print("Listing Hansard debate files from dataset {}...".format(dataset\_name))

bucket\_numbers = get\_bucket\_numbers\_for\_dataset\_name(dataset\_name)

file\_list = []

for bucket\_number in bucket\_numbers:

with open("hansard\_gathering/data\_buckets/{}.txt".format(bucket\_number)) as f:

file\_list.extend([filename.rstrip() for filename in f.readlines()])

for \_file in file\_list:

yield \_file

def get\_hansard\_span\_files(dataset\_name: str) -> Generator[str, None, None]:

"""

For a given dataset name, yield just the span files (list of sentence starts

and stops) for each debate in that dataset. Only used to get the total

number of sentences in the dataset at present.

:param dataset\_name:

:return:

"""

print("Listing Hansard span files from dataset {}...".format(dataset\_name))

bucket\_numbers = get\_bucket\_numbers\_for\_dataset\_name(dataset\_name)

file\_list = []

for bucket\_number in bucket\_numbers:

with open("hansard\_gathering/data\_buckets/{}.txt".format(bucket\_number)) as f:

file\_list.extend([filename.rstrip().replace(".txt", "-spans.txt") for filename in f.readlines()])

for span\_file in file\_list:

yield span\_file

def file\_lines(fname: str) -> int:

"""

Fast implementation to get number of lines in a file - useful with span files,

to count total number of different sentences.

:param fname:

:return:

"""

# with thanks to

# https://stackoverflow.com/questions/845058/how-to-get-line-count-cheaply-in-python

with open(fname) as f:

i = 0

for i, l in enumerate(f):

pass

return i + 1

def write\_total\_number\_of\_hansard\_sentences\_to\_file(dataset\_name: str):

"""

Get num of sentences in a particular dataset, dev, test or train.

Also accept dataset\_name 'ALL' while I work on dataset divisions.

Count number of sentences in the -spans files and write this out to

disk to save time.

:param dataset\_name: must be dev, test or train

:return:

"""

# Run on 25 July 2018 this was 182582013

sentences\_total = 0

for span\_file in get\_hansard\_span\_files(dataset\_name):

sentences\_total += file\_lines(span\_file)

with open("hansard\_gathering/processed\_hansard\_data/{}\_total\_sentences\_num".format(dataset\_name), "w+") as f:

f.write(str(sentences\_total))

def get\_total\_number\_of\_hansard\_sentences(dataset\_name: str):

"""

Get num of sentences in a particular dataset, dev, test or train.

Also accept dataset\_name 'ALL' while I work on dataset divisions.

Count number of sentences in the -spans files and return this to caller.

:param dataset\_name: must be dev, test or train, or ALL.

:return:

"""

print("Calculating total number of sentences in {} dataset...".format(dataset\_name))

sentences\_total = 0

for span\_file in get\_hansard\_span\_files(dataset\_name):

sentences\_total += file\_lines(span\_file)

return sentences\_total

def read\_total\_number\_of\_hansard\_sentences\_from\_file(dataset\_name) -> int:

"""

Get num of samples in a particular dataset, dev, test or train.

Also accept dataset\_name 'ALL' while I work on dataset divisions.

Read this information from disk.

:param dataset\_name: must be dev, test or train

:return:

"""

with open("hansard\_gathering/processed\_hansard\_data/{}\_total\_sentences\_num".format(dataset\_name), "r") as f:

sentences = f.read()

return int(sentences)

def get\_chunked\_hansard\_interpolations(dataset\_name: str) -> Generator[str, None, None]:

"""

:param dataset\_name: dev, test or train

Generator that goes over all Hansard debate files and returns their next sentence worth of interpolation-numbers,

using a span file.

:return:

"""

for \_file in get\_all\_hansard\_files(dataset\_name):

interpolations\_file = \_file.replace(

"processed\_hansard\_data", "interpolated\_hansard\_data")

with open(interpolations\_file, "r") as f:

interpolations\_data = f.read()

for span\_start, span\_end in chunk.get\_sentence\_spans(\_file):

yield interpolations\_data[span\_start:span\_end]

def unpickle\_large\_file(filepath) -> Any:

"""

See https://stackoverflow.com/questions/31468117/python-3-can-pickle-handle-byte-objects-larger-than-4gb

MacOS has a bug which stops objects larger than 4GB from being written out to file. What a pain!

:param filepath:

:return:

"""

max\_bytes = 2\*\*31 - 1

bytes\_in = bytearray(0)

input\_size = os.path.getsize(filepath)

with open(filepath, 'rb') as f\_in:

for \_ in range(0, input\_size, max\_bytes):

bytes\_in += f\_in.read(max\_bytes)

return pickle.loads(bytes\_in) # may need protocol=4 ?

def pickle\_large\_file(data\_structure, filepath):

"""

See https://stackoverflow.com/questions/31468117/python-3-can-pickle-handle-byte-objects-larger-than-4gb

MacOS has a bug which stops objects larger than 4GB from being written out to file. What a pain!

:param data\_structure:

:param filepath:

:return:

"""

max\_bytes = 2\*\*31 - 1

bytes\_out = pickle.dumps(data\_structure, protocol=4)

with open(filepath, 'wb') as f\_out:

for idx in range(0, len(bytes\_out), max\_bytes):

f\_out.write(bytes\_out[idx:idx+max\_bytes])

def get\_texts() -> Generator[str, None, None]:

"""

Return the texts from hansard files, without chunking into sentences. This

is only required for the keras dataset to build an alphabet, so we only need

to return a small subset. We make a bucket set called alphabet-sample for this.

:return:

"""

return get\_chunked\_hansard\_texts("alphabet-sample")

def get\_chunked\_hansard\_texts(dataset\_name: str) -> Generator[str, None, None]:

"""

:param dataset\_name: dev, test or train

Generator that goes over all Hansard debate files and returns their next sentence,

using their spans file. This is required to build the X tensor - the resulting

sentence-spans are each numerified before being turned into numpy arrays.

:return:

"""

for \_file in get\_all\_hansard\_files(dataset\_name):

with open(\_file) as f:

debate = f.read()

for chunk\_start, chunk\_end in chunk.get\_sentence\_spans(\_file):

yield debate[chunk\_start:chunk\_end]

## keras\_character\_based\_ner/src/matt/history.py

from keras\_character\_based\_ner.src.matt.file\_management import unpickle\_large\_file

from typing import Dict

# Examples in this file taken from

# Deep Learning with Python

# by François Chollet

# Published by Manning Publications, 2017

# Chapter 6 'Deep learning for text and sequences'

def graph\_model\_history(filepath, dest\_file\_name):

"""

Open a pickled `history` object created by a train (fit() invocation),

and graph out the non-null-label accuracy, categorical accuracy, and loss

on both training and validation datasets.

:param filepath: path to the pickled history file.

