README.md

OCR Post Correction

Post OCR correction through local text reuse detection (work in progress!)

Structure of the Project

The code is structured into folders which are all parts of the golang module "postCorr". Each folder is a package of the folder's name.

Alignment

Contains functions to perform local alignment(smith waterman algorithm + a faster heuristic algorithm) and suboptimal alignments between two regions of text. Also includes the code that 'clusters' alignments together to produce multiple witnesses to correct a passage.

Fingerprinting

Contains code for making representation of documents/strings into fingerprints - sets of hashed numerical values.

Readwrite

Functions that help convert from OCR format to our common document format and back again. Plans to support some commonly used formats like ALTO and PAGE.

Correction

Contains implementation of a consensus method to produce a common output from an alignment cluster.

python-utils

Contains python code for implementing a language model, and

Running the Code

Basic Setup With Golang

All instructions are for mac/linux. I can't guarantee there won't be issues on windows, but nothing is hardcoded which prevents windows use.

Installing and Building

- Install go: https://golang.org/doc/install
- Build the code with `go build`, this will create an executable called **postCorr**

Providing Datasets

- The main dataset must be provided with the flag *input*, and should consist of exclusively .txt files. The program will look for text reuse between any 2 of these files. The dataset can be split into any number of sub folders, and all utf-8 unicode characters are valid file contents. The program will avoid changing any newline, tab or space characters.
- A gold standard dataset for benchmarking can be provided with the flag *groundtruth*. This must have exactly the same file/folder structure as the input dataset. Providing this dataset automatically means an evaluation is run by the program after other stages have been completed.

Output

- If the *write* flag is set to true (default true), a folder *corrected* is created at runtime containing the new versions of the files changed by the program (unchanged files are not written).

- If the *logging* flag is set to true (default true), a folder *logs* is created at runtime containing json log files.
- Note: The *logs* directories should be renamed before the program is re-run, if the user wants them to be preserved and not overwritten. The *corrected* directory will not be overwritten, the program will instead create directories *corrected1*, *corrected2*... etc.

Command Line Flags

- The executable must be run with the relevant command line flags. Command line flags are set with a dash (-) and equals (=), for example:

```
./postCorr -input=datasets/dataset -fp=winnowing -k=20 -affine=true
```

- To view flags and their descriptions, run `./postCorr -h`

A table of command line flags and their interactions can be found below.

Flags	Datatype and Possible Values	Default Value	Description	Interaction With Other Flags
input	string	None	Path to directory containing OCR dataset, this is the only flag which must be set.	
groundtruth	string	None	Path to directory containing groundtruth dataset. If set, the program performs analysis on how well it performed when making edits to **input**.	
write	boolean: 'true' or 'false'	true	Whether or not to write corrected output to the 'corrected' directory	
logging	boolean	true	whether or not to generate log files in the 'logs' directory	
candidate_method	string: 'modp', 'winnowing' or 'minhash'	'modp'	Method of candidate selection - 0 mod p, winnowing, or minhash, as described in the paper	The flag **k** should also be set to a preferred value. Choosing 0 mod p means the flag **p** should also be set. Choosing winnowing means the flag **t** should also be set.
jaccard	string: 'weighted' or 'regular'	'weighted'	The type of jaccard index used for candidate selection	When setting **fp** to minhash, there is no implementation for the weighted jaccard, so the program reverts back to using the regular non weighted method.
k	integer: > 0	7	Length of k-grams used for fingerprinting in the candidate selection process	
t	integer: >= **k**	15	Size of winnowing threshold *t* when using winnowing	Must be greater than or equal to **k**
candidate_proportion	float: > 0 and <= 1	0.05	The proportion of pairs to select as candidate pairs for alignment. This will be the top proportion of scorers based on the score given by the candidate selection algorithm	

Flags	Datatype and Possible Values	Default Value	Description	Interaction With Other Flags
num_aligns	integer > 0	2	The number of local sequence alignments to attempt between two candidate documents. Higher numbers should help find multiple separate reused passages, but takes more time.	
align_threshold	integer >= 0	1	The minimum score of a previous local alignment to continue finding more alignments between a given pair	Helps save time if **num_aligns** is set to a higher value.
affine	boolean	false	Whether of not to use affine alignment	
fast_align	boolean	false	Whether or not to use heuristic alignment	**band_width** should be set
band_width	integer	200	The heuristic algorithm's dynamic programming band width *w*	
use_lm	boolean	false	Whether to use a language model - this requires running additional python code as described below	**lm_threshold** should be set
lm_threshold	float	0.1	The threshold p_t - the language model prevents edits on words that have $>p_t$ probability score from taking place.	
insert_delete	boolean	true	Whether to use insert/deletion to correct errors as well as substitution, as laid out in the paper	The flags **l_delete** and **l_insert** should be set
l_delete	integer > 0	2	The maximum length of character sequence that the algorithm will attempt considers an erroneous deletion in consensus vote.	
l_insert	integer > 0	2	The maximum length of character sequence that the algorithm will attempt considers an erroneous insertion in consensus vote.	

Using Additional Python Based Parts Of the Codebase

Installing dependencies

- Install a new python 3.7 environment
- Within the environment, install dependencies with `pip install -r requirements.txt`

Running the Language Model Code

nltk is the relevant dependency here, and requires some extra steps to get data for training.

- run the following inside a shell of the python environment

```
>>> import nltk
>>> nltk.download('reuters')
>>> nltk.download('brown')
```

- Next, in the base directory of the code, run the model with `python python-utils/language_model/language_model.py` , and wait until the code finishes training and displays a server ready message.

- We can now run the **postCorr** executable with the flag `-use_lm=true`

Running the Browser Based Visualisation Tool

The browser based tool relies on the a run of the main program being completed with the *logging* flag being set to true. After this, the usage steps are as follows:

- Run the python server with `python python-utils/visualiser/server.py *input-directory*` , where *input* directory is the name of the directory containing the original OCR data.
- To view files in the browser tool, navigate to the page at `localhost:3000` , which displays a list of the files edited by the tool, and click on each file's link.