Enhance `MeTA Toolkit` and `metapy` Usability

Authors

Team DJ:

- David McGuire dmcguire@illinois.edu
- Jose Cols Matheus josec5@illinois.edu

Proposal

The goal of this project is to enhance the usability of the `MeTA Toolkit` and the `metapy` Python wrapper for future students of `CS 410 Text Information Systems`. Toward this end, several "pain points" have been identified, based on personal experience and the experience of our current cohort. This list overlaps with the *implied* motivation for some of the suggested improvements, but some of the motivation is entirely new and undocumented, thus far, providing a fresh perspective on the issue.

The primary pain point for `MeTA Toolkit` and `metapy` is the lack of build support. Concretely, a defect has existed on the main project for several years and has gone unaddressed (see meta-toolkit/meta #212), and this causes the OS-specific builds to fail for any Operating System for which there is not already a pre-built binary package. Because the list of pre-built binaries is small, static, and aging in the absense of *any project maintenance whatsoever*, the experience is getting worse, year-over-year. This project aims to address the build support problem by creating maintainable, automated build infrastructure on public, cloud-based systems (e.g.: GitHub Actions, CircleCI, Travis CI, etc.). Because the original maintainers have completely abandoned the project, and do not respond to requests for support, this will be done on a fork of both the main `MeTA Toolkit` repo (meta-toolkit/meta), as well as the `metapy` Python Wrapper repo (meta-toolkit/metapy). From there, the publicly-accessible infrastructure will be handed over to `CS 410` TA's for future maintenance. This build infrastructure will maintain a broad set of Python-version-specific builds that *continuously integrate* `MeTA Toolkit` C++ code and `metapy` Python code, to test for new defects as underlying dependencies change and evolve. Moreover, the build configuration will be extensible by future maintainers to new versions of Python as they are released.

Next, a secondary consideration when it comes to build support is whether students in `CS 410` should be expected to build C++ code for course Machine Problems that are, otherwise, exclusively in Python. In fact, requiring that level of technical support defeats the purpose of having a Python wrapper in the first place, and lags **well behind** the user experience for many other classes in the MCS and MCS-DS programs, which have Notebook-based, cloud-native assignments with automated grading integrated directly into the Coursera platform (e.g.: `CS 441 Applied Machine Learning`, `CS 598 Deep Learning for Healthcare`, etc.). That being said, providing pre-built binaries for all the myriad of OS's that students might choose to run is not practical for a number of non-technical reasons, not least of which that build infrastructure running License-based, Non-FOSS Operating Systems (primarily Windows and macOS) costs money, and the build infrastructure has no budget for that. Therefore, instead, builds will run on GNU/Linux, but will be supplemented with full Docker Container support via a publicly-hosted container repository (e.g.: GitHub Packages, DockerHub, etc.). Specifically, fully-containerized workflows will be built and documented, then published as first-class build artifacts.

Lastly, while this project can not hope to explore Notebook-based automation that is tightly integrated with Coursera, as is available with other courses in the program, Google Colab is fully within reach of all students, as an integrated Google account is part of the University IT offering, under the banner Google Apps @ Illinois. Therefore, as the third piece of usability enhancement, previously-published tutorials and demos that take the form of Jupyter Notebooks will be rebuilt from the ground up within Colaboratory, including links that allow any student collaborator to open the Notebook, save a local copy, and start editing. The hope is that this inspires future coursework maintainers to think outside of the confines of the single student's local machine, where the OS is unconstrained, and start redirecting limited support staff effort towards controlled and sustainable centralized environments, like Google Colab.