:param dest\_file\_name: a destination file name for the files. This name will

be used 3 times, with words added to indicate which metric is shown in its graph.

e.g. 'toy-model'

:return:

"""

import matplotlib # type: ignore

matplotlib.use('TkAgg')

import matplotlib.pyplot as plt # type: ignore

history\_dict = unpickle\_large\_file(filepath)

cat\_acc = history\_dict['categorical\_accuracy']

non\_null\_label\_acc = history\_dict['non\_null\_label\_accuracy']

loss = history\_dict['loss']

val\_loss = history\_dict['val\_loss']

val\_cat\_acc = history\_dict['val\_categorical\_accuracy']

val\_non\_null\_label\_acc = history\_dict['val\_non\_null\_label\_accuracy']

epochs = range(1, len(cat\_acc) + 1)

plt.figure(1)

plt.plot(epochs, cat\_acc, 'bo', label='Training acc')

plt.plot(epochs, val\_cat\_acc, 'b', label='Validation acc')

plt.title('Training and validation accuracy')

plt.legend()

plt.savefig('keras\_character\_based\_ner/graphs/{}-acc.png'.format(dest\_file\_name))

plt.figure(2)

plt.plot(epochs, loss, 'bo', label='Training loss')

plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.legend()

plt.savefig('keras\_character\_based\_ner/graphs/{}-loss.png'.format(dest\_file\_name))

plt.figure(3)

plt.plot(epochs, non\_null\_label\_acc, 'bo', label='Non null label accuracy')

plt.plot(epochs, val\_non\_null\_label\_acc, 'b', label='Validation Non null label accuracy')

plt.title('Training and validation non null label accuracy')

plt.legend()

plt.savefig('keras\_character\_based\_ner/graphs/{}-non-null-label-acc.png'.format(dest\_file\_name))

## keras\_character\_based\_ner/src/matt/minify\_dataset.py

"""

Make a dataset smaller - the toy dataset we chose is too large to get a feel for saving out the model

"""

from keras\_character\_based\_ner.src.matt.file\_management import unpickle\_large\_file, pickle\_large\_file

from keras\_character\_based\_ner.src.config import Config

from keras.preprocessing.sequence import pad\_sequences # type: ignore

MAX\_BATCH = 4000

def minify(path\_to\_list\_file):

"""

Truncate a python list object to MAX\_BATCH batches, so we can produce a smaller dataset to

feed the model.

:param path\_to\_list\_file:

:return: a numpy array with 1st dimension truncated to MAX\_BATCH

"""

assert "list" in path\_to\_list\_file, \

"minify MUST take a list object, not a numpy array"

print("Minifying {}".format(path\_to\_list\_file))

list\_obj = unpickle\_large\_file(path\_to\_list\_file)

truncated\_list\_obj = list\_obj[:MAX\_BATCH]

return pad\_sequences(truncated\_list\_obj, maxlen=Config.sentence\_max\_length)

def minify\_all():

"""

Make a mini-version of all tensors in the 'toy' dataset

:return:

"""

files = ["x\_list-dev-toy.p", "x\_list-test-toy.p", "x\_list-train-toy.p",

"y\_list-dev-toy.p", "y\_list-test-toy.p", "y\_list-train-toy.p"]

for \_file in files:

mini\_data = minify("keras\_character\_based\_ner/src/{}".format(\_file))

mini\_file\_name = \_file.replace("-toy.p", "-mini.p").replace("\_list", "\_np")

pickle\_large\_file(mini\_data, "keras\_character\_based\_ner/src/{}".format(mini\_file\_name))

## keras\_character\_based\_ner/src/matt/model\_integration.py

# MIR file added to provide integration with Keras

from keras\_character\_based\_ner.src.matt.alphabet\_management import get\_pickled\_alphabet

from keras\_character\_based\_ner.src.matt.file\_management import get\_all\_hansard\_files

from keras\_character\_based\_ner.src.matt.file\_management import pickle\_large\_file, unpickle\_large\_file

from keras\_character\_based\_ner.src.matt.file\_management import get\_chunked\_hansard\_texts

from keras\_character\_based\_ner.src.matt.file\_management import get\_chunked\_hansard\_interpolations

from keras\_character\_based\_ner.src.matt.file\_management import get\_total\_number\_of\_hansard\_sentences

from keras\_character\_based\_ner.src.config import Config

from typing import List, Tuple

from hansard\_gathering import numerify, chunk

from statistics import median

class NoDatasetSizeFoundException(Exception):

"""

Exception to raise if use asks for a dataset size that does not exist

"""

pass

def get\_labels():

"""

Return list of different labels used for NEs in the dataset.

:return:

"""

# 1 = LOC, 2 = ORG, 3 = PER, 0 = null

return ["LOC", "ORG", "PER"]

def create\_x\_toy(sentence\_maxlen, dataset\_name):

"""

Create X tensor by reading in all debates in the current dataset,

taking them chunk by chunk, converting the letters to numbers, and

building a list-of-lists-of-ints structure.

Then use keras pad\_sequences to ensure uniform length (len == sentence\_maxlen)

with left-hand-side padding, and write out both the list object and pad\_sequences'

resulting numpy array to pickled files.

:param sentence\_maxlen:

:param dataset\_name: train, test, dev or eval

:return:

"""

from keras.preprocessing.sequence import pad\_sequences # type: ignore

debug = True

if debug:

print("Generating X tensor")

# Model is overfitting. Try reducing tensor size for each dataset

# to see if that fixes NaN-validation problem.

cutoff = {

"train": 1000000,

"test": 60000,

"dev": 60000,

}

alphabet = get\_pickled\_alphabet()

x\_list = []

for idx, hansard\_sentence in enumerate(get\_chunked\_hansard\_texts(dataset\_name)):

if idx >= cutoff[dataset\_name]:

break

numbers\_list = numerify.numerify\_text(hansard\_sentence, alphabet, sentence\_maxlen)

x\_list.append(numbers\_list)

if debug:

print("Building x, progress {} %".format((idx / cutoff[dataset\_name]) \* 100)) if idx % 5000 == 0 else None

# Write X so we don't have to regenerate every time...

pickle\_large\_file(x\_list, "keras\_character\_based\_ner/src/x\_list-{}-toy.p".format(dataset\_name))

# pad\_sequences takes care of enforcing sentence\_maxlen for us

x\_np = pad\_sequences(x\_list, maxlen=sentence\_maxlen)

# Write X so we don't have to regenerate every time...

pickle\_large\_file(x\_np, "keras\_character\_based\_ner/src/x\_np-{}-toy.p".format(dataset\_name))

def onehot(i: int, maxlen: int) -> List[int]:

"""

Turn an integer into a onehot vector for that integer

:param i: Int to change to onehot

:param maxlen: length of the onehot vector

"""

onehot\_vector = [0 for \_ in range(maxlen)]

onehot\_vector[i] = 1

return onehot\_vector

def create\_y\_toy(sentence\_maxlen, dataset\_name):

""""

Create Y tensor by reading in the required spans of each chunk of the debates

in the current dataset, and returning the equivalent list of NE numbers

from the interpolated file for that debate.

As per create\_x, we make a Python list-of-lists-of-ints, pickle it, then

use pad\_sequences to make a numpy array, which we also pickle.

"""

from keras.preprocessing.sequence import pad\_sequences # type: ignore

debug = True

if debug:

print("Generating Y tensor")

# Model is overfitting. Try reducing tensor size for each dataset

# to see if that fixes NaN-validation problem.

cutoff = {

"train": 1000000,

"test": 60000,

"dev": 60000,

}

y\_list = []

onehot\_vector\_length = len(get\_labels()) + 1 # list of labels plus one extra for non-NE

for idx, interpolated\_hansard\_sentence in enumerate(

get\_chunked\_hansard\_interpolations(dataset\_name)):

if idx >= cutoff[dataset\_name]:

break

y\_list.append([onehot(int(num), onehot\_vector\_length) for num in interpolated\_hansard\_sentence])

if debug:

print("Building y, progress {} %".format((idx / cutoff[dataset\_name]) \* 100)) if idx % 5000 == 0 else None

# Write Y so we don't have to regenerate every time...

pickle\_large\_file(y\_list, "keras\_character\_based\_ner/src/y\_list-{}-toy.p".format(dataset\_name))

# pad\_sequences takes care of enforcing sentence\_maxlen for us

y\_np = pad\_sequences(y\_list, maxlen=sentence\_maxlen)

# Write X so we don't have to regenerate every time...

pickle\_large\_file(y\_np, "keras\_character\_based\_ner/src/y\_np-{}-toy.p".format(dataset\_name))

def get\_median\_sentence\_length(dataset\_name) -> float:

"""

Find median length of all sentences in the corpus - so we can make sensible decisions about chunking for tensors.

