Information Retrieval on the USC

Work by: Cordeiro Antonio, Gravili Niccolò, Rastogi Archit, Criveanu Bogdan

ABSTRACT

With our project we want to propose an easy and efficient way to access the United States Code, which is defined on the house.gov website as:

"*a consolidation and codification by subject matter of the general and permanent laws of the United States. It is prepared by the Office of the Law Revision Counsel of the United States House of Representatives.*"

Our objective is, given in input a natural language query, to output the section of the US code that is most likely to hold something relevant to the query, the way we achieve this is through an NLP approach: vectorizing the documents in the dataset and the query and then ranking such documents by computing the cosine similarity with the query vector.

We implement and rate the accuracy of different approaches for this same task.

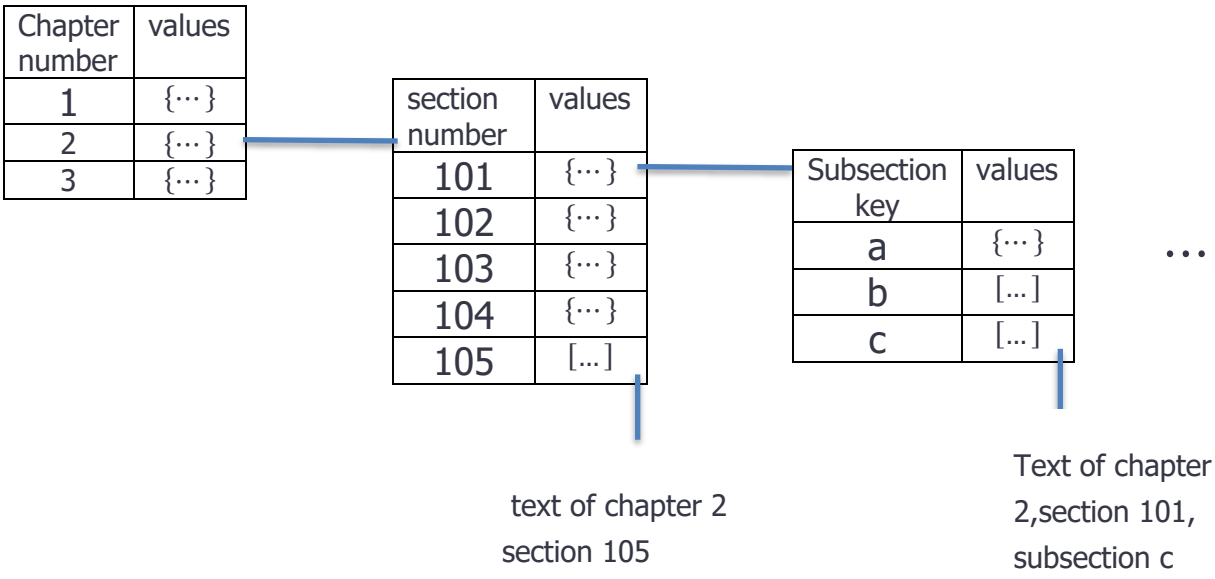
FORMATTING THE DATA

Retrieval

The US Code is reported on the official site *https://uscode.house.gov/*. but, after thorough research across the internet we were not able to find a well-organized version of these regulations useful for our purposes. Thus, we had to retrieve it by ourselves and organize it to our liking. The only useful type of file is the XML for the 54 titles, so we started working on that.

The XML file for each title is subdivided into chapters and then sections, but the structure is not even. Often there are optional upper levels, such as subtitles and subchapters, and optional subdivisions, like subsections, paragraphs, subparagraphs, and clauses.

We wrote a script that takes just the text of the regulations, excluding the XML formatting information and irrelevant text present on the page and puts it into a dictionary respecting the original structure of the data as closely as possible. This dictionary holds as keys the chapter numbers and as values other dictionaries which in turn hold as keys sections numbers and as values text if there is no further subdivision or another nested dictionary and so on.



