The Guide to the Code

Nomenclature

Chief data set: The first data set that is compiled, including all observations, regardless of whether they wind up belonging to the training or test set. Its only columns are, on the one hand, the one consisting of the preprocessed texts, and, on the other, the three different response variables being taken into account.

Feature-enriched data set: The last data set, with all the information that is added to the chief data set as features.

Feature dictionary: The dictionary with the names of the feature-enriched data set's columns—as values—linked to the type of transformations that were applied to the texts amounting to the chief data set's observations to generate them—as keys.

Training set: All the observations from either one of the data sets that correspond to the texts downloaded with the code contained in the scripts stored in the Scrapers and Downloaders folder.

The target variables:

- Literariness, which is a binary-class type of label denoting whether an observation was or not retrieved from a website hosting works of literature
- Source Cat, which is a multi-class type of label indicating the categories assigned to the observations' sources
- Source, which is a multi-class type of label referencing the names of the websites the texts forming the chief data set's observations were extracted from

Test set: The observations corresponding to Amazon-movie and TV reviews, on the one hand, and Yelp-business reviews on the other.

Secondary target variables:

- Helpfulness: Is a continuous target variable corresponding to the ratings Amazon-movie and TV reviews received from its online readers. It equates to the percentage of positive feedback divided by the total amount of feedback each review received, so long as it was rated by at least 25 people.
- Usefulness: Is a continuous target variable of integer values corresponding to how Yelp-business reviews scored for the category representing how useful they were deemed by its online readers.
- Funny: Is a continuous target variable of integer values corresponding to how Yelp-business reviews scored for the category representing how funny they were deemed by its online readers.
- Cool: Is a continuous target variable of integer values corresponding to how Yelp-business reviews scored for the category representing how cool they were deemed by its online readers.

Pragmatrix Main Script

This script contains the code to load all the necessary paths, variables, dictionaries and data sets that will be needed to execute the code contained in most of the other scripts. It is also where the chief data is split into training and test set. Additionally, it displays the options users have to choose from in order to train, first, the different classifiers that show the importance of the engineered features in determining the observations' physical, while virtual, origins, and secondly, the classifier I have called 'the terminator', which is designed to predict the 'literariness'-target variable, so that a correlation between how similar, according to it, a review on the Internet is to pieces of literature and how useful it has been regarded and rated as by its online readers can be established.

Scrapers and Downloaders Folder

Short fiction Downloader: Downloads short stories from the literary websites Every Writers Resource, Electric Literature, Arabian Stories, the American Short Fiction and the American Literature online magazines, the Waccamaw Journal of contemporary literature, and the New Yorker Fiction Magazine.

Long Fiction Downloader: Downloads from the online library of the University of Adelaide works of literature, mostly novels, that have been widely-acknowledged as such, regardless of the language their original versions were written in and the century they date back to.

Ling Blogs Downloader: Downloads articles from the Oxford Dictionaries Blog and the Collins Dictionary Word Lovers' Blog.

News Downloader: Downloads news articles from the New York Times, the Washington Post, the Independent, the BBC, the Guardian, the Los Angeles Times, the NY Daily News, the San Francisco Chronicle, the Houston Chronicle and the Times of India.

Wiki Downloader: Downloads randomly-chosen Wikipedia articles.

Collocations Dict Scraper: Contains the functions to scrape the Online Oxford Collocation Dictionary anc convert the downloaded collocations into regular expressions that try to encompass the different morphological variants derivative words stemming from those subject to inflection represent, so that they may match as many correctly-built collocations in my data set's observations as possible. I have decided against 'lemmatizing' the collocations I have downloaded, because I have chosen to adopt a more restrictive approach and disregard the collocations that might have been built correctly, but which my system didn't detect, rather than count as collocations word groupings that aren't, despite resembling them, in the belief that the work of authors who write taking collocations into consideration will most likely present several of them. This script also includes a function to download phrases and idioms from the English Oxford Living Dictionaries.

Collocations Dict Downloader: Builds a dictionary with all the collocations, phrases and idioms of the most frequently occurring words in the English vocabulary. The keys do not only correspond to the words' roots, but also to their derivative forms. In order to include the conjugated verb forms of irregular verbs, not only as keys of the dictionary, but also inside the regular expressions replacing the irregular verbs present in the downloaded collocations, I previously downloaded a list with the most common irregular verbs of the English lexicon.

Cleaner

This script removes all the non-strictly discoursive elements from the texts I have downloaded with the several functions to be found in the scripts contained in the Scrapers and Downloaders folder. I have tried to, in as far as possible, accomplish this cleaning task by writing one single function that works for all texts, so that I could be sure that they would, at least, end up equivalently clean. It is also this script that has the code to build the chief data set, which is what is done after filtering the documents through the function called 'docs normalizer', to be found inside the 'prepro funcs' script, which cuts the texts into 3000-to-6000-long chunks and randomly chooses up to five excerpts from each observation, while discarding all texts with either three consecutive sentences reading exactly the same or sentences longer than 3000 characters.

