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
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. Petar Konovski and Phil Gregg'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.

I give thanks for everyone who prayed for me as I finished this thesis, in particular Rev'd Vanessa Conant and the staff of the Parish of Walthamstow. Finally, I would like to thank my wife Lydia 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

Tables

Code snippets

1. Table of Contents

2. 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] The target dataset was the Hansard, the record of debates in both of the houses of Parliament in the United Kingdom.[2] This dataset is now available via the Parliament UK Data API,[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] which has all debates from 1919 onwards in the House of Commons 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 5.7. 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] and Python's SPARQLWrapper library[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 6.1.
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 3.

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.

---

[1] https://github.com/0xnurl/keras_character_based_ner
[2] https://hansard.parliament.uk/
[3] http://www.data.parliament.uk/dataset/12 and http://api.data.parliament.uk/
[4] https://www.theyworkforyou.com/api/
[5] https://dbpedia.org/sparql
[6] https://rdflib.github.io/sparqlwrapper/

3. Overall Results

The sequence of models described in section 7.1 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] 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 7.1.1 (section 7.1 contains a full description of all the trained models). My model beats the baseline by 5.6% for categorical accuracy,[8] and by 9.7% for non-null label accuracy.[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] 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.

[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.

[8] Difference of 0.9594 and 0.9035, see Table 8

[9] Difference of 0.05775 and 0.1564, see Table 8

[10] At least, all Hansard debates available via the TheyWorkForYou API, which goes back to the year 1919 and covers the House of Commons.

4. 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 1. The original 'risks and mitigations' column has been removed as it is not relevant here.

Table 1 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 1.

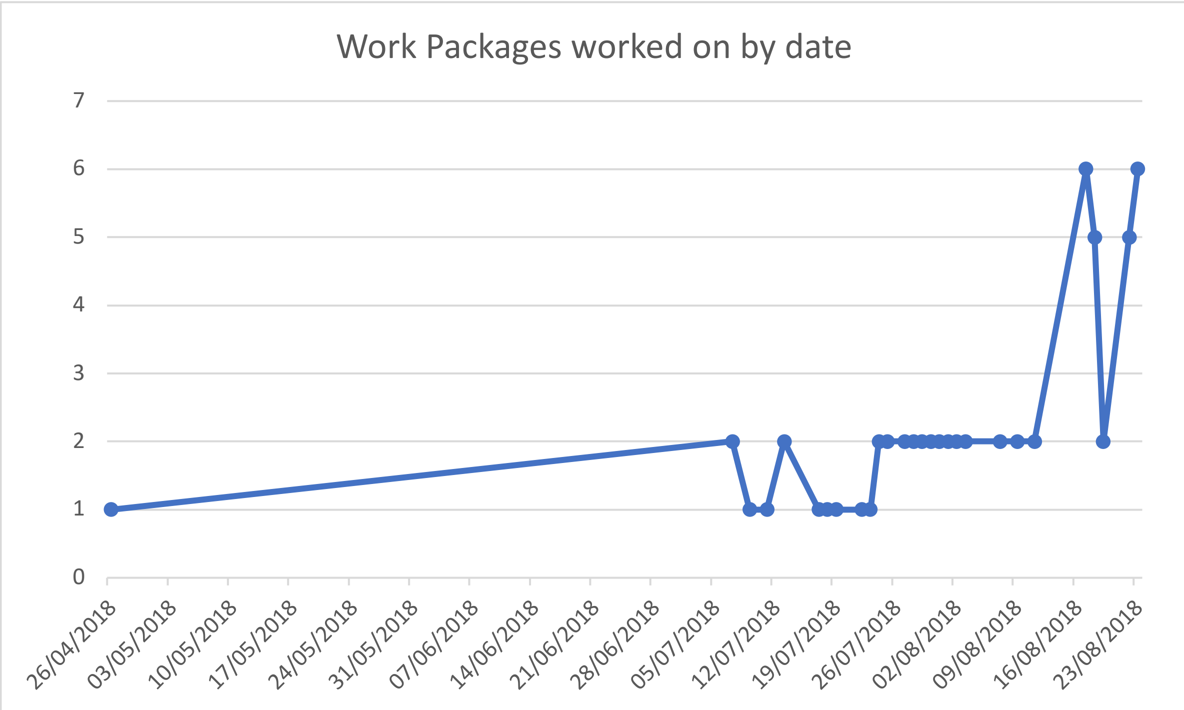

Figure 1 Actual time spent on each work package of project

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 wired into the machine learning model, 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.

5. Software Architecture

5.1. 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 2). 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 6.

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 7.2.

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.

5.2. 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] was chosen, after some experimentation with Argh,[12] Shovel[13] and Doit.[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

[11] http://www.pyinvoke.org/
[12] https://readthedocs.org/projects/argh/
[13] https://github.com/seomoz/shovel
[14] http://pydoit.org/

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 14) 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 12.

Step numbers in the sections below refer to the blue numbers in Figure 2.