From this structure we extracted a cleaner dictionary we could actually work with, having as keys strings including each index of each subdivision and as value the text corresponding to that division (e.g. "5.I.12.1202.a" is the key associated to "The term of office of each member of the Merit Systems Protection Board is 7 years.").

But when starting to run the first demos utilizing the values of this dictionary as our collection of documents, we realized that some subdivisions contained too little information to be useful, thus outputting results apparently not relevant to our queries.

We decided to rework the dictionary once more, creating a final version that holds as keys a string "title.chapter.section" and as value all the text in the section, ignoring any another uplevels or subdivisions present in the data (e.g. the key used in the previous example "5.I.12.1202.a" would just become "52.12.1202" and all the subsections in section 1202 would be included in it's value)

key	values
1.1.1	[...]
1.1.2	[...]
1.1.3	[...]
1.1.4	[...]
⋮	⋮
1.2.101	[...]
1.2.102	[...]
⋮	⋮

Queries

Querying the model and computing it's accuracy was a big challenge, there is no official dataset of queries and answers on the internet and using data from court judgements would be wrong as they could apply some state specific laws that are not present in the USC, the data itself is massive and written in technical language that we can't always correctly interpret without the help of a lawyer, queries that are too general may involve different aspects of the law and thus have more than one relevant section to them, in order to correctly test our model we need a big number of specific queries written in technical language that can each be answered by only one section of the U.S.C., to achieve this we leveraged chatGPT, we fed it random sections from the data and prompted it to generate questions about each section.

We generated a dictionary holding as keys section strings (e.g. "52.12.1202") and as values queries relevant to the section in the key. We generated 336 queries about 112 sections of the USC and used it to test our model.

Due to all the aforementioned problems and the use of chatGPT to generate the queries we cannot really claim any significance of our accuracy metrics, but they act as an indicator of how different versions of our model perform against each other.

Pre-processing

The text is pre-processed by simple lowercasing and stop words removal, we decided against other pre-processing techniques such as lemmatization as it seemed to just output worse results.

ACCURACY

We choose to measure the accuracy taking into consideration the first 10 most relevant sections the model outputs, however, Given the layered nature of the data, we need a metric that takes into account not only if the correct section is or isn't in the output but also in what position it has been ranked.

Furthermore, even if the correct section is not present in the model's output we still want to award the model some small non-zero accuracy score if it gets the title or the chapter right, thus we devised an accuracy score that works like this:

If the correct answer in is the model's output:

$$accuracy = 1 - 0.05 \cdot index$$

where $index$ is the position at which the correct answer has been ranked in the output

otherwise:

$$accuracy = \frac{0.2 \cdot n_t + 0.3 \cdot n_{ch}}{10}$$

where n_t is the number of answers with only the right title and n_{ch} is the number of answers with both the right title and the right chapter.

this way we award an accuracy in the range [0.55;1] if the correct section is in the model's output, or [0;0.5] if the correct section isn't found.

We also define a second simpler accuracy metric that just counts one if the correct answer is found in the output and 0 if not.

The combination of these two accuracy metrics should describe pretty well the behaviour of the model for our purposes

THE APPROACHES

As we mentioned earlier our approach consists in ranking the documents of our collection by cosine similarity with the query vector, in particular, utilizing a set of pre-trained embeddings for word2vec, we define a document embedding matrix $DE \epsilon \mathbb{R}^{dxh}$ (where d is the number of documents and h the dimensionality of the embeddings) such that each row of DE is the embedding of a section and a query embedding $q \epsilon \mathbb{R}^{1xh}$ that is a vector representing the query.

we then use the NearestNeighbours module from sklearn to compute the top 10 closest embeddings in DE to the query embedding q using cosine similarity as a metric.

