Early Warning System (EWS)

The file `EWS.py` contains wrappers for multiple functions that make up the core of the early warning system.

`avg_risk()`

The `avg_risk()` function calculates the average of the syntax score of the input corpus. The syntax score is calculated on the basis of a pre-scored database/csv of words according to the criteria that "risky" words are given positive scores and "safe" words are given negative scores.

The syntax score is calculated by simply summing the individual score of each word in the corpus. Note that words that don't exist in the database are *removed* from the calculation so that the average is not unnecessarily reduced by the 0 variables.

`extract_phrases()`

This function extracts key entities and phrases from the corpus using spaCy.

`risk_factors()` and `risk_mitigators()`

The pair of functions are analogous. That is, `risk_factors()` deals with sentences in the corpus which have a positive risk score, whereas `risk_mitigators()` deals with sentences in the corpus which have a negative risk score.

These functions return a list of sentences paired up with their individual risk scores. The sentences are selected as per the above described criteria.

`main_ews()`

`main_ews()` combines all of the functions above into a single callable one for ease of usage. This is also the interface to the API as described in `api.py`.

Rules-based Syntax Scoring

The files `scored.csv` and `tagged.csv` are used for rules-based syntax scoring and form the backbone for the calculation of risk scores.

`tagged.csv`

This file is created by the `pos_tagger.py` and is used as the backbone for the `scored.csv` database.

`scored.csv`

This database is created by manually adding heuristically defined scores that form for the basis for the syntax and contextual scoring algorithms.

Input and formatting guidelines

Articles can be fed into the system in either one of two ways:

- Through a live newsfeed (**newsstream_rss.py**, **newsstream.py** or similar)
- Row-wise torugh a csv (see demo.csv)

It is recommended that headlines be cleaned for punctuation and stopwords before input or should be preprocessed using the cleaning methods provided in `textcleaner.py`

Using live newsfeeds

There are two default options for the use of online newsfeeds:

Fetching data from news API (newsstream)

Requests through CLI interface
The default implementation of newsstream makes use of a CLI based interface as a means of sending requests the interface can be accessed

by running the program prior to the use of EWS (either as an external command or as an import statement within a script or notebook)

Requests through: `newsstream(req_h)`

Alternatively, requests for articles can be made in the format of the class `req_h`

i.e: `(sources = "source",q ="query", sortby = "relevancy" , pageSize = max page size is int(100))`

headlines and articles recieved can be accessed either through the `list`variables `titles` (headlines) and `bodies`(the articles themselves) or from `newsstream.csv`

RSS feed parsing (newsstream_rss)

RSS feeds can be parsed into the system by calling the function `fetch_feeds(sources)` with a list of feed URLs which returns a list of headlines or titles.

Using CSVs

CSVs can be directly parsed by accessing the electron interface or through the built in csv library.

`context.py`

This deals with all the contextual analysis of a corpus. The backbone for this is the recursively-called `maxiter()` function.

`context()`

This function acts as a wrapper and container for `maxiter()` such that it accounts for live updates.

`maxiter()`

`maxiter()` is a function that traces the maximum probability path in a probability matrix of all words in a corpus. In order to not backtrack, however, the function is designed to check if it has visited a node before using the `ignr_list` variable. This is achieved by setting the probability values of the visited nodes as 0, i.e. the least probability possible.

`rescore()`

`rescore()` builds on top of `context()` and `maxiter()`. It returns an average of the risks of the maximum probability path. This is then the considered to be the “new” score for the word in question.

`context_sentence()`

A simple wrapper for sentences instead of words to make using the `rescore()` function natural. Note that the final output is normalised on the basis of the original risk score of the sentence.

`context_docs()`

Extending `context_sentence()` to be easily used with a corpus.

`return_cooc()`

Returns the co-occurrence matrix of the words in the corpus

`bn_probs()`

Returns a probability matrix based on word co-occurrences, froms the basis of the contextual scoring process