:param dataset\_name:

:return:

"""

sentence\_lengths = []

for \_file in get\_all\_hansard\_files(dataset\_name):

for span\_start, span\_end in chunk.get\_sentence\_spans(\_file):

span\_len = span\_end - span\_start

sentence\_lengths.append(span\_len)

return median(sentence\_lengths)

def get\_x\_y(dataset\_name, dataset\_size="toy") -> Tuple:

"""

Returns a Python tuple x and y, where x and y are Numpy arrays!

x: Array of shape (batch\_size, sentence\_maxlen).

Entries in dimension 1 are alphabet indices, index 0 is the padding symbol

y: Array of shape (batch\_size, sentence\_maxlen, self.num\_labels).

Entries in dimension 2 are label indices, index 0 is the null label

I guess batch\_size here refers to the WHOLE batch?

"""

if dataset\_size == "toy":

x\_np = unpickle\_large\_file("keras\_character\_based\_ner/src/x\_np-{}-toy.p".format(dataset\_name))

y\_np = unpickle\_large\_file("keras\_character\_based\_ner/src/y\_np-{}-toy.p".format(dataset\_name))

elif dataset\_size == "mini":

x\_np = unpickle\_large\_file("keras\_character\_based\_ner/src/x\_np-{}-mini.p".format(dataset\_name))

y\_np = unpickle\_large\_file("keras\_character\_based\_ner/src/y\_np-{}-mini.p".format(dataset\_name))

else:

raise NoDatasetSizeFoundException()

return x\_np, y\_np

def get\_x\_y\_generator(sentence\_maxlen, dataset\_name):

"""

Generator that returns a tuple each time, of inputs/targets as Numpy arrays. Each tuple

is a batch used in training.

Given the size of data we are dealing with, I think this will be necessary

to integrate with the keras. We should probably decide a batch size B \*within\*

out dataset, then dynamically do a create\_x and create\_y on that batch-size

within the dataset's debates, and yield (short) x and y tensors.

:return: Generator object that yields tuples (x, y), same as in get\_x\_y()

"""

from keras.preprocessing.sequence import pad\_sequences # type: ignore

debug: bool = False

alphabet = get\_pickled\_alphabet()

onehot\_vector\_length = len(get\_labels()) + 1 # list of labels plus one extra for non-NE

batch\_length: int = Config.batch\_size

batch\_position: int = 0

total\_sentences: int = get\_total\_number\_of\_hansard\_sentences(dataset\_name)

print("Preparing generators...")

x\_generator = get\_chunked\_hansard\_texts(dataset\_name)

y\_generator = get\_chunked\_hansard\_interpolations(dataset\_name)

for batch\_idx in (batch\_position, total\_sentences, batch\_length):

print("Generating new batch for keras, on sentence {} of {}"

.format(batch\_position, total\_sentences))

x\_list = []

y\_list = []

batch\_end = min(batch\_idx + batch\_length, total\_sentences)

for idx in range(batch\_idx, batch\_end):

if debug:

print("Generating sequence {} of {}, the end of this batch"

.format(idx, batch\_end - 1))

x\_raw = next(x\_generator)

x\_processed = numerify.numerify\_text(x\_raw, alphabet, sentence\_maxlen)

x\_list.append(x\_processed)

y\_raw = next(y\_generator)

y\_processed = [onehot(int(num), onehot\_vector\_length) for num in y\_raw]

y\_list.append(y\_processed)

batch\_position = batch\_end

print("Padding and converting to numpy arrays...")

x\_np = pad\_sequences(x\_list, maxlen=sentence\_maxlen)

y\_np = pad\_sequences(y\_list, maxlen=sentence\_maxlen)

print("Batch generation done up to {}, yielding to Keras model".format(batch\_position))

yield(x\_np, y\_np)

## keras\_character\_based\_ner/src/matt/persist.py

from keras\_character\_based\_ner.src.model import CharacterBasedLSTMModel

from keras\_character\_based\_ner.src.dataset import CharBasedNERDataset

from keras\_character\_based\_ner.src.matt.file\_management import unpickle\_large\_file

from typing import Callable, Dict

from keras.models import load\_model, Sequential # type: ignore

class SavedCharacterBasedLSTMModel(CharacterBasedLSTMModel):

def \_\_init\_\_(self, config, dataset):

super().\_\_init\_\_(config, dataset)

def save(self, filepath):

"""

MIR Added method to save model to disk

:param filepath: file path under which to save

:return:

"""

return self.model.save(filepath)

def manual\_evaluate(self, x\_test, y\_test, batch\_size):

"""

Provide a hook to manually run model.evaluate() without needing to create

a new Dataset object each time. Useful for cross-fold evaluation.

:param x\_test: x of the test dataset

:param y\_test: y of the test dataset

:param batch\_size:

:return:

"""

return self.model.evaluate(x=x\_test, y=y\_test, batch\_size=batch\_size)

def manual\_fit(self, x\_train, y\_train, batch\_size, epochs):

"""

Provide a hook to manually run model.fit() without needing

to create a new Dataset object each time. Useful for cross-fold evaluation.

:param x\_train:

:param y\_train:

:param batch\_size:

:param epochs:

:return:

"""

return self.model.fit(x=x\_train,

y=y\_train,

batch\_size=batch\_size,

epochs=epochs,

verbose=1

)

def predict\_long\_str(self, s: str):

"""

Override CharacterBasedLSTMModel's own predict\_str. This is because

we want to be able to predict strings that are longer than config.sentence\_max\_length.

Other than setting the 2nd argument of str\_to\_x to 'sys.maxsize', the rest is unchanged

from the original predict\_str function.

:param s:

:return:

"""

x = self.dataset.str\_to\_x(s, len(s))

predicted\_classes = self.predict\_x(x)

chars = self.dataset.x\_to\_str(x)[0]

labels = self.dataset.y\_to\_labels(predicted\_classes)[0]

return list(zip(chars, labels))

class LoadedToyModel(SavedCharacterBasedLSTMModel):

"""

A loaded model with all the functionality of a CharacterBasedLSTMModel

"""

def get\_model(self):

print("Loading in model from previous training of Toy dataset")

model\_path = "keras\_character\_based\_ner/src/toy\_dataset.keras.h5"

custom\_objects: Dict[str, Callable] = {

'non\_null\_label\_accuracy': SavedCharacterBasedLSTMModel.non\_null\_label\_accuracy

}

model: Sequential = load\_model(model\_path, custom\_objects=custom\_objects)

print("Completed loading in model from previous training of Toy dataset")

return model

class LoadedMiniModel(SavedCharacterBasedLSTMModel):

"""

A loaded model with all the functionality of a CharacterBasedLSTMModel

"""

def get\_model(self):

print("Loading in model from previous training of Mini dataset")

model\_path = "keras\_character\_based\_ner/src/mini\_dataset.keras.h5"

custom\_objects: Dict[str, Callable] = {

'non\_null\_label\_accuracy': SavedCharacterBasedLSTMModel.non\_null\_label\_accuracy

}

self.model: Sequential = load\_model(model\_path, custom\_objects=custom\_objects)

class AlphabetPreloadedCharBasedNERDataset(CharBasedNERDataset):

"""

A version of CharBasedNERDataset where we don't need to create an alphabet dynamically

by unioning together a set of texts. This means a. it's quicker to load the whole model

and b. we can run the model independently of having the texts to hand.