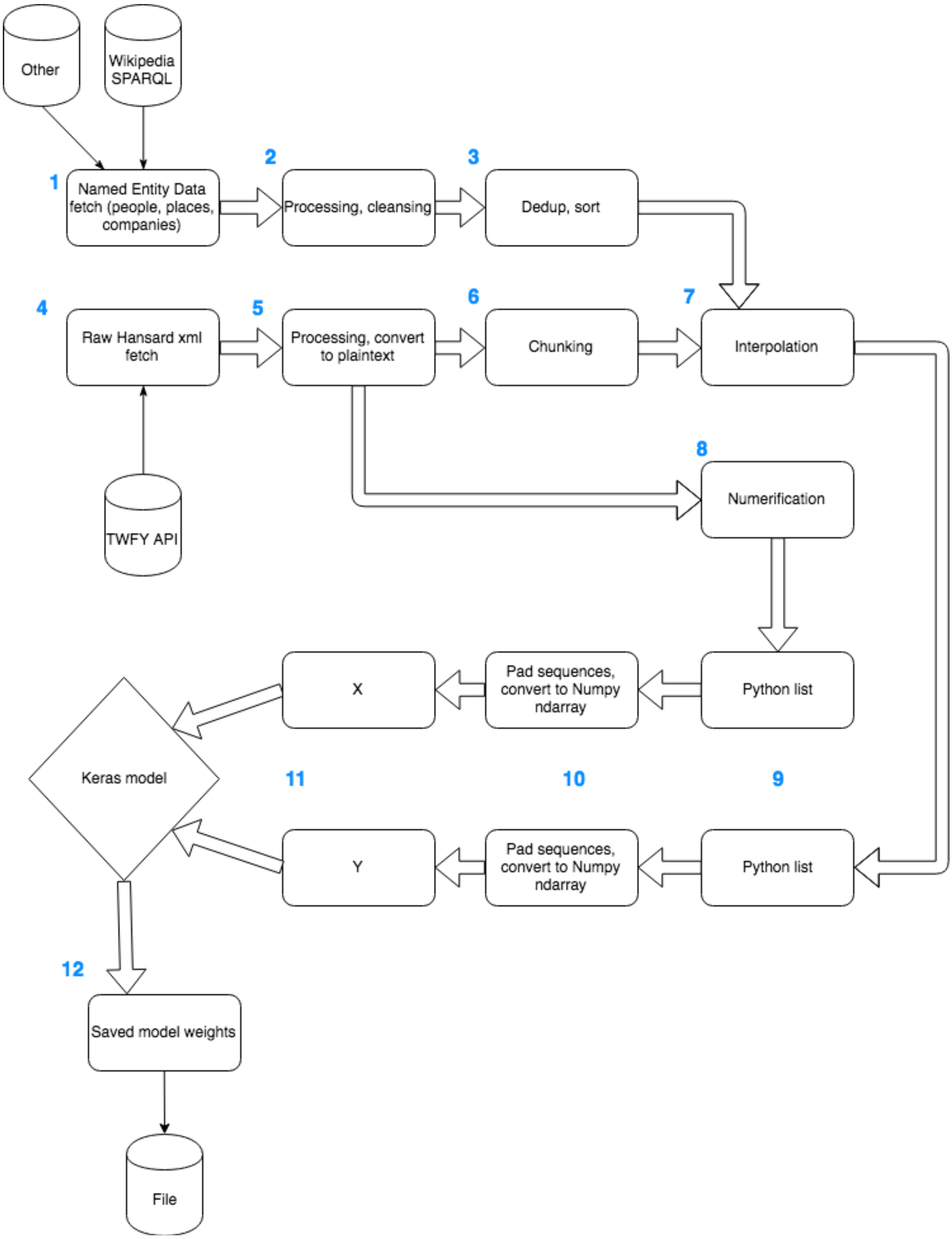

Figure 2 pipeline data processing model

5.3. 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] 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 2 shows, the size of the DBPedia datasets dwarf the other datasets for all three Named Entity (NE) types.

Table 2 % 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 6.1.

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.

5.4. 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 commons 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.

The XML documents downloaded contained all the debates for a given day. For example, the XML document for 17th May 2018 contains the text for all the Commons debates that happened on that day. In order to write out each Commons debate to its own file by title, the 'colnum' column number values were taken from the TheyWorkForYou returned

[15] https://www.kaggle.com/new-york-city/nyc-baby-names

metadata, and were used to divide up the XML document into separate documents for each day. Each element in the XML document has a ‘colnum’ tag which could be checked to see if it was in the correct range – see Code Snippet 1.

```
for elem_type in elem_types:
    for tag in tree.findall(elem_type):
        colnum = tag.attrib['colnum']
        if int(colnum) not in range(debate_colnum_start, debate_colnum_end):
            tree.remove(tag)
```

Code Snippet 1 Break XML documents up into one document per debate

5.5. 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] which fails to remove nested HTML tags, the lxml library’s etree module[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.

5.6. 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 5.7). 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 2.

[16] https://github.com/mozilla/bleach
[17] https://lxml.de/api/lxml.etree-module.html

```
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)
```

Code Snippet 2 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 6). 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 3 shows.