The three different approaches we implemented each differ in the way we define q and DE.

each approach is both implemented with the google news word2vec embeddings and the law2vec embeddings (law2vec embeddings are vectors of size 200 learnt over a corpus of legal documents including the USC itself)

Unweighted

In this approach we define the i_{th} row of DE as:

$$DE_i = \frac{\sum_{w \in W_i} w}{\mathrm{n_i}}$$

Where W_i is the sequence of word2vec embeddings for the words that are part of the i_{th} document in the collection and $\mathrm{n_i}$ is the number of words in the i_{th} document

Meaning each document embedding is the average of the embeddings for the words making it up, similarly the query embedding q is defined as:

$$q = \frac{\sum_{w \in W_q} w}{\mathrm{n}_q}$$

where W_q is the sequence of word2vec embeddings of the words in the query and n_q the number of words in the query

weighted

In this approach we compute each document embedding as the weighted average of the embeddings for the words making up the document using as weights are the tf-idf embeddings for each word.

We define DE as:

$$DE = TFIDF \times WE$$

Where $TFIDF \in \mathbb{R}^{d \times w}$ (With d number of documents in the collection and w number of words in the vocabulary) is the term-document matrix obtained by applying tf-idf on the collection of documents

And $WE \in \mathbb{R}^{w \times h}$ is the matrix holding the embeddings obtained trough word2vec for each word in the vocabulary.

The query embedding q is defined as:

$$q = TFIDF_q \times WE$$

where $TFIDF_q \in \mathbb{R}^{1 \times w}$ is the tf-idf embedding for the query

in order to make this approach work the query embedding is weighted trough tf-idf, as showed ,but doing this means that all out-of-vocabulary words that may be present in the query get weighted to 0 and thus not considered at all, in part defeating the purpose of using pre-trained word2vec embeddings, the next approach aims to find a solution to this.

mixed query

this approach defines DE in the same way the weighted approach does, we only change the way q is defined:

$$q = 0.5 \cdot q_{weighted} + 0.5 \cdot q_{unweighted}$$

where:

$$q_{weighted} = TFIDF_q \times WE$$

$$q_{unweighted} = \frac{\sum_{w \in W_q} w}{\mathrm{n}_q}$$

The intuition behind this approach is that a mix between the weighted and unweighted query embeddings will partially hold the benefits of weighting the query without throwing away the information of the out-of-vocabulary words.

THE RESULTS

Results on ChatGpt generated queries

The following table holds the results for all approaches in both accuracy metrics considered.

APPROACH	Avg. CUSTOM ACCURACY	DOCUMENTS FOUND (%)
Unweighted google word2vec	0.57	0.60
Weighted google word2vec	0.58	0.60
Mixed query google word2vec	0.59	0.62
Weighted law2vec	0.52	0.54
Unweighted law2vec	0.51	0.53
Mixed query law2vec	0.52	0.54

We notice that google word2vec performs consistently better than law2vec suggesting that a bigger, more general training set for word2vec embeddings may be better than smaller but more specific one for tasks such as this one.

We also notice that the weighted and mixed query models perform better than their unweighted counterparts.

General queries

In this section we query the system with simple and general queries to test if it's capable to output reasonable answers.

Once again, we state our inability to correctly interpret how relevant the sections found by the model really are to the query given the complexity of the laws themselves.

The first query is:

"*I shot the president and now he is dead, what now?"*

The relevant answers by the model seem to be:

Section 18.41.879

https://uscode.house.gov/view.xhtml?req=(title:18%20section:879%20edition:prelim)%20OR%20(granuleid:USC-prelim-title18-section879)&f=treesort&edition=prelim&num=0&jumpTo=true

Ranked in 1st place.

Section 02.61.6111

https://uscode.house.gov/view.xhtml?req=(title:2%20section:6111%20edition:prelim)%20OR%20(granuleid:USC-prelim-title2-section6111)&f=treesort&edition=prelim&num=0&jumpTo=true

Ranked in 7th place.

