medke

Domain knowledge entity extraction for scientific documents in biomedical science.

##Dependencies

Language(s) Used: Python 3.7

Python Modules:

1. sklearn-crfsuite
2. nltk

3. scipy
4. numpy
5. scikit-learn

##Description

The medke codebase is a repurposed program based off of the codebase semeval2017 (https://github.com/SeerLabs/semeval2017) with the code being updated from Python 2 to Python 3. The purpose of this program is the extraction of DKEs from text. At the time of writing this document, the DKE extractor program consists of 20 executable python files.

The .py file interactions can be divided into three groups with one group being all .py files found in the `crfModel` folder , the second being any file within the `svmModel` folder, and the last being any .py file outside the previous mentioned folders. For brevity, the groups will be labeled crfModel, svmModel, and Other respectively.

—crfModel (`ClassifyCRFtoANN.py` , `convert.py` , `CRFNER.py` , `DataExtraction.py` , `Domain-Entities-extraction-given-JSON.py` , `FeatureExtraction.py` , `PhraseEval.py` , and `semeval_to_BIO.py`)—

Note: All testing and training data can be found in the `medicalData` folder

1. Using `semeval-to-BIO.py` , files from the `testData` and `trainData` folders are converted into BIO-style tags for classification. The script returning tokenized terms with BIO notation to identify key phrases and offset boundaries. All outputted files are stored in the `formatBIO` folder as `*_output.txt` .

2. For each original file, three fields were taken, the word (`token`), the tag (`B_KP` , `I_KP`) and the index. The files in the folder `formatBIO` under `formatBIO/testBIO` and `formatBIO/trainBIO` are fed into `convert.py` in accordance to the `testList.txt` and `trainList.txt` . The converted files are stored in `convertedBIO/test` and `convertedBIO/train` respectively.

3. These converted files were combined using `awk` [`awk '{print $1"\t"$2}' medicalData/convertedBIO/train/*.txt > medicalData/combinedTrain.txt`] to produce the training data (`combinedTrain.txt`) and test data (`combinedTest.txt`). Note these files contain only two columns (tab separated): the token and the tag. The text files can be found in the `convertedBIO` folder.

4. `CRFNER.py` trains a linear chain CRF model and outputs the model as a pickle file (`linear-chain-crf.model.pickle`). You can do a hyper parameter optimization on the training data.

5. `DataExtraction.py` and `FeatureExtraction.py` contains the code to prepare the data and extract features. Both are used by `CRFNER.py`. `CRFNER.py` can also use `PhraseEval.py` if the user wishes to view how the phrase extraction works. Note the pos tags are extracted during the data extraction step.

6. `ClassifyCRFtoANN.py` uses the trained model to predict the token classes and output the predicted ann file. An example input to the code is a file in `medicalData/convertedBIO/test`, e.g., `python ClassifyCRFtoANN.py medicalData/convertedBIO/test/abbott-2006-01.txt`. Output is `medicalData/predictedANN/abbott-2006-01.ann`. Calls upon `DataExtraction.py`, `FeatureExtraction.py`, and `PhraseEval.py` to accomplish this.

7. `Domain-Entities-extraction-given-JSON.py` extracts DKEs from a given JSON through the use of the trained linear chain CRF model (`linear-chain-crf.model.pickle`) and outputs a file containing the DKEs. An example input would be any file found in `DKE-tests/json`, e.g., `python Domain-Entities-extraction-given-JSON.py DKE-tests/json/borchers-2015-06.txt`. Output is `DKE-tests/DKE-output/borchers-2015-06-DKE.txt`.

—Other (`annParser.py`, `config.py`, `eval.py`, `gen_keyphrase_*.py`, `scix_eke-*.py`, `scix_test.py`, `SVM.py`, and `TxtTrainParser.py`)——

1. The three `scix_eke-3.*.py` scripts must be supplied with the arguments TESTDIR and OUTDIR in order to execute. The TESTDIR supplies the directory containing the .txt files to be tested and the OUTDIR supplies the directory containing the output .ann files. The three scripts take the supplied .txt files, run through them for keyphrases (specifically NP), and output them as .ann files. For `scix_eke-3.2.py` and `scix_eke-3.3.py`, a third argument N is necessary which specifies the the N chars before and after the keyphrase extracted. Note that each script calls upon a specific `gen_keyphrase_*.py` script.
    - `scix_eke-3.1.py`, `scix_eke-3.2.py`, and `scix_eke-3.3.py` call upon `gen_core_keyphrase_core_stanford.py`
    - Note the output directory is `scixOutput`
2. The script `scix_test.py` serves to test a single file for keyphrase generation instead of a whole directory. It requires a single text file to be supplied for its argument and calls upon `gen_keyphrase_core_bounds.py` when looking for the nounphrases.

3. The script `TxtTrainParser.py` is a more detailed version of the `scix_eke-*.py` scripts. It parses through the supplied training data to identify NP, PP, and VP types while also including phrase labels. The script `annParser.py` is used heavily by `TxtTrainParser.py` to help accomplish this. It should be noted that `config.py` contains the input and output directories in addition to the corenlpserverurl.

4. For `TxtTrainParser.py` and certain `scix_eke-3.1.py` to run properly, a Stanford CoreNLP server must be running on the system. The CoreNLP packages can be downloaded here (http://nlp.stanford.edu/software/stanford-corenlp-full-2018-02-27.zip). After unpackaging the zip, just cd into the stanford-corenlp-full-2018-02-27 directory and run the command below to start the server.

    - ```
      java -mx4g -cp "*" edu.stanford.nlp.pipeline.StanfordCoreNLPServer \ -preload tokenize,ssplit,pos,lemma,ner,parse,depparse \ -status_port 9000 -port 9000 -timeout 15000 &
      ```

5. `eval.py` serves to calculate the P, R, F1, and Macro F when supplied with a folder containing gold standard .ann files and a folder containing prediction .ann files. The folders are `crfModel/medicalData/testData/anns` and `crfModel/medicalData/predictedANN` respectively.

—svmModel(`CreateNegative.py`)

1. `CreateNegative.py` serves to create the samples necessary for the SVM classifier. The text file `annList.txt` is used to cycle through all 100 manual annotations and all 100 eke annotations to create the negative annotations under the folder `svmModel/AnnotationData/Negative/` .

2. The resulting 100 negative annotation file are combined using the command `cut -d$'\t' -f 3 Negative/*.ann > combinedNegAnn.ann` to create a single file with all the negative DKEs. The same is done with all manual annotations to create a `combinedPosAnn.ann` file.

3. The `combinedPosAnn.ann` and `combinedNegAnn.ann` are then joined to create `PosAndNegAnn.ann` which serves as the sample for the SVM classifier.

##To Do

- Complete the SVM classifier
      ```