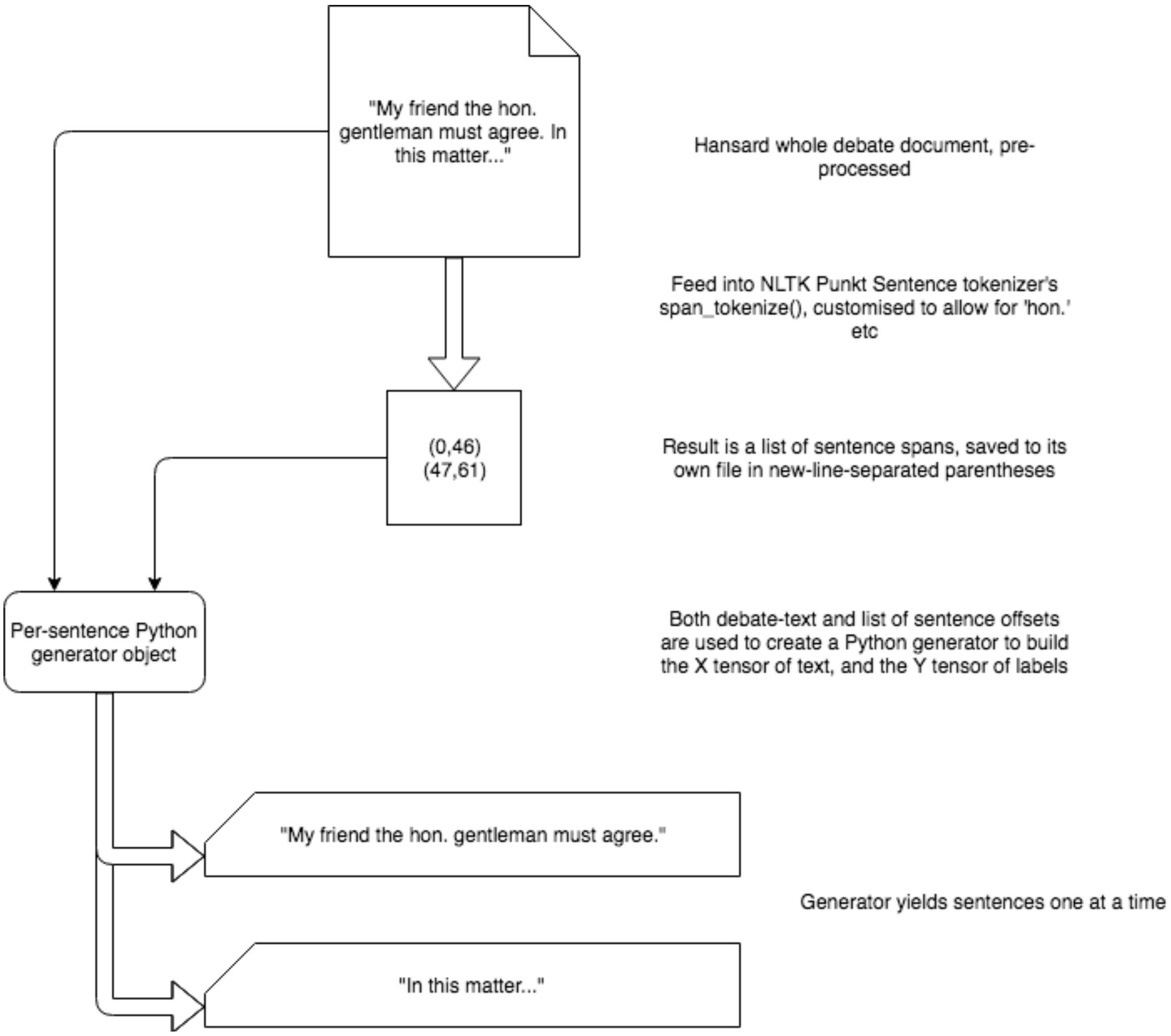

Figure 3 chunking process

5.7. Hansard interpolation

Step 7. The interpolation algorithm is detailed in Figure 4. 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 6.3). The default value used for n was 4, and as such 4-word NEs are the longest that we can interpolate. The ngrams are right-padded with the Python nil object, ‘None’ (see step 3 of Figure 4), 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 3. Note how ‘Railtrack’ has been labelled 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 12.

```
(masters_venv) mattlap:~/bbk/masters_project mralph$ inv hansard-display-interpolated-file --filepath
'hansard_gathering/processed_hansard_data/1994-05-09/Orders of the Day &#8212; Italian Steel Industry.txt'
Using TensorFlow backend.
…
Railtrack has operational responsibility for a new safety regime which will ensure that the already high standard of safety on
the railways is maintained and improved.
2222222220000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000
```
Code Snippet 3 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 6.2. 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 9.2.

5.8. 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 5.9, 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] 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.

[18] http://www.unicode.org/versions/Unicode11.0.0/

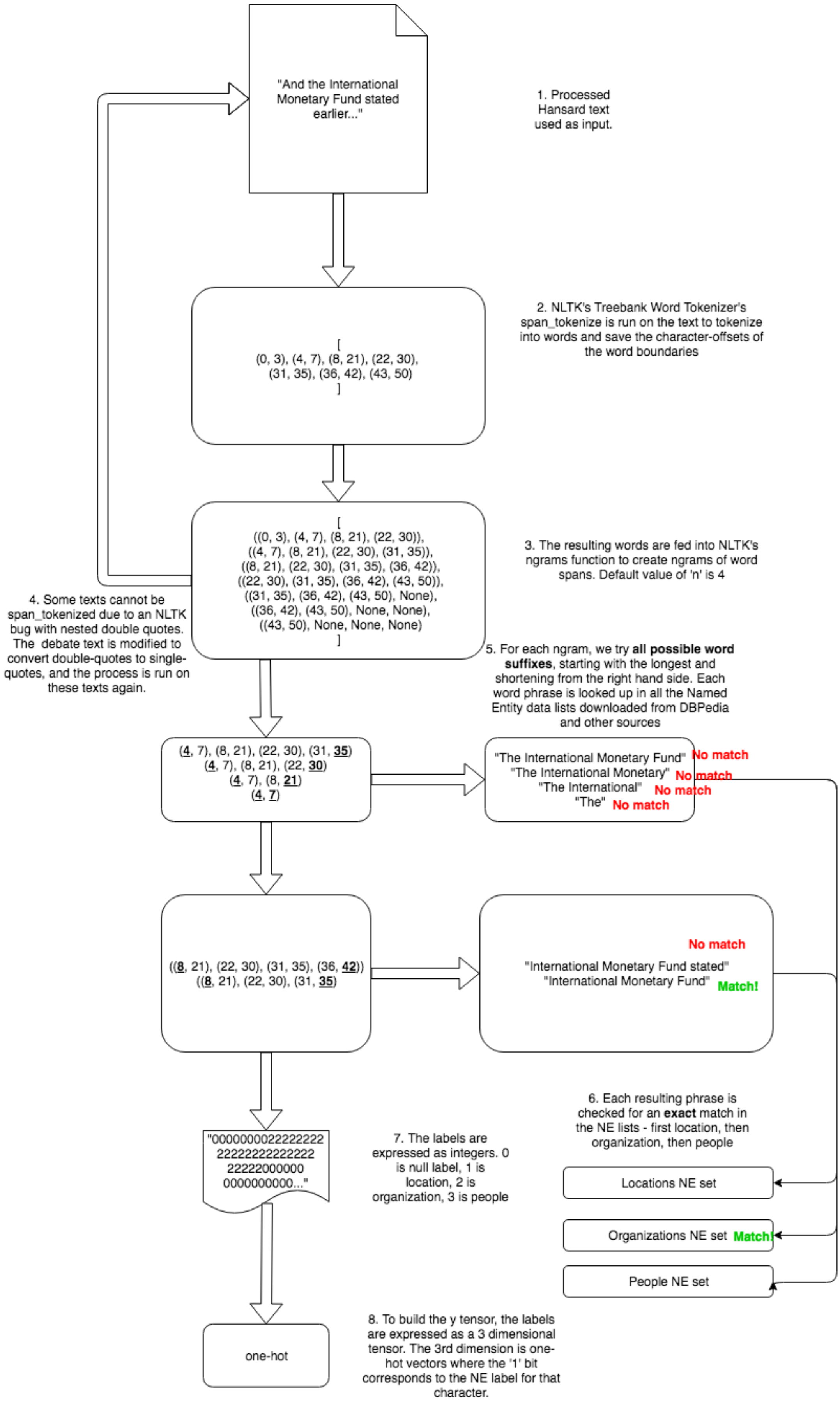

Figure 4 Interpolation algorithm

5.9. 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 4. 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 7.1.3 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 7.1.5.

```python
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.
    """
    # Keeping ALL artificially small for now while we get the model working.
    if dataset_name == "ALL":
        return list(range(8))
    elif dataset_name == "train":
        return list(range(0, 4))
    elif dataset_name == "dev":
        return list(range(4, 6))
    elif dataset_name == "test":
        return list(range(6, 8))
    # Small set of debates to build an alphabet off
    elif dataset_name == "alphabet-sample":
        return [0]
    else:
        return []
```

Code Snippet 4 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 2, and referred to as 'numerification' in the codebase. See section 5.8 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.

5.10. 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 3 and Table 4.

Table 3 X tensor dimensions

Dimension	Content	Length
1	Text Samples	Batch-size length (varies depending on dataset)
2	Characters	Sentence_maxlen (200)

Table 4 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 13.

6. Implementation issues

6.1. 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 5.

Table 5 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.

6.2. 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 5 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).

```
# 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)]
```

Code Snippet 5 logic to avoid re-interpolating overlapping NEs

6.3. 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] "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] 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 5.7. 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 4.

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 9.3.

6.4. 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

[19] https://github.com/nltk/nltk/issues/2076
[20] https://github.com/nltk/nltk/pull/1864

'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.