"""

def \_\_init\_\_(self):

print("Using pickled alphabet for dataset")

self.alphabet = unpickle\_large\_file("keras\_character\_based\_ner/src/alphabet.p")

self.labels = self.BASE\_LABELS + self.get\_labels()

self.num\_labels = len(self.labels)

self.num\_to\_label = {}

self.label\_to\_num = {}

self.init\_mappings()

## keras\_character\_based\_ner/src/matt/predict.py

from keras\_character\_based\_ner.src.config import Config

from keras\_character\_based\_ner.src.matt.persist import LoadedToyModel, LoadedMiniModel

from keras\_character\_based\_ner.src.dataset import CharBasedNERDataset

def model\_toy\_predict\_file(file\_path: str):

"""

Take saved toy Keras model, load it and use it to predict the named entities in a file of text.

The file can be any text - it doesn't need to be a debate file.

:param file\_path: path to a text file to predict

:return:

"""

with open(file\_path) as f:

file\_contents = f.read()

config = Config()

dataset = CharBasedNERDataset()

lm = LoadedToyModel(config=config, dataset=dataset)

return lm.predict\_long\_str(file\_contents)

def model\_toy\_predict\_str(string: str):

"""

Take saved toy Keras model, load it and use it to predict the named entities in a file of text.

The file can be any text - it doesn't need to be a debate file.

:param string: The string to predict NEs for

:return:

"""

config = Config()

dataset = CharBasedNERDataset()

lm = LoadedToyModel(config=config, dataset=dataset)

return lm.predict\_long\_str(string)

def model\_mini\_predict\_file(file\_path: str):

"""

Take saved mini Keras model, load it and use it to predict the named entities in a file of text.

The file can be any text - it doesn't need to be a debate file.

:param file\_path: path to a text file to predict

:return:

"""

with open(file\_path) as f:

file\_contents = f.read()

config = Config()

dataset = CharBasedNERDataset()

lm = LoadedMiniModel(config=config, dataset=dataset)

return lm.predict\_long\_str(file\_contents)

## keras\_character\_based\_ner/src/matt/train.py

from keras\_character\_based\_ner.src.config import Config

from keras\_character\_based\_ner.src.dataset import CharBasedNERDataset

from keras\_character\_based\_ner.src.matt.model\_integration import get\_x\_y as matt\_get\_x\_y

from keras\_character\_based\_ner.src.matt.file\_management import pickle\_large\_file

from keras\_character\_based\_ner.src.matt.persist import LoadedToyModel, SavedCharacterBasedLSTMModel

class ToyConfig(Config):

"""

Override Config with something suitable for toy testing - i.e. only a few epochs

"""

max\_epochs = 8

class ToyCharBasedNERDataset(CharBasedNERDataset):

def get\_x\_y(self, sentence\_maxlen, dataset\_name='all'):

"""

Override super-class definition in CharBasedNERDataset so we use toy data, not mini data

:param self:

:param sentence\_maxlen:

:param dataset\_name:

:return:

"""

return matt\_get\_x\_y(dataset\_name, "toy")

def toy\_dataset\_fit():

print("Fitting toy dataset")

config = ToyConfig()

dataset = ToyCharBasedNERDataset()

model = SavedCharacterBasedLSTMModel(config, dataset)

history = model.fit()

history\_dict = history.history

model.evaluate()

print(model.predict\_str('My name is Margaret Thatcher, and I greatly enjoy shopping at Tesco when I am in Birmingham!'))

model.save("keras\_character\_based\_ner/src/toy\_dataset.keras.h5")

pickle\_large\_file(history\_dict, "keras\_character\_based\_ner/src/toy\_dataset.history.p")

def toy\_dataset\_refit():

"""

Continue training on toy dataset after loading in from disk

:return:

"""

config = ToyConfig()

dataset = ToyCharBasedNERDataset()

model = LoadedToyModel(config, dataset)

history = model.fit()

history\_dict = history.history

model.evaluate()

print(model.predict\_str('My name is Margaret Thatcher, and I greatly enjoy shopping at Tesco when I am in Birmingham!'))

model.save("keras\_character\_based\_ner/src/toy\_dataset.keras.h5")

pickle\_large\_file(history\_dict, "keras\_character\_based\_ner/src/toy\_dataset.history.p")

def mini\_dataset\_fit():

class MiniConfig(Config):

"""

Override Config with something suitable for quick testing - i.e. only a few epochs

"""

max\_epochs = 2

config = MiniConfig()

class MiniCharBasedNERDataset(CharBasedNERDataset):

def get\_x\_y(self, sentence\_maxlen, dataset\_name='all'):

"""

Override super-class definition in CharBasedNERDataset so we use mini data, not toy data.

:param self:

:param sentence\_maxlen:

:param dataset\_name:

:return:

"""

return matt\_get\_x\_y(dataset\_name, "mini")

dataset = MiniCharBasedNERDataset()

model = SavedCharacterBasedLSTMModel(config, dataset)

history = model.fit()

history\_dict = history.history

model.evaluate()

print(model.predict\_str('My name is Margaret Thatcher, and I greatly enjoy shopping at Tesco when I am in Birmingham!'))

model.save("keras\_character\_based\_ner/src/mini\_dataset.keras.h5")

pickle\_large\_file(history\_dict, "keras\_character\_based\_ner/src/mini\_dataset.history.p")

def full\_dataset\_fit\_generator():

print("Fitting full dataset using generator")

config = Config()

dataset = CharBasedNERDataset()

model = SavedCharacterBasedLSTMModel(config, dataset)

history = model.fit\_generator()

history\_dict = history.history

model.evaluate\_generator()

print(model.predict\_str('My name is Margaret Thatcher, and I greatly enjoy shopping at Tesco when I am in Birmingham!'))

model.save("keras\_character\_based\_ner/src/full\_dataset.keras.h5")

pickle\_large\_file(history\_dict, "keras\_character\_based\_ner/src/full\_dataset.history.p")

## ne\_data\_gathering/companies.py

#!/usr/bin/env python

from ftplib import FTP

from typing import Any, List, Dict, Generator

from ne\_data\_gathering.util import capitalise\_text\_list, write\_to\_data\_file

from ne\_data\_gathering import util

import os

import csv

import requests

# FTP companies data is too dirty to use :(

def nasdaq():

nasdaq\_csv\_companies = dedup(process\_nasdaq\_csv())

write\_to\_data\_file(nasdaq\_csv\_companies, "companies", "nasdaq\_csv\_companies.txt")

def lse():

lse\_data = process\_lse\_download()

write\_to\_data\_file(lse\_data, "companies", "lse\_manual\_download.txt")

def dbpedia(src\_dir, file\_path):

dbpedia\_sparql\_extract\_companies("{}{}".format(src\_dir, file\_path))

util.dbpedia\_post\_processing(

"{}{}".format(src\_dir, file\_path), "processed\_ne\_data{}".format(file\_path))

def conll2003eng():

conll\_companies = util.process\_conll\_file(util.conll\_file, 'ORG')

util.write\_to\_data\_file(conll\_companies, "companies", "conll\_2003.txt")

def download\_and\_process(src\_dir, file\_path) -> None:

nasdaq()

lse()

dbpedia(src\_dir, file\_path)

conll2003eng()

def download\_nasdaq(data\_files: List[str]) -> List[str]:

