DOCUMENT CLASSIFIER

A program that takes a list of document locations and outputs two tables: a classification and summary table of all documents in the list and a classification and summary table of all the paragraphs in the documents in the list

fasttext is licensed under BSD-3

THIRD DRAFT

Preparation

0. Build corpus of sentences and labels of what they are about

- maybe by labelling paragraphs or documents that one is sure discuss only one topic, extracting sentences, and applying label to each sentence

1. Use pretrained ones: https://github.com/facebookresearch/fastText/blob/master/pretrained-vectors.md

2. Train fastText sentence classifier

Program structure

*Preparing data*

0. open document

1. preprocessing document:

- read in document

- change to lowercase all letters after a dot

- remove multiple consecutive spaces

- separate into sentences

*Document classification*

2. apply fastText classifier to text into categories:[targets], mixed, other

option:

-everything that follows serves the purpose of making classification more responsive to our application domain, of which we do not have enough data to

3. apply classifier to sentences in the text

4. agglomerate sentence classifications in some way to ensure that the number of ‘sentence classifcations’ in the output is the same (e.g. 100). E.g. divide text into 100 chunks (option:first into paragraphs, then, if necessary, further), each sentence assigned to one chunk, majority voting within chunks. If the document is shorter than 100 sentences, some sentence classifications are duplicated/tripled, etc.

5. add, for each IID, the number of instances detected in the document to the table

6. train classifier over the table with document category, chunk categories, #IIDs, etc

optional: classify paragraphs and croiss reference with likely IIDs to improve prioritisation of parts of documents for human parsing of privacy relevance

the hope is that by including near-sentence level information, fastText misclassifications can still be detected in the step that follows. If there is no information to be gained from these additional variables, the classifier in 6. can just ignore them. Including too many can result in overfitting though

SECOND DRAFT

Preparation

Pre-train word vectors (x4, for each language. Or in parallel?) - use FastText

Find and categorise a sufficient number of paragraphs. Important: also from a wide variety of non-target text types

Test different (parametric) models on the annotated paragraphs

Program structure

*Preparing data*

0. open document

1. preprocessing document:

- read in document

- change to uppercase all letters after a dot

- remove multiple consecutive spaces

- separate paragraphs by \n

- lemmatise words (?)

- separate paragraphs (alternative upper bound per unit?)

*Summarising documents*

2. a) pass each paragraph to function to

- retrieve word vector:

- if none exists: find morphologically closest matching word that has a vector, and retireve that vector

- take average of vectors

b)do as in 2.a) and then

- compute cosine distance between all pairs of consecutive paragraphs.

- Take the k (2?) biggest distances as boundaries between k+1 blocks in the document.

*-* Take average of paragraphs in each block.

c) run a RNN over each paragraph.

- take k (4?) snapshots of hidden state

- character or word-level RNN

*Document classification*

3. predict paragraph category (based on vector or RNN profile): ridge logistic regression? SVM?

*Output*

4. append paragraph number, document, paragraph location, paragraph vector, probabilities of belonging to each category, to table of paragraphs

(IDEALLY, regex and NER results should be put in the same table, for each paragraph in which they appear)

*Document classification*

5. when all paragraphs in a document are processed: categorise document as of a category if e.g. >80% of paragraphs of a certain category, classify document as that category, otherwise mixed. Append to table: document, document category, for each category, proportion of paragraphs in that category, mean of paragraph vectors, document location

6. append paragraph table to a table file (maybe divide into multiple tables at a certain size)

FIRST DRAFT

2. check whether there is a word vector for each word in the text:

-if not, take max similarity score to a string that has a vector

3. create bag of vectors for each paragraph

4. compare paragraph vectors to profile paragraph vectors

5. K(maybe 3?) nearest paragraphs categorisation for each paragraph

alternatively: train an alorithm

6. If a threshold (>80%) of paragraph categorisations is reached for a document, classify that document in that category, otherwise classify as mixed: [any category that has over 20% of paragraphs]

7. output one table of documents, document classification, and one of paragraphs, paragraph classification, document, location in document.