6.5. 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 5 and Figure 6, both taken from Hansard debate "Flying Bomb Attacks (Meetings with Ministers" from the 7th July 1944.


```
<?xml version="1.0" encoding="ISO-8859-1"?>
<publicwhip scrapeversion="a" latest="yes">
<major-heading id="uk.org.publicwhip/debate/1944-07-07a.1429.0"  colnum="1429">PRAYERS</major-heading>
<speech id="uk.org.publicwhip/debate/1944-07-07a.1429.1"  colnum="1429" time="">
<p>[Mr. SPEAKER <i>in the Chair</i>]</p>
</speech>
<major-heading id="uk.org.publicwhip/debate/1944-07-07a.1429.2"  colnum="1429">FLYING BOMB ATTACKS (MEETINGS
WITH MINISTERS)</major-heading>
<speech id="uk.org.publicwhip/debate/1944-07-07a.1429.3"  hansard_membership_id="3897"
speakerid="uk.org.publicwhip/member/14496"  speakername="The Secretary of State for Foreign Affairs (Mr. Eden)"
colnum="1429" time="">
<p>The suggestion made by the Prime Minister yesterday that my right hon. Friends the Home Secretary…p>
</speech>
```


Figure 5 Hansard debate, XML format

PRAYERS

[Mr. SPEAKER in the Chair]

FLYING BOMB ATTACKS (MEETINGS WITH MINISTERS)

The suggestion made by the Prime Minister yesterday that my right hon. Friends the Home Secretary…

Figure 6 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.

7. Evaluation and Testing

7.1. 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 5.7.

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 6. Baseline accuracy is shown in Table 7 and described in section 7.1.1. Evaluation of all the models trained is shown in Table 8 and analysed in the rest of this section below.

Table 6 Model datasets used

Model name	Training dataset size	Dev dataset size	Test dataset size	Epochs[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 7.1.5.

Table 7 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

[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

Table 8 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]	0.9570	0.5270	0.1386
ToyV2	Test	ToyV2	0.9538	0.5532	0.1564
ToyV3	Test	ToyV3	0.9594	0.5454	0.1364

7.1.1. 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 7.

7.1.2. 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 7.1.3), 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

[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.

returned NULL labels for every character; see Table 8, where the mini model's non-null label accuracy is 0.0.

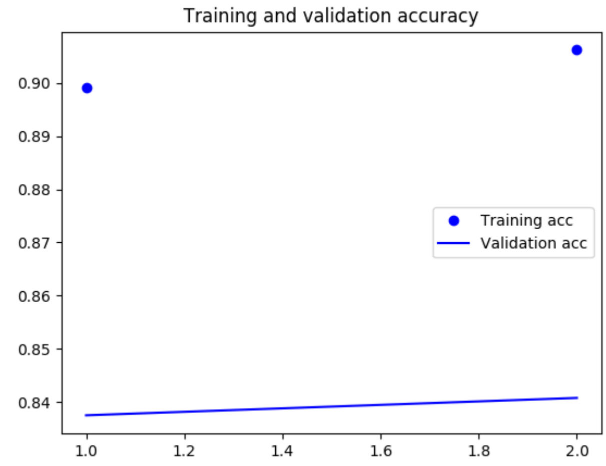

Figure 7 Mini dataset accuracy

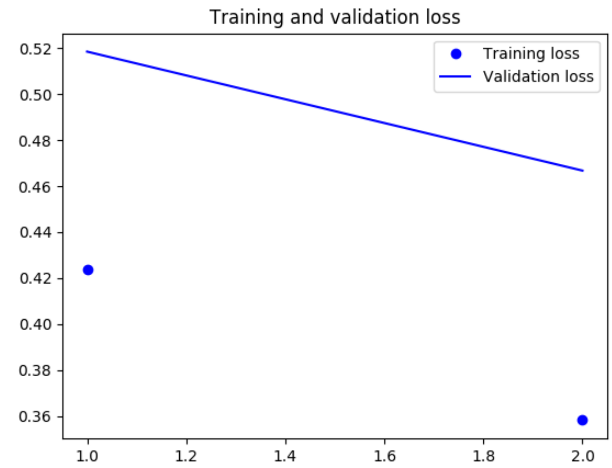

Figure 8 Mini dataset loss

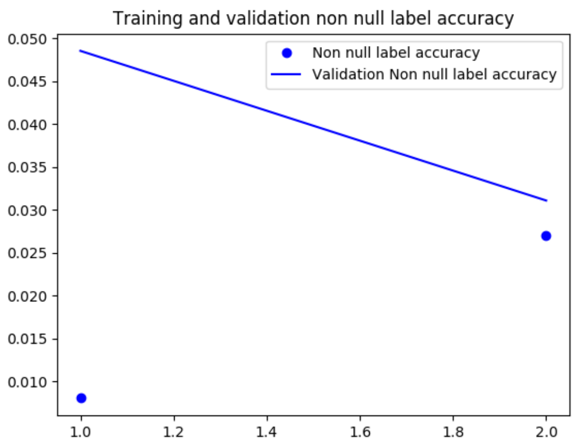

Figure 9 Mini dataset non-null label accuracy

7.1.3. 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.