class Reader:

def \_\_init\_\_(self):

self.data = ""

def \_\_call\_\_(self, bytes\_data):

self.data += bytes\_data.decode('utf-8')

conn = FTP('ftp.nasdaqtrader.com')

conn.login()

conn.cwd('SymbolDirectory')

r = Reader()

for f in data\_files:

conn.retrbinary("RETR {}".format(f), r)

return r.data.split("\n")

def filter\_names(company\_data: List[str]) -> List[str]:

company\_names = [d.split("|")[1] for d in company\_data if len(d.split("|")) > 1]

return list(filter(lambda company\_name: company\_name != "", company\_names))

def process\_nasdaq\_ftp():

data\_files = ["nasdaqlisted.txt", "otherlisted.txt"]

company\_data = download\_nasdaq(data\_files)

write\_to\_data\_file(filter\_names(company\_data), "companies", "nasdaq\_ftp\_companies.txt")

def dedup(data: List[str]) -> List[str]:

return list(set(data))

def process\_nasdaq\_csv() -> Generator[str, None, None]:

nasdaq\_exchanges = "AMEX NASDAQ NYSE".split()

for exchange in nasdaq\_exchanges:

csv\_url = "https://www.nasdaq.com/screening/companies-by-industry.aspx?exchange={}&render=download"\

.format(exchange)

r = requests.get(csv\_url)

processed\_text = r.text.replace("\r\n", "\n").replace("&#39;", "'")

csv\_data = processed\_text.split("\n")

reader = csv.reader(csv\_data)

for row in reader:

if len(row) > 1:

yield(row[1])

def dbpedia\_sparql\_get\_company\_count() -> int:

sparql\_query = """

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX dbo: <http://dbpedia.org/ontology/>

PREFIX dbp: <http://dbpedia.org/property/>

SELECT COUNT(\*)

WHERE { ?resource foaf:name ?name .

?resource rdf:type dbo:Organisation .

}

"""

res = util.dbpedia\_do\_sparql\_query(sparql\_query)

return int(res['results']['bindings'][0]['callret-0']['value'])

def dbpedia\_sparql\_extract\_companies(company\_list\_file):

# With help from https://rdflib.github.io/sparqlwrapper/

# and https://stackoverflow.com/questions/38332857/

# sparql-query-to-get-all-person-available-in-dbpedia-is-showing-only-some-person

if os.path.exists(company\_list\_file):

os.unlink(company\_list\_file)

total = dbpedia\_sparql\_get\_company\_count()

for i in range(0, total, 10000):

result\_list = []

offset = str(i)

print("We're at {sofar} out of {total}".format(sofar=offset, total=total))

sparql\_query = """

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX dbo: <http://dbpedia.org/ontology/>

PREFIX dbp: <http://dbpedia.org/property/>

SELECT ?name

WHERE { ?resource foaf:name ?name .

?resource rdf:type dbo:Organisation .

}

"""

sparql\_query\_offset = "LIMIT 10000 OFFSET {}".format(offset)

response = util.dbpedia\_do\_sparql\_query(sparql\_query + sparql\_query\_offset)

results = response['results']['bindings']

result\_list.extend([res['name']['value'] for res in results])

print("Adding {count} to companies list file".format(count=len(results)))

with open(company\_list\_file, 'a') as f:

f.writelines("\n".join(result\_list))

def process\_lse\_download() -> List[str]:

# manually downloaded on 3rd March 2018 from

# http://www.londonstockexchange.com/statistics/companies-and-issuers/companies-defined-by-mifir-identifiers-list-on-lse.xlsx

# Pandas cannot cope with this xlsx :(

with open('raw\_ne\_data/lse\_manual\_download.txt') as f:

\_lse = f.readlines()

return capitalise\_text\_list(\_lse)

## ne\_data\_gathering/people.py

import csv

from typing import List, Generator

import os

import sys

from ne\_data\_gathering import util

def nyc():

nyc\_baby\_names = sorted(set(process\_kaggle\_nyc\_baby\_names()))

util.write\_to\_data\_file(nyc\_baby\_names, "people", "nyc\_baby\_names.txt")

def dbpedia\_post\_processing(src\_dir, file\_path):

util.dbpedia\_post\_processing(

"{}{}".format(src\_dir, file\_path), "processed\_ne\_data{}".format(file\_path))

def dbpedia(src\_dir, file\_path):

dbpedia\_sparql\_extract\_people("{}{}".format(src\_dir, file\_path))

util.dbpedia\_post\_processing(

"{}{}".format("raw\_ne\_data", file\_path), "{}{}".format("processed\_ne\_data", file\_path))

def conll2003eng():

conll\_people = util.process\_conll\_file(util.conll\_file, 'PER')

util.write\_to\_data\_file(conll\_people, "people", "conll\_2003.txt")

def download\_and\_process(src\_dir, file\_path):

nyc()

dbpedia(src\_dir, file\_path)

conll2003eng()

def process\_kaggle\_nyc\_baby\_names() -> Generator[str, None, None]:

with open('raw\_ne\_data/Most\_Popular\_Baby\_Names\_by\_Sex\_and\_Mother\_s\_Ethnic\_Group\_\_New\_York\_City.csv') as f:

data = f.readlines()

for row in csv.reader(data):

yield row[3].capitalize()

def dbpedia\_sparql\_get\_people\_count() -> int:

sparql\_query = """

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX dbo: <http://dbpedia.org/ontology/>

PREFIX dbp: <http://dbpedia.org/property/>

SELECT COUNT(\*)

WHERE { ?resource foaf:name ?name .

?resource rdf:type dbo:Person .

}

"""

res = util.dbpedia\_do\_sparql\_query(sparql\_query)

return int(res['results']['bindings'][0]['callret-0']['value'])

def dbpedia\_sparql\_extract\_people(people\_list\_file):

# With help from https://rdflib.github.io/sparqlwrapper/

# and https://stackoverflow.com/questions/38332857/

# sparql-query-to-get-all-person-available-in-dbpedia-is-showing-only-some-person

if os.path.exists(people\_list\_file):

os.unlink(people\_list\_file)

# total\_people = dbpedia\_sparql\_get\_people\_count()

total\_people = 2109301

for i in range(0, total\_people, 10000):

people\_list = []

offset = str(i)

print("We're at {sofar} out of {total}".format(sofar=offset, total=total\_people))

sparql\_query = """

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX dbo: <http://dbpedia.org/ontology/>

PREFIX dbp: <http://dbpedia.org/property/>

SELECT ?name

WHERE { ?resource foaf:name ?name .

?resource rdf:type dbo:Person .

}

"""

sparql\_query\_offset = "LIMIT 10000 OFFSET {}".format(offset)

response = util.dbpedia\_do\_sparql\_query(sparql\_query + sparql\_query\_offset)

results = response['results']['bindings']

people\_list.extend([res['name']['value'] for res in results])

print("Adding {count} to people list file".format(count=len(results)))

with open(people\_list\_file, 'a') as f:

f.writelines("\n".join(people\_list))

## ne\_data\_gathering/places.py

#!/usr/bin/env python

import os

import sys

from ne\_data\_gathering import util

def dbpedia(src\_dir, file\_path):

dbpedia\_sparql\_extract\_places("{}{}".format(src\_dir, file\_path))

util.dbpedia\_post\_processing(

"{}{}".format("raw\_ne\_data", file\_path), "{}{}".format("processed\_ne\_data", file\_path))

def conll2003eng():

conll\_places = util.process\_conll\_file(util.conll\_file, 'LOC')

util.write\_to\_data\_file(conll\_places, "places", "conll\_2003.txt")

def download\_and\_process(src\_dir, file\_path) -> None:

dbpedia(src\_dir, file\_path)

conll2003eng()

def dbpedia\_sparql\_get\_place\_count() -> int:

sparql\_query = """

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX dbo: <http://dbpedia.org/ontology/>

PREFIX dbp: <http://dbpedia.org/property/>

SELECT COUNT(\*)

WHERE { ?resource foaf:name ?name .

