Documentation for Enhancing and Extending Metapy versioning and functionality

By: David Orona, Nikil Nair
Team Name: import teamname

Introduction/Challenges:

This project originated with the goal of improving the versioning and cross-OS functionality for metapy. We, however, have run into numerous issues trying to implement the latter, and have far exceeded our required 20 hour time commitment/person. As such we needed a new avenue to be able to contribute/improve the current state of metapy. To speak a little bit about some of the issues we encountered with the latter: there were numerous functional issues inherently branching from a lack of documentation and available notation to properly navigate the corresponding way to update/accommodate respective packages. Additionally all of the example code is relatively outdated and the majority of python functions are a wrapper over C++ functionality. After exhausting this option we decided to spend more time trying to figure out different ways we could update metapy so we settled on adding external package integration as an optimal way to improve the current functionality. We primarily settled on NLTK and similar NLP packages as a core way that we could use to improve metapy. A majority of the metapy code has inherent flaws and is outdated with regards to current methods. As a result we didn't find much need in utilizing its existing functionality. Below is a setup guide and a listing of the functions that we have implemented through NLTK and other packages.

Setup:

To setup this project it is recommended to follow the listed steps:

1. Please ensure that you have anaconda configured on your system. This will be useful since metapy will need to function through the utilization of python 3.6. You may install anaconda through this link: https://www.anaconda.com/
2. Once you have anaconda configured you will need to setup a virtual environment. To do this first enter: *conda create --name myenv36 python=3.6*
3. The above will create a virtual environment with python 3.6 installed. To run the instance you will need to enter: *conda activate myenv36*
4. With the above configured the virtual environment should be up and running, however, will have none of the respective packages installed. To deal with this it is important to install the necessary packages that are imported in the *nltk_additions.py* file as well as the *nltk_test.py* file . You can utilize *pip install …* for every package, and if your environment is properly configured you shouldn't run into any issues.
5. With the environment configured and the respective packages installed, you may now run our built in test suite, which performs and cross-checks functionality for ease of evaluation. All test cases should pass. You may also at this point freely explore the integrated functionality.

Functions

1. *inl2_retrieval*: We added the function *inl2_retrieval* as a means of abstracting some of the complexity that comes with creating a retrieval function. This is a primary part of MP

2.1 and is something that could have been much simpler to define and work through. As a result the latter was created to fulfill that role.

2. *Pos_tagging*: Part of speech tagging is an integral part of natural language processing and is a functionality that many models utilize within pre-processing. As such it is pertinent to most of the MP's that we have worked on and is integral within metapy.
3. *ndcg:* We added the *ndcg* function as a means of abstracting some of the complexity that we encountered from parameter modifications within MP 2.2. Discounted cumulative gain is a core piece of logic within NLP and as such it was added to expand functionality.
4. *naive_bayes_classifier:* The purpose of adding the *naive_bayes_classifier* function within metapy was to showcase the power of classifiers, as some were used within MP3.2, but more importantly it is to enable functionality that can be added on in future updates. As metapy grows as a package other classifiers and models can be enabled for functional purposes.
5. *stem_lemmatize:* The reasoning behind adding the *stem_lemmatize* function is due to the importance that stemming and lemmatization plays in the field of natural language processing, especially in pre-processing. One of the most important aspects of training a competent model is to ensure that appropriate data is used for testing/training. This function will assist with the latter.
6. *get_text_sentiment:* A very important concept within the field of NLP is sentiment analysis, as it is a way for a model to recognize the underlying meaning and context of a given piece of text. Sentiment analysis is a core piece of this field, and as such is an important function to have within metapy.
7. *max_entropy:* This function was implemented as a means of expanding the functionality enabled through classifiers. As can be seen earlier we have added a Naive-Bayes classifier which is relatively simple in its analysis, however, now we have also added the functionality for a max_entropy classifier which is a much more comprehensive and functionally beneficial model.
8. *analyze_collocation:* The primary reason we added collocation functionality within metapy is to expand its breadth of available tools. We wanted to incorporate functions that would also accomplish more involved tasks such as collocation. This function aims to identify the subtleties and nuances that can be often found within respective pieces of text.
9. *analyze_corpus:* Our reasoning for adding *analyze_corpus* as a core function within the metapy package has to do with improving pre-processing. Corpus analysis and proper training and testing data is imperative to ensure a model is working at its peak. This function enables that.
10. *generate_tree:* Through working through this class and exploring the vast complexities within the fields of text information systems and NLP as a whole, we recognized that it can often be challenging to visualize how words and different parts of speech interact with each other. As such we have added *generate_tree* to fulfill this role and bring better visualizations within metapy.

Consensus/Summary

This project implementation has two key parts. The first is the core functionality which can be found in *nltk_additions.py* and the second is the respective test suite which can be found in *nltk_test.py*. Both these files are located within the src folder inside the metapy folder. This project creates 10 new functions and respective pieces of added functionality for package integration. The logic and reasoning behind why we chose to create the specific functions that we did is to primarily ensure the MP's are more modularized and that the functionality is properly enabled. In other words most of the added integration functions have some connection to the MP's, and as a result we added the functionality as a means of potentially expanding metapy's functionality and capabilities, while also enabling an easier interface to work with. Metapy is a package that needs a lot of improvement to be able to be competitive with other top packages such as NLTK and Gensim, however, with incremental improvements such as this it becomes more user-friendly and has much-needed integrated functionality that most NLP tasks would require.