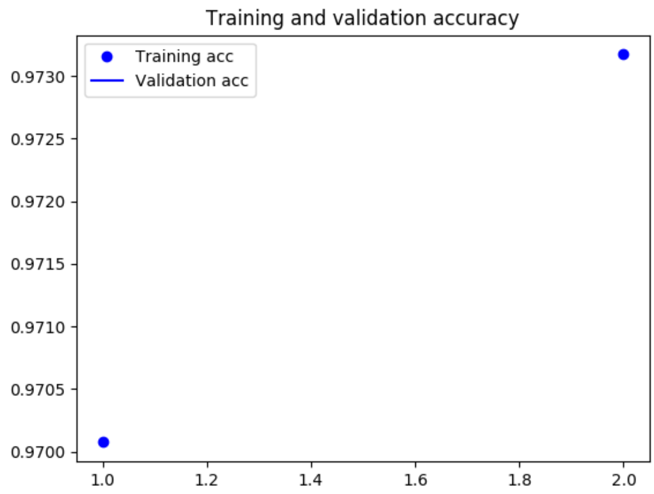

Figure 10 Toy dataset NaN validation accuracy

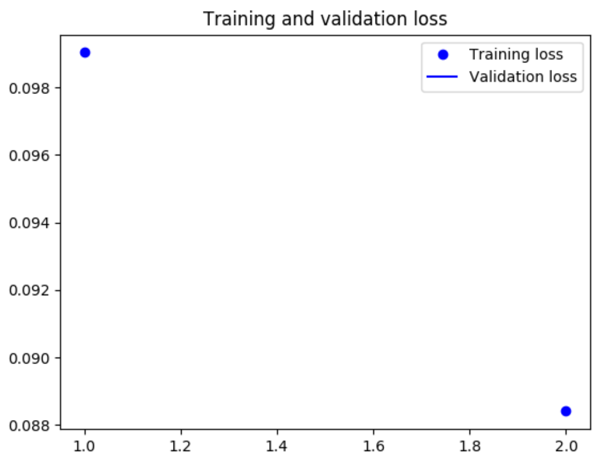

Figure 11 Toy dataset NaN validation loss

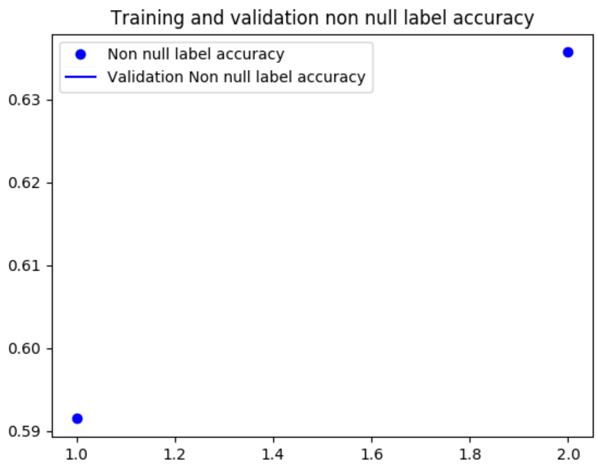

Figure 12 Toy Dataset NaN Validation Non-Null Label Accuracy

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 10, Figure 11 and Figure 12 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 8 (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.

7.1.4. 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 13, Figure 14 and Figure 15).

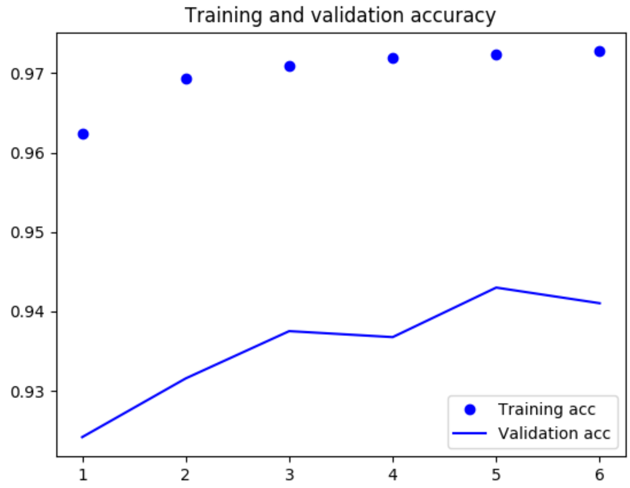

Figure 13 ToyV2 dataset accuracy

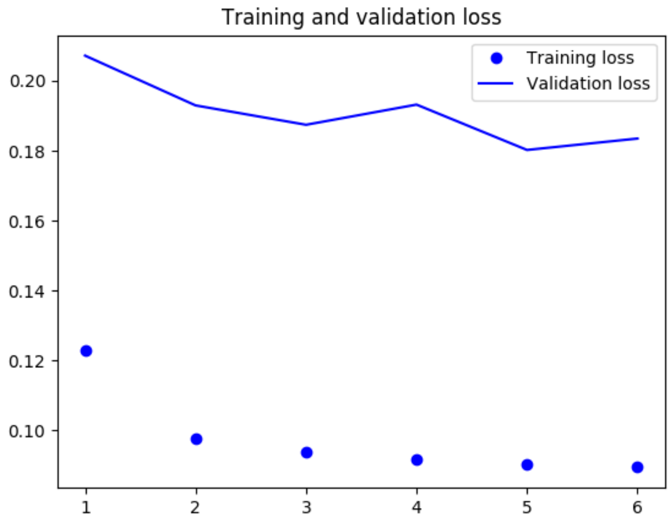

Figure 14 ToyV2 dataset loss

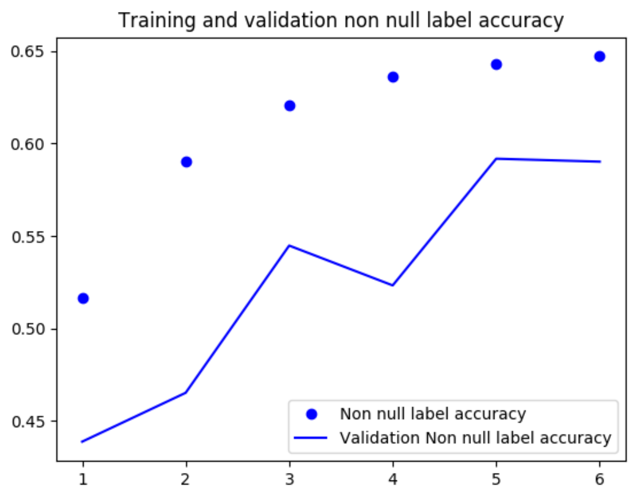

Figure 15 ToyV2 dataset non-null label accuracy

Note from Table 8 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 8 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 16, Figure 17 and Figure 18. 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 8.