?resource rdf:type dbo:Place .

}

"""

res = util.dbpedia\_do\_sparql\_query(sparql\_query)

return int(res['results']['bindings'][0]['callret-0']['value'])

def dbpedia\_sparql\_extract\_places(list\_file):

# With help from https://rdflib.github.io/sparqlwrapper/

# and https://stackoverflow.com/questions/38332857/

if os.path.exists(list\_file):

os.unlink(list\_file)

total = dbpedia\_sparql\_get\_place\_count()

for i in range(0, total, 10000):

result\_list = []

offset = str(i)

print("We're at {sofar} out of {total}".format(sofar=offset, total=total))

sparql\_query = """

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX dbo: <http://dbpedia.org/ontology/>

PREFIX dbp: <http://dbpedia.org/property/>

SELECT ?name

WHERE { ?resource foaf:name ?name .

?resource rdf:type dbo:Place .

}

"""

sparql\_query\_offset = "LIMIT 10000 OFFSET {}".format(offset)

response = util.dbpedia\_do\_sparql\_query(sparql\_query + sparql\_query\_offset)

results = response['results']['bindings']

result\_list.extend([res['name']['value'] for res in results])

print("Adding {count} to places list file".format(count=len(results)))

with open(list\_file, 'a') as f:

f.writelines("\n".join(result\_list))

## ne\_data\_gathering/util.py

from typing import List, Set, Generator, Dict, Any

from os.path import realpath, dirname

from nltk.corpus import stopwords # type: ignore

from nltk.tokenize import word\_tokenize # type: ignore

import os

import re

conll\_file = 'raw\_ne\_data/eng.list'

def capitalise\_text\_list(l: List[str]) -> List[str]:

new\_list = []

def capitalise(text: str):

return text[:1].upper() + text[1:].lower()

for elem in l:

new\_elem = " ".join([capitalise(word) for word in elem.split()])

new\_list.append(new\_elem)

return new\_list

def write\_to\_data\_file(data: List[str], category: str, file\_name: str) -> None:

file\_path = realpath(\_\_file\_\_)

data\_path = "{}/processed\_ne\_data/{}/{}".format(dirname(file\_path), category, file\_name)

os.makedirs(dirname(data\_path), exist\_ok=True)

with open(data\_path, "w+") as f:

f.write("\n".join(data))

f.write("\n")

def dbpedia\_do\_sparql\_query(sparql\_query: str) -> Dict[Any, Any]:

from SPARQLWrapper import SPARQLWrapper, JSON # type: ignore

sparql = SPARQLWrapper("http://dbpedia.org/sparql")

sparql.setQuery(sparql\_query)

sparql.setReturnFormat(JSON)

results = sparql.query().convert()

return results

def process\_conll\_file(filepath, tag) -> Generator[str, None, None]:

with open(filepath) as f:

lines = f.readlines()

for line in lines:

contents = line.split(" ")

if contents[0] == tag:

yield " ".join(contents[1:]).rstrip()

def surrounded\_by\_chars(\_line: str, start\_char, end\_char=None) -> bool:

if end\_char is None:

end\_char = start\_char

return \_line.startswith(start\_char) and \_line.rstrip().endswith(end\_char)

def remove\_outer\_brackets(\_line: str) -> str:

return \_line[1:-2] + \_line[-1]

def all\_stop\_words(line, stop\_words: Set[str]) -> bool:

line\_words = word\_tokenize(line)

if all(word in stop\_words for word in line\_words):

return True

else:

return False

def dbpedia\_post\_processing(src\_list\_file, dest\_list\_file):

src\_list\_file = "ne\_data\_gathering/{}".format(src\_list\_file)

stop\_words = set(stopwords.words('english'))

debug = False

res\_lines = []

processed\_list\_file = "ne\_data\_gathering/{}".format(dest\_list\_file)

with open(src\_list\_file, 'r+', encoding='utf-8') as f:

lines = sorted(set(f.readlines()))

for line in lines:

# Remove double quotes

line = line.replace('"', '')

# Left-trim any whitespace

line = line.lstrip()

# Get rid of lines that are entirely numbers or symbols

if re.match("""^[!@£$%^&\*()0-9 ]+$""", line):

print("DEBUG: Removing symbol lines {}".format(line)) if debug else None

continue

# If whole line is surrounded by brackets, remove those brackets

if line.startswith("(") and line.endswith(")\n"):

print("DEBUG: found bracketed line: {}".format(line)) if debug else None

line = line[1:-2] + "\n"

# If line starts with more than one single quote, remove all the single quotes at start

match = re.match("""^'{2,}(.\*)""", line)

if match is not None:

print("DEBUG: remove extraneous prefixed single quotes in line {}".format(line)) if debug\

else None

line = match.group(1)

# If line ends with more than one single quote, remove all the single quotes at start

match = re.match("""(.\*)'{2,}$""", line)

if match is not None:

print("DEBUG: remove extraneous suffixed single quotes in line {}".format(line)) if debug \

else None

line = match.group(1) + "\n"

# If line starts with just whitespace and/or asterisks, remove them

match = re.match("""^[\* ]+(.\*)""", line)

if match is not None:

print("DEBUG: remove extraneous prefixed spaces/asterisks in line {}".format(line)) \

if debug else None

line = match.group(1)

# Remove words shorter than 4 chars (they all have final newline)

# These tend to be strange stub words like 'ar' which are low-value and hard to filter.

if len(line) < 5:

print("DEBUG: Removing short line {}".format(line)) if debug else None

continue

# If all words in the line are stop words, remove the line

if all\_stop\_words(line, stop\_words):

continue

res\_lines.append(line)

with open(processed\_list\_file, 'w+') as f:

f.writelines(res\_lines)

## simple\_gui/simple\_gui.py

from typing import List, Tuple

from flask import Flask, render\_template, request

from hansard\_gathering import filesystem

from keras\_character\_based\_ner.src.matt.persist import LoadedToyModel

from keras\_character\_based\_ner.src.matt.eval import init\_config\_dataset

from simple\_gui.util import format\_prediction\_string

import tensorflow as tf

app = Flask(\_\_name\_\_)

cache = {}

# Tensorflow default graph has to be captured to avoid a TF threading bug

# when running with Flask:

# https://github.com/keras-team/keras/issues/2397

graph = None

def initialize\_keras\_model():

global graph

cache["model"] = LoadedToyModel(\*init\_config\_dataset())

# Set Tensorflow graph as soon as model is set

graph = tf.get\_default\_graph()

@app.route('/')

def get\_dates\_list():

dates = filesystem.get\_dates\_list()

return render\_template('index.html', dates=dates)

@app.route('/date/<date>/')

def get\_hansards\_by\_date(date):

debates: List[Tuple[int, str]] = list(filesystem.get\_debates\_by\_date(date))

return render\_template('date.html', date=date, debates=debates)

@app.route('/date/<date>/<debate\_title>')

def view\_hansard(date, debate\_title):

debate = filesystem.view\_hansard(date, debate\_title)

debate\_paras = debate.split("\n")

return render\_template('debate.html', date=date, debate\_title=debate\_title, debate\_paras=debate\_paras)

# Add a 'predict' route for AJAX posting of content to be predicted

@app.route('/predict/', methods=['POST'])

def predict\_text():

global graph

with graph.as\_default():

model = cache['model']

data: str = request.get\_data().decode(encoding='UTF-8')

prediction: List[Tuple[str]] = model.predict\_long\_str(data)

gui\_prediction: str = format\_prediction\_string(prediction)

return gui\_prediction

def main():

initialize\_keras\_model()

app.run(load\_dotenv=False, debug=True, port=5000, threaded=True)

## simple\_gui/util.py

from typing import List, Tuple

def format\_prediction\_string(prediction: List[Tuple[str]]) -> str:

"""

Take a prediction string returned by the Keras model.