The second query is:

"*How is the American flag composed?*"

The relevant answers by the model seem to be:

Section 04.1.1

https://uscode.house.gov/view.xhtml?req=(title:4%20section:1%20edition:prelim)%20OR%20(granuleid:USC-prelim-title4-section1)&f=treesort&edition=prelim&num=0&jumpTo=true

Ranked 1st

All other answers also seem to be closely related to the query, talking about the national flag day, the rules against disrespect to the American flag etc.

The third query is:

"*What are the education programs for disabled kids?"*

The relevant answers by the model seem to be:

Section 20.33.1400

https://uscode.house.gov/view.xhtml?req=(title:20%20section:1400%20edition:prelim)%20OR%20(granuleid:USC-prelim-title20-section1400)&f=treesort&edition=prelim&num=0&jumpTo=true

ranked 5th

Section 20.33.1450

https://uscode.house.gov/view.xhtml?req=(title:20%20section:1450%20edition:prelim)%20OR%20(granuleid:USC-prelim-title20-section1450)&f=treesort&edition=prelim&num=0&jumpTo=true

ranked 9th

all other answers seem to talk about different child education programs and 9 out of 10 answers are in title 20, which is titled "education".

The fourth query is:

"*I ran over a cat with my car".*

The relevant answer by the model seems to be:

Section 07.33.1450

https://uscode.house.gov/view.xhtml?req=(title:7%20section:2158%20edition:prelim)%20OR%20(granuleid:USC-prelim-title7-section2158)&f=treesort&edition=prelim&num=0&jumpTo=true

ranked in 1st place.

Section 18.3.47

https://uscode.house.gov/view.xhtml?req=(title:18%20section:47%20edition:prelim)%20OR%20(granuleid:USC-prelim-title18-section47)&f=treesort&edition=prelim&num=0&jumpTo=true

ranked in 2nd place.

Section 7.54.2160

https://uscode.house.gov/view.xhtml?req=(title:7%20section:2160%20edition:prelim)%20OR%20(granuleid:USC-prelim-title7-section2160)&f=treesort&edition=prelim&num=0&jumpTo=true

ranked in 9th place.

All other answers seem to only to talk about cars or animals in more general aspects.

The fifth query is:

"how many years in jail can I face for evading taxes?"

The relevant answers by the model seem to be:

section 26.75.7202

https://uscode.house.gov/view.xhtml?req=(title:26%20section:7202%20edition:prelim)%20OR%20(granuleid:USC-prelim-title26-section7202)&f=treesort&edition=prelim&num=0&jumpTo=true

ranked 1st

section 26.75.7201

https://uscode.house.gov/view.xhtml?req=(title:26%20section:7201%20edition:prelim)%20OR%20(granuleid:USC-prelim-title26-section7201)&f=treesort&edition=prelim&num=0&jumpTo=true

ranked 4th

all answers seem to talk about evading or omitting to pay taxes in more specific contexts and in general about tax laws violations.

THE PROJECT ZIP FILE

The project zip file is broken into two folders, the "data & parsing" folder holds the script used to parse trough the xml's and to extract the various data dictionaries described in the paper, there will also be the json files for the dictionaries of some of the titles as an example

The second folder "model" will hold a "required data" zip file and a ipynb file with the code for all approaches and accuracy metrics.

In order to try out the model it will be necessary to download the "required data" zip file and open the code into a colab notebook, upon running the second cell of code the user will be asked to upload a file, upload the "required data" file(it will take a while) ,after completing this

step all of the code in the notebook will be runnable, we also include the possibility to query our model with custom queries choosing the approach and pre-trained vectors to use.

links and papers that helped.

https://web.stanford.edu/~jurafsky/slp3/6.pdf

https://web.stanford.edu/~jurafsky/slp3/14.pdf

https://dl.gi.de/bitstream/handle/20.500.12116/3987/B29-2.pdf?sequence=1&isAllowed=y

https://towardsdatascience.com/calculating-document-similarities-using-bert-and-other-models-b2c1a29c9630

https://archive.org/details/Law2Vec