Keras NLI

The script containing the code—for the most part, copied from the one Stephen Merity has posted on his Github page (https://github.com/Smerity/keras_snli)—to train a neural network with both the Stanford Natural Language Inference (SNLI) Corpus and the MultiGenre NLI (MultiNLI) Corpus, both by Bowman et alii, which are corpora of sentence pairs that have been sorted according to what they entail into three categories—contradiction, consistency, and neutrality, which is the one that covers the cases where neither one could be established. Stephen Merity's neural network is meant to be trained on the aforementioned corpora and, subsequently, applied to the word embeddings that comprise the texts forming the chief data set's observations, so that the predictions it gives can be incorporated to the feature-enriched data set as additional features.

Feature Trove

This script contains the class to call the classes and functions that transform the texts constituting the chief data set's observations into numeric features and builds both the feature-enriched data set and the feature dictionary, as well as the function that allows users to make sure that all the necessary data sets, plus the feature dictionary, are built, if they hadn't been built previously, and are loaded into the working directory. It is called inside the Pragmatrix Main script.

Prepro Funcs

get proxy, which is a function used in the scripts inside the Scrapers and Downloaders folder to bypass IP address blocking when accessing the content of websites.

flatten, which is a generator used in the Collocations Dict Scraper script to flatten a list of lists of strings

docs normalizer, used in the Cleaner script and explained in the section set aside for its description.

Classy N Funky

This script contains all the functions and classes to generate the features that I want to add to the chief data set inside the Feature Trove script, along with the functions to prepare the input for the aforementioned code, which can be summarised as follows:

Bassicker: Is a function to compute whether the authors of the texts that constitute the chief data set's observations have tried to create dramatic tension by playing with the length of their texts' sentences—for instance, by juxtaposing a sequence of long sentences with a drastically shorter sentence—in order to try and drive their creations to develop a rhythm.

Spacy Parser and Spacyer: Are functions to parse the texts' sentences using Spacy and extract:

- the average amount of digits and punctuation marks
- the average smoothed log probability estimates of the words' types
- the average amount of words that could be verified that belong to the English vocabulary
- the part-of-speech tags the texts consist of
- the syntactic-dependency tags the texts consist of

Colls checker and collocater: Are, respectively, a function and a class designed to check whether the words in the chief data set's observations contribute to forming linguistic collocations and whether the collocations they might be a part of appear more or less frequently in the whole data set. This is achieved by looking up the words of the chief data set's observations in the collocations' dictionary I have compiled from the entries I have downloaded from the Online Oxford Collocation Dictionary and the English Oxford Living Dictionaries, and checking whether the sequence of words preceding and following them match any collocation from the dictionary.

TFIDFer: Is a class to build six different TFIDFs, two for each different input being fed into Scikit-learn's Tfidf vectorizer:

- The different inputs are:
  - the texts, before undergoing any transformation
  - the part of speech tags the texts equate to, according to Spacy
  - the syntactic dependency tags the texts equate to, also according to Spacy
- The TFIDFs constructed from each input:
  - Unigram TFIDF, with up to 600 word columns, without stopwords
  - 2-to-4-gram TFIDF, with up to 300 word-cluster columns, with stopwords

Doc2wecker: Is a class that, making use of Gensim's doc2vec algorithm, creates three 200-long vectors for each observation, one for the literal texts, another for the texts' part-of-speech-tags and a third one for their syntactic-dependency tags, representing the average predictability of the four surrounding tokens of each token in each document, given the company the tokens keep on average in the whole corpus. Additionally, it creates another three 200-long vectors, one for each target variable, that stand for the cosine distances between each observation and each label. This class is used for feature engineering as well as, inside the Pragmatrix Main script, as a classifier for target variable prediction, after taking the mean of the three aforementioned distances.

NLIyer: Is the function where I predict whether each pair of sentences in the texts of the data set's observations seem contradictory, consistent to each other, or neither, when compared to the sentences' pairs of the SNLI and the MultiNLI corpora. See the Keras NLI script's description for further details.

Classifiers

Log reg: A logistic regression classifier to asses how difficult it is to predict how literary an observation is. More specifically, I wanted to train an interpretable model that is known to be less prone to overfit than others, in order to find out which is the minimum amount of textual characteristics that need to be taken into account to obtain a reasonably accurate result in predicting whether a given text shares the characteristics of those retrieved from a place that hosts pieces of literature.

Rooted: A simple tree classifier to display the importance of all the added features for predicting the three target variables and show, as well, the decisions the model takes to rank the observations according to how likely it deems it is they belong to one class or the other.

Xgbooster: On the one hand, a xgboost to predict the three target variables with all the features but the cosine distance vectors generated by the Doc2wecker, to compare its performance with that of an LSTM that only takes the texts, without any transformations applied to them, as input. On the other hand, it can be used as 'the terminator', mentioned previously, with which I deploy the technique known as stacking by not removing but the cosine distance vectors generated by the Doc2wecker corresponding to the 'literariness'-label.

NN Prepper, LSTMer and Plot Confusion Matrix: The first function is meant to prepare the input for the two neural networks I am training. I also load Google's pre-trained Glove vectors, which is what I use as word embeddings. The second function is a simple sequential LSTM neural network to predict all the target variables with the texts of the feature-enriched data set's observations. The third function is meant to plot the confusion matrix of the LSTM neural network's output.

Dropping: Is a function to remove from the feature-enriched data set the columns that aren't supposed to be fed to the classifiers, because they are either target variables, primary or secondary, or not informative.