Make it nice to print in HTML, so it clearly indicates different NE types to a user.

:param prediction: a list of tuples of strings which are characters of the text, zipped up with

their NE prediction, e.g. [('A', 'LOC')], or '0' for the null label

:return: a nicely formatted NE type for user to use. V1: ++ for loc, \*\* for org, \_\_ for person

"""

label\_start\_chars = {

"0": "",

"LOC": "<loc>",

"ORG": "<org>",

"PER": "<per>",

}

label\_end\_chars = {

"0": "",

"LOC": "</loc>",

"ORG": "</org>",

"PER": "</per>",

}

result: List[str] = []

previous\_label\_state: str = "0"

for char, label in prediction:

# label-start

if previous\_label\_state == "0" and label != "0":

result.append(label\_start\_chars[label])

result.append(char)

# label-end

elif previous\_label\_state != "0" and label == "0":

result.append(label\_end\_chars[previous\_label\_state])

result.append(char)

# label-continue

elif label == previous\_label\_state:

result.append(char)

else:

raise RuntimeError("Unexpected state")

previous\_label\_state = label

return "".join(result)

## simple\_gui/static/char-ner.js

console.log("char-ner javascript loaded");

$( ".unrendered" ).click(function(event) {

const $elem = $( this );

const text = $elem.text();

$elem.effect("bounce", "slow");

$.post("http://localhost:5000/predict/", text, function(data) {

$elem.text(data);

$elem.removeClass("unrendered");

$elem.addClass("rendered");

$elem.unbind("click");

})

});

## simple\_gui/static/style.css

body {

background-color: #ffffff;

}

h1 {

color: #fff;

font-family: Arial, Helvetica, sans-serif;

}

.rendered {

color: blue;

}

## simple\_gui/templates/date.html

<html>

<head>

Debates for date {{ date }}. <br/>

<a href="{{ url\_for('get\_dates\_list') }}"> Up one level</a>

</head>

<ul>

{% for debate\_idx, debate\_title in debates %}

<li>

<a href="{{ url\_for('view\_hansard', date=date, debate\_title=debate\_title) }}">

{{ debate\_idx + 1 }} - {{ debate\_title }}

</a>

</li>

{% endfor %}

</ul>

</html>

## simple\_gui/templates/debate.html

<html>

<head>

This is the debate from date {{ date }} titled {{ debate\_title }}<br/>

<a href="{{ url\_for('get\_hansards\_by\_date', date=date) }}"> Up one level.</a> <br/><br/>

<link rel="stylesheet" href="{{ url\_for('static', filename='style.css') }}">

</head>

<body>

{% for para in debate\_paras %}

<p class="unrendered">{{ para }}</p>

{% endfor %}

</body>

<!-- Load JQuery - http://flask.pocoo.org/docs/1.0/patterns/jquery/ -->

<script src="//ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>

<script>window.jQuery || document.write(

'<script src="{{ url\_for('static', filename='jquery.js') }}">\x3C/script>')</script>

<script src="{{ url\_for('static', filename='char-ner.js') }}"></script>

<script src="{{ url\_for('static', filename='jquery-ui.min.js') }}"></script>

</html>

## simple\_gui/templates/index.html

<html>

<head>

List of dates from which to select a Hansard debate.

</head>

<ul>

{% for date in dates %}

<li><a href="{{ url\_for('get\_hansards\_by\_date', date=date) }}">{{ date }}</a></li>

{% endfor %}

</ul>

</html>

## test/test\_chunk.py

from hansard\_gathering.chunk import chunk\_hansard\_debate\_file\_nltk

from hansard\_gathering.chunk import nltk\_get\_tokenizer, get\_sentence\_spans

import os

tokenizer = nltk\_get\_tokenizer()

def get\_contents(fake\_path: str) -> str:

with open(fake\_path, "r") as f:

contents = f.read()

return contents

def test\_chunk\_hansard\_debate\_file\_nltk(fs):

fake\_path = "/a/b/longfile.txt"

fake\_spans\_path = "/a/b/longfile-spans.txt"

contents = "There once was a happy dog. He grew to an old age. The end."

fs.create\_file(fake\_path, contents=contents)

chunk\_hansard\_debate\_file\_nltk(fake\_path, tokenizer)

expected\_spans = "(0,27)\n" +\

"(28,50)\n" +\

"(51,59)"

assert get\_contents(fake\_spans\_path) == expected\_spans

os.unlink(fake\_path)

contents2 = "My friend the hon. Gentleman will surely agree. This must end now."

fs.create\_file(fake\_path, contents=contents2)

chunk\_hansard\_debate\_file\_nltk(fake\_path, tokenizer)

expected\_spans2 = "(0,47)\n" +\

"(48,66)"

assert get\_contents(fake\_spans\_path) == expected\_spans2

def test\_get\_sentence\_spans(fs):

spans: str = "(0,27)\n" + \

"(28,50)\n" + \

"(51,59)"

fake\_spans\_path: str = "/fake/sent-spans.txt"

fake\_debate\_path: str = "/fake/sent.txt"

fs.create\_file(fake\_spans\_path, contents=spans)

generator = get\_sentence\_spans(fake\_debate\_path)

expected\_list = [(0, 27), (28, 50), (51, 59)]

assert list(generator) == expected\_list

## test/test\_companies.py

from ne\_data\_gathering.companies import capitalise\_text\_list

def test\_capitalise\_text():

data = ["A LIST OF", "VARIOUS SHOUTY", "STRINGS"]

expected = ["A List Of", "Various Shouty", "Strings"]

assert(list(capitalise\_text\_list(data)) == expected)

## test/test\_interpolate.py

from typing import Set

from nltk.tokenize import TreebankWordTokenizer # type: ignore

from hansard\_gathering.interpolate import interpolate\_one

from hansard\_gathering.interpolate import ngram\_span\_search\_named\_entities, overlaps

def test\_interpolate\_one(fs):

all\_places: Set[str] = {"London", "New York", "Las Vegas"}

all\_people: Set[str] = {"Margaret Thatcher", "Ernest Hemingway"}

all\_companies: Set[str] = ["Sainsburys", "Tescos", "The White House"]

tokenizer = TreebankWordTokenizer()

file\_contents = "I do recall that Margaret Thatcher was good at finding Sainsburys in London"

file\_path = './hansard\_gathering/processed\_hansard\_data/1976-02-09/Abortion (Amendment) Bill' \

+ ' (Select Committee).txt'

interpolated\_file\_path = './hansard\_gathering/interpolated\_hansard\_data/1976-02-09/Abortion ' \

