Vito D'Orazio

Documentation for MID5 Document Classification

June 5, 2016

**Notes:**

The scripts are written in python2, python3, and perl. Below, python3 denotes python3, and python is python2.

Some of the scripts are interactive, and some require arguments.

To use the SVMB classifier, you'll need SVM-Light (http://svmlight.joachims.org/)

The "MID" format is referred to several times below. This format is as follows, but note that not all metadata fields are included:

Key: 20110402-1-639-AP\_1-2April\_2011\_LN\_NP2.TXT

Date: 20110402

Source: Associated Press Online

Dateline: LITTLE ROCK Ark.

GoogledDateline: Little Rock, AR, USA

AmericanDateline: True

Countries: ('UNITED STATES OF AMERICA', 2)

>>>>>>>>>>>>>>>>>>>>>>

Officials say a Delta Airlines plane landed safely at Little Rock National

Airport after striking a flock of large birds.

Airport spokeswoman T.J. Williams says the pilots of Flight 5087 from Atlanta to

Little Rock reported hitting the birds Friday afternoon about 20 miles southeast

of the Arkansas airport. She says the pilots were able to land the plane without

incident.

Williams says Delta officials plan to examine the plane and decide if anything

needs to be done. She says officials don't know what type of birds the plane

hit.

<<<<<<<<<<<<<<<<<<<<<<

---------------------------------------------------------------

**Starting with LN Downloads**

***docdelim.py***

$ python3 docdelim.py

This script is interactive. Follow the instructions. The regex for delimiting LexisNexis documents is hardcoded in the script, so just enter "LN" when prompted.

Writes filedelim.json, a file that contains a list of all the text files, their delimiters, and the number of docs.

***docprint.py***

$ python3 docprint.py

This script expects filedelim.json as input.

It extracts as many metadata fields as recognized from the documents. Note, if the documents aren't from LexisNexis, this will likely not find any metadata and everything will end up in the text (which may or may not be ok).

This script asks for 'json' or 'MID' format for the documents.\* output file.

This script also writes a tab-separated file that contains all metadata, including the unique identified attached to each document. This summary.tsv file does not contain the document text.

This will take a while to run, depending on how many documents there are. Maybe 10 minutes for 270,000 LN documents.

This writes "documents.txt", which is the MID-formatted file that includes all the documents.

**Classification**

The documents.txt file is now ready to be classified. Bill has set up a classifier that takes this 'MID' format as input.

**To continue with Bill's classifier:**

<copy over Bill's instructions here once the classifier is ready>

**To continue with PreText and the SVM-B classification:**

***text\_tokens.pl***

$ perl text\_tokens.pl CountryCodes.111214.txt stopwords.txt porter.pm

The three arguments that are passed to text\_tokens.pl are Schrodt's 'CountryInfo' file, a list of stopwords, and a stemming algorithm. Ensure that each of these three arguments correspond to a file in the same directory.

This file takes as input documents.txt. Note that a documents.json file will not work.

This skips over all metadata, and removes stopwords and named entities from the text. It also stems all words.

This outputs: tokens.txt and spreadsheet.tsv. Note that spreadsheet.tsv is (I think) the same as summary.tsv from docprint.py.

***term\_doc\_dict.pl***

$ perl term\_doc\_dict.pl 10 NTFSVM vocabulary.txt

The first argument, here it is 10, is a document frequency threshold. 10 is probably fine. The second argument contains the type of feature representation for the output format. To continue with the SVM classification, enter NTFSVM.

The third argument is the vocabulary produced by the training set. This may not be necessary for all classifiers, but it is for the SVM Light format. This file here is vocabulary.txt. Each line contains a token and a number separated by a tab. This list of tokens must be in the same order as the list of tokens that was used to train the model.

This script takes tokens.txt as input, and outputs a file svm\_ntf.dat. This file can be read by SVM-Light, the software used to classify with the SVMB classifier.

**Classification with SVM-Light**

SVM-Light should be installed from: http://svmlight.joachims.org/

Next, use the SVMB classifier, which is mid5v0.model

$ svm\_classify svm\_ntf.dat mid5v0.model svm\_ntf.predict

Next, mv the \*.dat and \*.predict files back to the directory with the documents.txt (and other) files.

***merge\_results.pl***

$ perl merge\_results.pl 0 svm\_ntf2011.predict svm\_ntf2011.dat

Note here that the 0 corresponds to the cutoff point using the SVM-Light score.

This script requires "MID.ISO.CODES.txt" and "documents.txt" to also be in the same directory.

This outputes four files: sorted\_pos.txt sorted\_pos.tsv sorted\_neg.txt sorted\_pos.tsv. These files are self-explanatory: the .tsv files are the summary files, and the .txt files are the full "MID" format text files.

**After Classification**

After classification, however that might happen (SVMB, or Bill's classifier), there will be a file that is "MID" formatted and contains the documents classified as positive.

There are three steps for adding metadata and further trimming documents:

1. remove documents with datelines in the United States other than New York, NY and Washington, DC (googleddateline.py)
2. remove documents without two states mentioned (actorcount.py)
3. remove exact duplicates (MIDtoDB.py)

***googled\_dateline.py***

$ python googled\_dateline.py

Assumes that a file called "sorted\_pos.txt" exists in directory

This adds two metadata fields: GoogledDateline (formatted address), and AmericanDateline (true/false)

Note that this uses Google's geolocator, which is limited to 15,000 per day, so be careful with this one.

This scripts writes "sorted\_pos2.txt" as output

***actor\_count.py***

$ python actorcount.py

This adds one metadata field: Countries: (string with country followed by number of mentions)

Takes as input Schrodt's CountryInfo file, "CountryInfo.120116.txt" and "sorted\_pos2.txt"

This script writes "sorted\_pos3.txt"

***MIDtoDB.py***

Note that this script first requires the user to create a database. The script will then define a table and add it to the db.

To create the database:

$ sqlite3 middb.sqlite

drop table if exists mid;

create table mid (

Headline string,

Key string,

Date string,

Source string,

Dateline string,

Byline string,

Collection string,

Language string,

Subject string,

Organization string,

Geographic string,

Loaddate string,

Pubtype string,

StoryText string,

GoogledDateline string,

AmericanDateline boolean,

Countries string

);

.exit

Then run MIDtoDB.py

$ python MIDtoDB.py

To use the db to delete all documents that are exact duplicates:

$ sqlite3 middb.sqlite

create index x\_key on mid(key); -- this is indexing the table

create index x\_storytext on mid(storytext);

select key, count(key) as how\_many

from mid

group by key

having count(key) > 1;

to count the records:

$ sqlite3 middb.sqlite

select count (\*) from mid;

delete from mid

where key not in

(

select min(key)

from mid

group by

storytext

)

This bring us from 6410 to 5605 for 2011.

Note that much of what is written is redundant. I highly suggest that when this process is complete, delete the excess files and make a record of what was done.