3. Extracting basic information with regular expression

Fortunately, all documents in our dataset begin with a structured opening information, described below:

1) Case number of the document is listed on the top of the whole text;

2) Following the case number is the title of the document, which is in format “PARTY A v. PARTY B” where PARTY A and PARTY B both provide the name as well as the role (plaintiff or defendant) of both parties involved in this case;

3) Position of the date of the court decision in the text is indicated by a special line which contains only one single “|” character. The next line of this “|” line is the date we need;

4) The first appearance of the court name is the full name of the court, which means it starts with “the united states district court”, making it easy to extract;

5) The final verdict of the case is a short single paragraph limited to 1-5 words and several specific words such as “affirmed”, “reversed” and “vacated”, etc..

(A figure?)

Since 1) to 5) listed above all appear in a fixed order and comply to obvious patterns, they can be extracted using simple text/string processing programs and regular expressions.

4. Preprocess of Texts – before extracting patent terms and key sentences

4.1 Removing raw annotations

Raw text files retrieved from Westlaw’s website contain needless annotations.

1) Captions and figure number of figures in raw .pdf documents (e.g. “\*1394”, “Fig. 4”);

2) Some senteces are annotated with superscripts (e.g. “… the 579 patent is invalid.1”);

3) Some words are partially enclosed in square brackets (e.g. “[w]here”, “[a]ccording”). These square-bracket-enclosed words are originally used as indicators of law-related citations;

It is necessary to remove annotations listed above to keep the data as clean as possible and ensure the quality of upcoming tokenization and sentence segmentation tasks. We again use regular expressions to find these annotations and remove them. Some annotations difficult to detect are manually removed by hand.

4.2 Tokenization and Sentence segmentation

SpaCy (acknowledgement?) is used for most NLP-related part of this project. Nevertheless, court decision document in our dataset contain peculiar domain-specific tokens such as law numbers (e.g. “col 1. ll. 12-34” ), case numbers (e.g. “239 F.3d 1324”), trademarks (e.g. “(r)”) and abbreviations of organizations (e.g. “Lab.”, “Inc.”), etc.. Default spaCy becomes insufficient in handling these special cases so that it is vital to add special cases mentioned above to makee a more robust tokenizer.

In terms of sentence segmentation, several customized rules are embedded into the spaCy pipeline to tackle corresponding tricky problems. Some important ones are listed below:

1) A whole sentence breaking into several parts. This can be solved by detecting whether a sentence ends with a period. If not , the sentence is not complete and should be merged with next “sentence piece”. We do not need to consider other punctuations such as “!” and “?” since court decision documents are relatively objective and seldom show sentimental inclination;

2) We split exceedingly long sentences by spliting sentences with “:”;

3) If a sentence contains unpaired brackets or quotation marks, this sentence should be merged with next “sentence piece” till the brackets or quotation marks are correctly matched. This is crucial since court decision documents tend to include a large amount of explanatory information enclosed in brackets;

4) A line in raw text isolated by two blank lines beside it is regarded as a single sentence, even if it might not end with a period or is a gramatically incomplete sentence.

4.3 POS tagging and dependency parsing

We run preprocessed raw text through spaCy nlp pipeline and let spaCy do the POS tagging and dependecy parsing work. The output documents of spaCy pipeline are stored on disk for downstream tasks, i.e. patent terms and key sentences extraction.

4.4 Annotating the patent terms and key sentences (Annotating answers)

5. Extracting patent terms – Supervised key phrases extraction

5.1 High-level method description

In this work, extracing patent terms from a document is viewd as a supervised key extraction

task. Candidate noun chunks are picked out using certain filtering strategies defined on our own. Next, we run a model over all candidate noun chunks and generate a score for each noun chunk .For this part we have tried both TF-IDF model and logistic regression model.

5.2 Find candidate noun chunks

Since we assume that the patent terms appear in former parts of a document, we only take first 40% of noun chunks and abandon the rest. Then, we filter out organizaton names, law numbers and other useless terms using an exclusion directionary. Additionally, if two noun chunks are identical after lemmatization, they are regarded as the same type. At this point we have acquired a relatively clean set of candidates.