+ '(Amendment) Bill (Select Committee).txt'

fs.create\_file(file\_path, contents=file\_contents)

fs.create\_file(interpolated\_file\_path, contents="")

interpolate\_one(file\_path, tokenizer, "processed", all\_places, all\_companies, all\_people)

with open(interpolated\_file\_path) as f:

contents = f.read()

assert contents == "000000000000000003333333333333333300000000000000000000022222222220000111111"

def test\_ngram\_span\_search\_named\_entities():

span\_window = ((20, 33), (34, 42), (43, 47), (48, 53))

text = 'I have searched the International Monetary Fund rules, and I cannot find under which rule this is done.'

all\_places = {"Qatar"}

all\_companies = {"Sainsburys", "International Monetary Fund"}

all\_people = {"Ed Milliband"}

result = ngram\_span\_search\_named\_entities(span\_window, text, all\_places, all\_companies, all\_people)

expected\_result = 20, 47, 2

assert result == expected\_result

def test\_overlaps():

assert overlaps(((98, 102), (102, 103), None, None), 101)

assert not overlaps(((98, 102), (102, 103), None, None), 97)

## test/test\_matt.py

from keras\_character\_based\_ner.src.matt.file\_management import file\_lines

def test\_file\_lines(fs):

file\_contents = """One line

Another line

A Third Line""" # no final newline, just like in our span files

fs.create\_file("/var/data/a.txt", contents=file\_contents)

result = file\_lines("/var/data/a.txt")

expected = 3

assert result == expected

## test/test\_model\_integration.py

from keras\_character\_based\_ner.src.matt.model\_integration import onehot

def test\_onehot():

result = onehot(4, 7)

expected = [0, 0, 0, 0, 1, 0, 0]

assert result == expected

## test/test\_preprocessing.py

from hansard\_gathering.preprocessing import unxml\_hansard\_document

def test\_unxml\_hansard\_document():

text = """

<?xml version="1.0" encoding="ISO-8859-1"?>

<publicwhip scrapeversion="a" latest="yes">

<major-heading id="uk.org.publicwhip/debate/1940-03-20a.1953.0" colnum="1953">Preamble</major-heading>

<speech id="uk.org.publicwhip/debate/1940-03-20a.1953.1" colnum="1953" time="">

<p><i>The House met at a Quarter before Three of the Clock</i>, Mr. SPEAKER <i>in the Chair</i>.</p

"""

result = unxml\_hansard\_document(text)

expected = b'\n Preamble\n \n The House met at a Quarter before Three of ' \

b'the Clock, Mr. SPEAKER in the Chair.'

assert result == expected

## test/test\_simple\_gui\_util.py

from simple\_gui.util import format\_prediction\_string # type: ignore

def test\_format\_prediction\_string():

# Nonsense sentence, "A Hull Shell Emmma"

zipped\_data = [('A', '0'),

(' ', '0'),

('H', 'LOC'), ('u', 'LOC'), ('l', 'LOC'), ('l', 'LOC'),

(' ', '0'),

('S', 'ORG'), ('h', 'ORG'), ('e', 'ORG'), ('l', 'ORG'), ('l', 'ORG'),

(' ', '0'),

('E', 'PER'), ('m', 'PER'), ('m', 'PER'), ('a', 'PER'), ('.', '0')]

# V2 basic tags

expected\_result = "A <loc>Hull</loc> <org>Shell</org> <per>Emma</per>."

actual\_result = format\_prediction\_string(zipped\_data)

assert expected\_result == actual\_result

## test/test\_util.py

from ne\_data\_gathering.util import all\_stop\_words

from ne\_data\_gathering.util import surrounded\_by\_chars

from nltk.corpus import stopwords # type: ignore

def test\_surrounded\_by\_chars():

s = "(a nicely bracketed string)\n"

s2 = "(a nicely bracketed string with no newline)"

assert(surrounded\_by\_chars(s, "(", ")"))

assert(surrounded\_by\_chars(s2, "(", ")"))

def test\_all\_stop\_words\_true():

stop\_words = set(stopwords.words('english'))

line = "and the of in"

result = all\_stop\_words(line, stop\_words)

expected = True

assert result == expected

def test\_all\_stop\_words\_false():

stop\_words = set(stopwords.words('english'))

line = "and the of in Canary Beelzebub"

result = all\_stop\_words(line, stop\_words)

expected = False

assert result == expected

1. <https://github.com/0xnurl/keras_character_based_ner> [↑](#footnote-ref-1)
2. <https://hansard.parliament.uk/> [↑](#footnote-ref-2)
3. <http://www.data.parliament.uk/dataset/12> and <http://api.data.parliament.uk/> [↑](#footnote-ref-3)
4. <https://www.theyworkforyou.com/api/> [↑](#footnote-ref-4)
5. <https://dbpedia.org/sparql> [↑](#footnote-ref-5)
6. <https://rdflib.github.io/sparqlwrapper/> [↑](#footnote-ref-6)
7. Unfortunately, since version 2 Keras does not calculate precision, recall or F1 by default, so 0xnurl’s model does not include these metrics. My comparison here of accuracy from my project to F1 in (Kuru, Arkan Can and Deniz, 2016) is based on the assumption that accuracy is always a more ‘generous’ metric, as it does not explicitly penalise for lack of recall. Even my project’s *accuracy* score is lower than Kuru, Can and Deniz’s F1 score, so my model’s F1 score would doubtless be lower still. [↑](#footnote-ref-7)
8. Difference of 0.9594 and 0.9035, see Table 7 [↑](#footnote-ref-8)
9. Difference of 0.05775 and 0.1564, see Table 7 [↑](#footnote-ref-9)
10. At least, all Hansard debates available via the TheyWorkForYou API, which goes back to the year 1919. [↑](#footnote-ref-10)
11. <http://www.pyinvoke.org/> [↑](#footnote-ref-11)
12. <https://readthedocs.org/projects/argh/> [↑](#footnote-ref-12)
13. <https://github.com/seomoz/shovel> [↑](#footnote-ref-13)
14. <http://pydoit.org/> [↑](#footnote-ref-14)
15. <https://www.kaggle.com/new-york-city/nyc-baby-names> [↑](#footnote-ref-15)
16. <https://github.com/mozilla/bleach> [↑](#footnote-ref-16)
17. <https://lxml.de/api/lxml.etree-module.html> [↑](#footnote-ref-17)
18. <http://www.unicode.org/versions/Unicode11.0.0/> [↑](#footnote-ref-18)
19. <https://github.com/nltk/nltk/issues/2076> [↑](#footnote-ref-19)
20. <https://github.com/nltk/nltk/pull/1864> [↑](#footnote-ref-20)
21. 8 Epochs were planned for all the ‘Toy’ datasets, but Keras’ EarlyStopping module stopped the training after either 6 or 7 epochs as the validation loss on the dev dataset stopped improving [↑](#footnote-ref-21)
22. As ToyV1 gave Not A Number results when evaluation on its dataset, it was also evaluated on the ‘mini’ dataset, which is just the first 4000 results from ToyV1, in order to get some sort of representative evaluation data. [↑](#footnote-ref-22)
23. See Table 5 for a description of this dataset. [↑](#footnote-ref-23)
24. <https://github.com/jmcgeheeiv/pyfakefs> [↑](#footnote-ref-24)
25. <http://jinja.pocoo.org/> [↑](#footnote-ref-25)
26. <http://werkzeug.pocoo.org/> [↑](#footnote-ref-26)
27. <https://jqueryui.com/> [↑](#footnote-ref-27)
28. It is also interesting that 0xnurl’s model uses an embedding as the first layer of the model, whereas the paper on which the model is based, (Kuru, Arkan Can and Deniz, 2016), uses one-hot as the input layer. This could also be tried, to see if it affects the evaluation results. [↑](#footnote-ref-28)