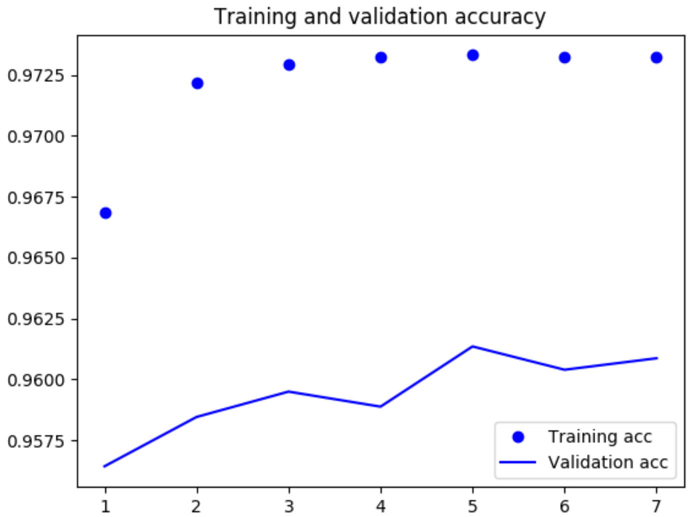

Figure 16 ToyV3 dataset accuracy

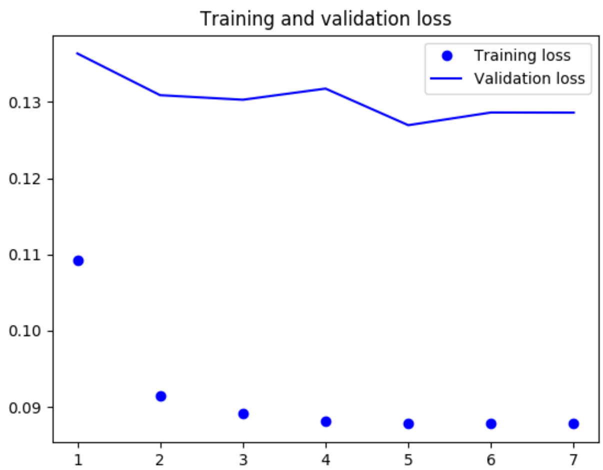

Figure 17 ToyV3 dataset loss

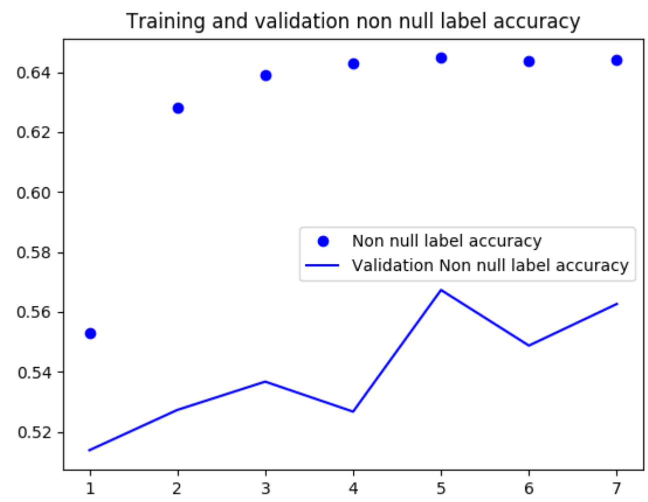

Figure 18 ToyV3 dataset non-null label accuracy

7.1.5. 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 7.1.3 would recur, given the much larger dataset (182,575,072 sentences in total, see Table 6). 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 6 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.

```
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)
```

Code Snippet 6 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 9.5.

7.1.6. 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 7 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 8 shows an example of this approach.

Because of issues mentioned with using the full dataset explored in section 7.1.5, 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.

```
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)
```

Code Snippet 7 k-fold cross validation

```
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)
```

Code Snippet 8 Python subclassing to add model evaluate

The cross-validation results are shown in Table 9. As of 14th September 2018, the departmental server, Venus, is evaluating epoch 10 out of 10 (having run since 24th August), so these results will be available shortly.

Table 9 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]	10	3	TBC	TBC	TBC

7.2. 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 5.1.

In order to validate that a function was writing out expected values to disk, Pyfakefs[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 9 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 10, 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.

```
def test_onehot():
    result = onehot(4, 7)
    expected = [0, 0, 0, 0, 1, 0, 0]
    assert result == expected
```

Code Snippet 9 a regular Pytest unit test

[23] See Table 6 for a description of this dataset.
[24] https://github.com/jmcgeheeiv/pyfakefs

```
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 ==
"0000000000000000033333333333333333000000000000000000000222222222200001
1111"
```
Code Snippet 10 A Pyfakefs Pytest unit test

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

7.3. Overall evaluation

The user experience was manually validated using Simple-GUI (see section 8). 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 19 of the view pre-NE annotation, and in Figure 20 of post-NE annotation.

We do business with the United States Administration because the United States is our closest strategic partner. Where we disagree on issues such as steel, we make our voice very clear. We do not support the use of section 232 as a mechanism for dealing with the overproduction of steel. That actually hits the United States' allies and not the designed target, which was China. Citing national security, particularly in Britain's case, makes no sense at all given that some of the steel that we send to the United States goes into its military programmes.

Figure 19 Example Hansard text before NE annotations - SimpleGUI

We do business with <loc>the United States</loc> Administration because <loc>the United States</loc> is our closest strategic partner. Where we disagree on issues such as steel, we make our voice very clear. We do not support the use of section 232 as a mechanism for dealing with the overproduction of steel. That actually hits <loc>the United States</loc>' allies and not the designed target, which was China. Citing national security, particularly in <loc>Britain</loc>'s case, makes no sense at all given that some of the steel that we send to <loc>the United States</loc> goes into its military programmes.

Figure 20 Example Hansard text after NE annotations - SimpleGUI

8. Graphical User Interface

8.1. 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 8.2 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 21 shows the design of the HTTP endpoints to the server, and the role of client-side Javascript.

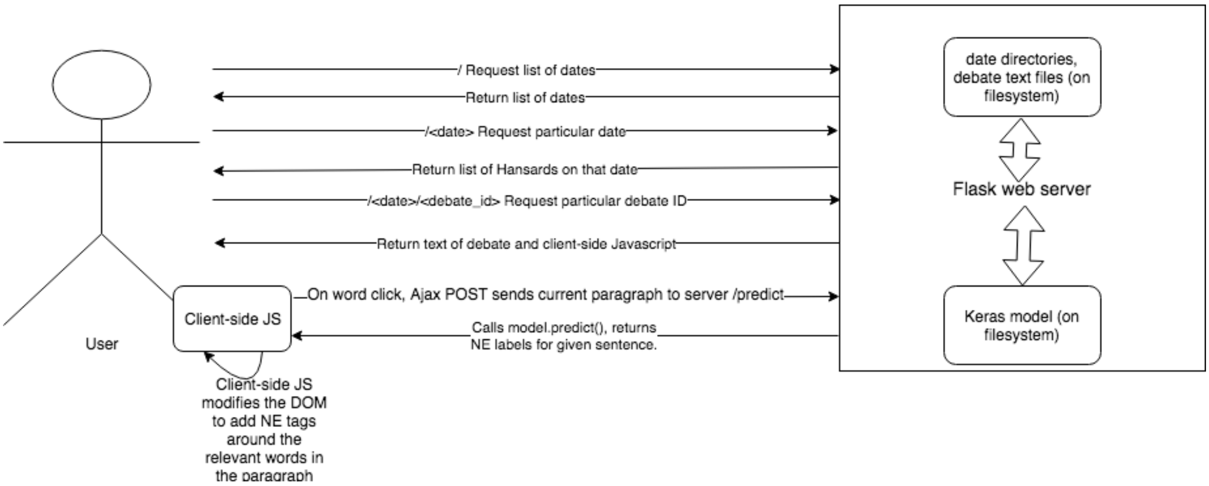

Figure 21 Simple-GUI basic design

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] which we use to template the HTML files for the site, and Werkzeug[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

[25] http://jinja.pocoo.org/
[26] http://werkzeug.pocoo.org/

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 21, 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] "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.