5.3 Evaluating candidate noun chunks with neo TF-IDF scores

For each candidates noun chunks, we calculate the average TF-IDF score of its valid component words (words without a spcific self-defined stopword set are considered “valid”).

One variation here is that we modify the original TF-IDF formula as following:

-

Compared to orginal tfidf formula, here we add two exponential power on both the term and term. The basic ideas are:

1) We want to increase the strength of penalization on terms that appear in multiple documents. Intuitively, the patent term we are extracting is a domain-specific term that best describes the main focus of document, and thus is supposed to appear in other documents as few times as possible. As a result, β is set >=1 to enhance the influence of term;

2) We want to penalize the terms with exceeding high term frequency. During the tests, we discover that terms extracted with exceedingly high score are indeed topic-related terms but not the exact patent name we desire. For instance, in *Amazoncom Inc v Barnesandnoblecom Inc*, “shopping cart” ranks as the top term, which is indeed topic-specific and related to the exact patent name “1-Click(r)”. Intuitively, we need alleviate the phenomenon that a frequent term “drowns” the true patent terms. As a result, α is set <=1 to reduce the impact of term;

The results of permutating α and β are shown in Part 7.

5.4 Evaluating candidate noun chunks with logistic regression

We incorporate the benefits of supervised machine learning approaches into this task by applying a logistic regression model. For each candidate noun chunks, we extract the following features:

* The offset from the beginning of documents
* The tfidf score of its root word / The average tfidf score of its valid component words
* The POS tags of its root word and its head word in dependency tree
* Whether the term contains the trademark “(r)”
* Other word shape information (e.g. capitalized? contains digits? etc.)

After extracting these features, a logistic extraction model is trained using these features as data and annotation introduced in Part 4.4 as labels. The size of training set is 154 documents, whereas 66 documents are used as valid and test set, 33 for each. With this model, we are able to evaluate a confidence score for each noun chunks to be the true patent term.

6. Extracting key sentences

Using spaCy pipeline and a customized sentencizer, we are able to obtain the sentence set. Similar to extracting patent terms, for each sentence we evaluate a confidence score for it to be the key sentence.

6.1 Logistic regression

A logistic regression model using featurized sentences as input is implemented for this part, the features of a sentence include:

* Dicretized sentence length (dicretized range: 0-5, 5-10, 10-20, 20-50 and 50+)
* The
* Average TF-IDF score of sentence’s valid component words
* Maximum value of TF-IDF scores for all noun chunks in the sentence
* Whether the sentence contains trademark “(r)”
* Whether the sentence contains patent id numbers
* Whether the sentence contains specfic verbs (e.g. “claim”, “describe”) which serve as indicators of target sentence.

The training and predict processing is similar to Part 5.4 and the result is presented in Part 7.

6.2 CNN method

We have also tried Convolutional Neural Network (CNN) for text to extract key sentences. The structure of CNN used is illustrated in the Fig. (?).

(插图 CNN结构)

|  |  |
| --- | --- |
| filter size | filter number |
| 2 (for bi-grams) | 10 |
| 3 (for tri-grams) | 5 |
| 4 (for 4-grams) | 2 |

Firstly, each sentence is run through a embedidng layer (using pretrained gloVe embeddings【引用？】) and is transfromed into a sequence of word embeddings. Next, a convolution layer is added and filters of various sizes are designed for detecting bi-grams (e.g. “… patent claims …”), trigrams (e.g. “… the assignee of …”) and 4-grams. After convolution, a maxpooling layer is used to reduce the hidden representation vector into a fixed size. Finally a dense layer and is added to calculate the final score for a sentence.

As discussed, 154 documents are used to train, 33 for validation and 33 for testing. We perform a early stopping strategy to reduce the effect of overfitting.

From the result presented in Part 7, we conclude that CNN is able to extract indicating patterns for a sentence to be a key sentence. However, in general CNN experiments we try suffer from overfitting problem and since the training set in this project is relatively small, the result is relatively unstable and varies a lot across different train/valid/test sets.

7. Metrics and results

8. Conclusion