8.2. 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.

[27] https://jqueryui.com/

As of this writing, on 14th September 2018, the loading of Hansard data into Samtla is complete for a number of sample days (the whole dataset was found to take too long), including 17th May 2018. This project's Javascript and Flask server code has been uploaded to the Birkbeck server Victoria2a, but there are still some Python library dependencies problems to be sorted before the integration can be demonstrated on Samtla.

9. Summary and Conclusions

9.1. 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. Also the accidental corruption or deletion of data, and consequent restoration from backups, necessitates the taking of rigorous backups at each stage of processing, adding to the time taken.

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

9.2. 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 7.1 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 5.7 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 6.2, 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.

9.3. 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 6.3, 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.

9.4. Neural networks are slow and opaque

As detailed in 7.1.5, 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. Even running cross-validation took more than 3 weeks to run, on a large Birkbeck server.

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]

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.

9.5. 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 (the code to utilise the fit_generator method has already been written and submitted as part of this project). 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 9.2. 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 span of time. Perhaps their content is simply too broad to be considered as a single genre. If that were true, better results could hypothetically be achieved by filtering Hansard debates down to those on similar topics, such as transportation, or international affairs.

[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.

A bug was discovered right at the end of this project, whereby a few documents were repeated across multiple datasets. 84 documents were found to be repeated, over the 1584 documents in the datasets used for the 'Toy' datasets. This was caused by the XML responses from the TheyWorkForYou unexpectedly containing all debates for a given day, when the HTML responses contain just one debate each. This does mean that the model may have been evaluated on some data on which it was trained in up to 5% of the documents in the 'train' dataset. The XML-downloading code has since been fixed, but all the data should be downloaded again and re-processed to re-validate the accuracy statistics given above.

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.

10. References

The following publications were quoted in this report and the proposal, and used as the theoretical underpinning for this project:

Harris, M. *et al.* (2014) 'The anatomy of a search and mining system for digital humanities', *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries*, (August), pp. 165–168. doi: 10.1109/JCDL.2014.6970163.

Jurafsky, D. and Martin, J. H. (2009) *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition*. Second. Pearson/Prentice Hall.

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.

Smarr, J. and Manning, C. D. (2002) 'Classifying Unknown Proper Noun Phrases Without Context'.

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.

11. Appendix A: User Manual

11.1. Data pipeline manual

The data pipeline is run by using the invoke tasks listed in section 12, 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.

11.2. 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.

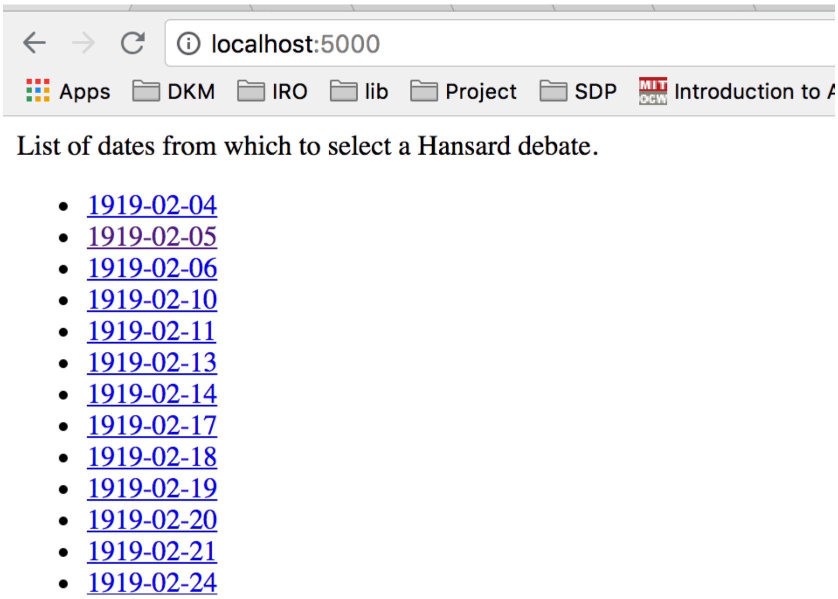

Figure 22 Simple-GUI index page

Figure 23 Simple-GUI date page

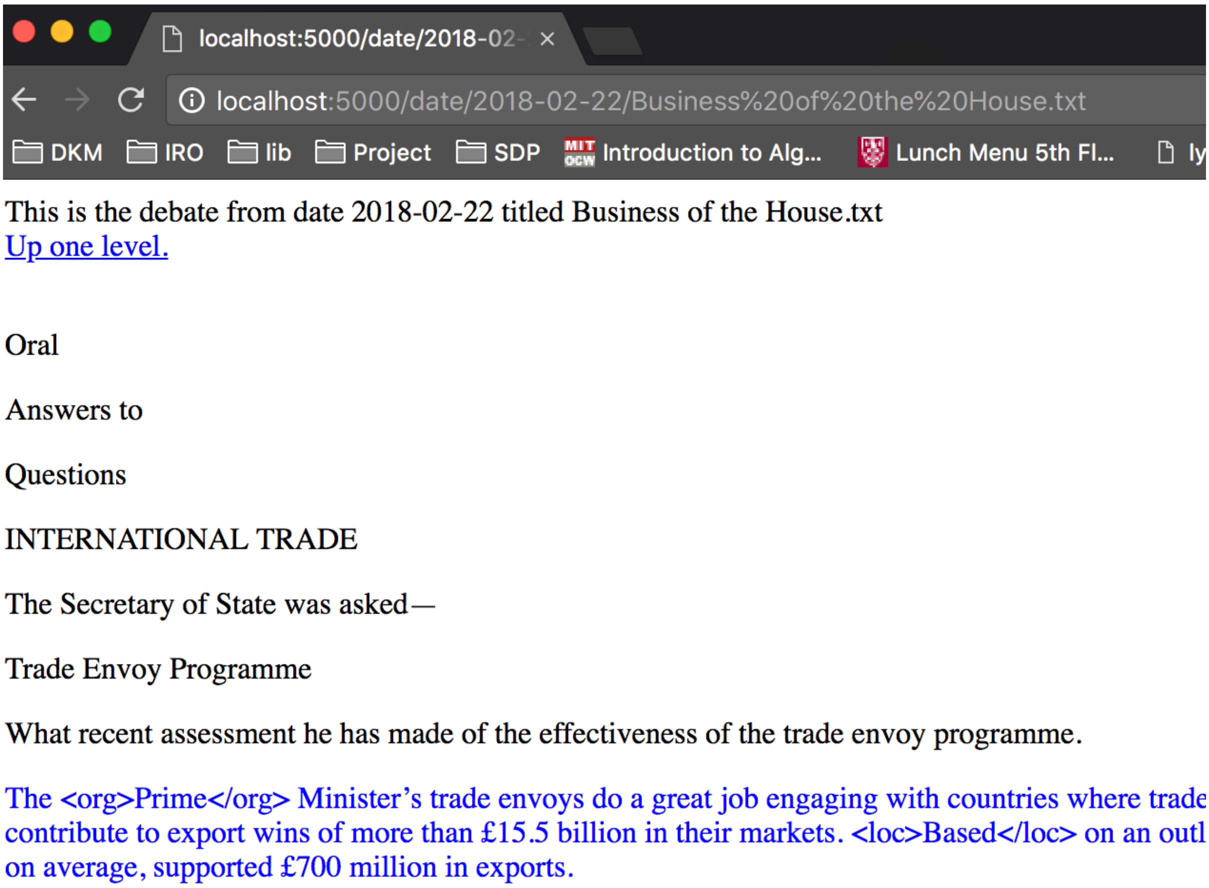

Figure 24 Simple-GUI debate page with an annotated paragraph.

12. Appendix B: 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 5.9.
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 5.6
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 the Commons, from a given starting date.
Hansard-download-for-date	Download all Hansard commons debates for a given 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 5.5.
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.

13. Appendix C: 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

14. Appendix D: 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.

14.1. 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_for_date(ctx, date):
    driver.get_hansards_for_date(date)


@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_process_for_date(ctx, date):

preprocessing.process_hansard_directory("hansard_gathering/raw_hansard_data/{}".forma
t(date))

@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 -v 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()
```

14.2. config_util/config_parser.py

```
import yaml


def parse_config():
    with open('./config.yml') as config:
        conf = yaml.load(config)
        return conf
```

14.3. 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("@@@")
```

14.4. hansard_gathering/driver.py

```
from config_util.config_parser import parse_config
from datetime import datetime, timedelta
from urllib.parse import urlparse
from lxml import etree  # type: ignore
import concurrent.futures
import json
import lxml  # type: ignore
import os
import requests
import re
import sys

# 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 get_debate_colnum_start(debate) -> int:
    return int(urlparse(debate[1]).query.split(".")[1])


def get_debate_colnum_end(idx, debates_list) -> int:
```

```
    if idx == len(debates_list) - 1:
        # This is last debate in xml, so get all remaining columns
        return sys.maxsize
    else:
        # My last colnum is the colnum at which next debate starts
        return get_debate_colnum_start(debates_list[idx + 1])


def remove_other_debate_content(tree, debate_colnum_start, debate_colnum_end):
    elem_types = "speech major-heading minor-heading oral-heading".split()
    for elem_type in elem_types:
        for tag in tree.findall(elem_type):
            colnum = tag.attrib['colnum']
            if int(colnum) not in range(debate_colnum_start, debate_colnum_end):
                tree.remove(tag)
    return tree


def download_and_split_debates_for_date(datestring, debates_list):
    """
    Given a list of debate titles, download them all into files,
    correctly splitting the XML into separate files per title.
    :param datestring: Date to download for
    :param debates_list: List of tuples of (title, HTML url, XML url)
    """
    # Currently, all debates for a given day are actually served in one big XML file.
    # Just take the debate XML url from the first one, so we only download
    # this XML once, and then take the right colnums for each debate title to
    # write out to separate debate files.

    parser = etree.XMLParser(ns_clean=True, recover=True, encoding='utf-8')
    xml_url = debates_list[0][2]
    if xml_url == "N/A":
        return
    xml_data = requests.get(xml_url).text

    for idx, debate in enumerate(debates_list):
        print("Data for {}: {}".format(datestring, debate))
        title = debate[0].replace("/", "")  # UNIX filenames cannot contain forward slash
        debate_colnum_start: int = get_debate_colnum_start(debate)
        debate_colnum_end: int = get_debate_colnum_end(idx, debates_list)

        tree = etree.fromstring(xml_data.encode('utf-8'), parser=parser)
        tree = remove_other_debate_content(tree, debate_colnum_start,
debate_colnum_end)
```

```
os.makedirs("hansard_gathering/raw_hansard_data/{datestring}".format(datestring=datest
ring), exist_ok=True)
        with open("hansard_gathering/raw_hansard_data/{datestring}/{title}.xml".format(
                datestring=datestring, title=title), "w") as f:
            f.write(lxml.etree.tostring(tree).decode('utf-8'))


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].replace("/", "")  # UNIX filenames cannot contain forward slash
        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=datest
ring), 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")

    # TODO lords titles don't seem to work with scraped xml?
    lords_titles = get_hansard_titles(datestring, "Debates", "lords")

    download_and_split_debates_for_date(datestring, commons_titles)


def get_hansards_for_date(date):
    date_regex = re.compile(r"\d\d\d\d-\d\d-\d\d")
    assert date_regex.match(date), \
        "Date must be yyyy-mm-dd"

    get_titles_and_download(date, "Debates")


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])
```

14.5. 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
```

14.6. 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")
```

14.7. 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
```

14.8. hansard_gathering/preprocessing.py

```
from lxml import etree  # type: ignore
from typing import Generator
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_directory(dir_path):
    print("Processing Hansard directory {}".format(dir_path))
    for _file in glob.glob("{dir_path}/*.xml".format(dir_path=dir_path)):
        print("Found file {}".format(_file))
        process_hansard_file(_file)


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)
```

14.9. 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)
```

14.10. 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")
```

14.11. 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)
```

14.12. 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(datas
et_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(datas
et_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]
```

14.13. 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))
```

14.14. 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))
```

14.15. 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)
```

14.16. 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()
```

14.17. 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)
```

14.18. 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")
```

14.19. 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)
```

14.20. 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__N
ew_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))
```

14.21. 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))
```

14.22. 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)
```

14.23. 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(host=0.0.0.0, load_dotenv=False, debug=True, port=5000, threaded=True)
```

14.24. 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)
```

14.25. 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");
    })
});
```

14.26. simple_gui/static/style.css

```
body {
    background-color: #ffffff;
}

h1 {
    color: #fff;
    font-family: Arial, Helvetica, sans-serif;
}

.rendered {
    color: blue;
}
```

14.27. 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>
```

14.28. 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>
```

14.29. 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>
```

14.30. 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
```

14.31. 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)
```

14.32. 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 ==
"000000000000000003333333333333333300000000000000000000022222222220000111
11"


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)
```

14.33. 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)
```

```python
    result = file_lines("/var/data/a.txt")
    expected = 3
    assert result == expected
```

14.34. test/test_model_integration.py

```python
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
```

14.35. test/test_preprocessing.py

```python
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
```

14.36. test/test_simple_gui_util.py

```python
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
```

14.37. 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
```