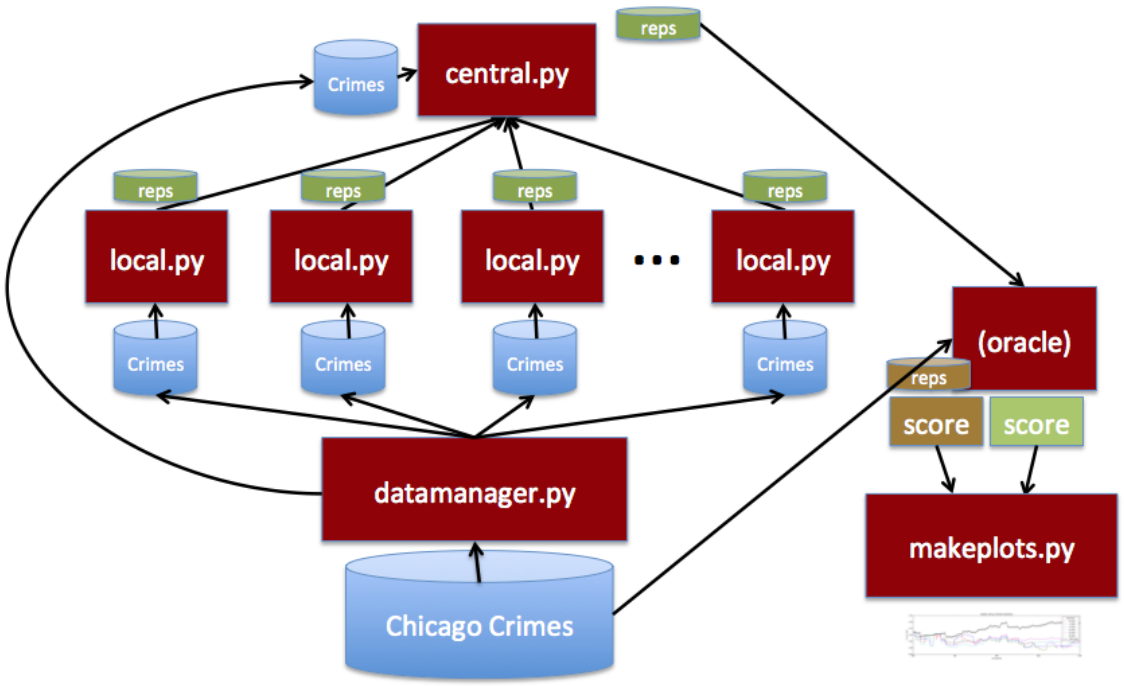
E Alexander & E Balkanski, May 2016, CS 262 final project

This system implements the greedy algorithm for data representative selection from Mirza & Soleiman's 2013 "Distributed submodular maximization: Identifying representative elements in massive data". It is extended to handle dynamic datasets. Given a dataset (in our case, a downloaded version of the Chicago crime dataset available at https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-present/ijzp-q8t2), the system divides the data, locally finds good representatives of each division, and sends these to a central system. The central system selects from among the received representatives to best represent its own division of the data. In our case, these divisions are all given new data points over time, and the frequency of communication from local processes is reflected in the performance of the central solution in comparison to a global greedy solution. See project report for details on motivation and performance.

Instructions for use: The file `demo` contains code to run a short demonstration of the system. To reproduce the experimental results in the project report, run the code in `experiment`. Simulations can be run with the function `datamanager.simulate`, individual processes started with `local.local` and `central.central`, and data initialization and updates performed with `datamanager.initialinsert` and `datamanager.update`.



There are 5 python files that interact according to the diagram above.

- `datamanager.py` contains functions:
  - `features`, used internally to preprocess data.
  - `initialinsert`, which inserts data from the large crimes set to the local files read by local and central processes, selecting which file uniformly at random.
  - `update`, which appends additional entries to these files at a set rate.
  - `simulate`, which calls `initialinsert`, starts local and central processes, then calls `update`. It also performs oracle scoring, recording the performance of the central solution (read from logs) on the entire dataset and running the the greedy algorithm run on the whole set by providing a local process with a file containing all data in the system.
- `local.py` contains the single function `local`, which monitors a data file for updates, recalculates representatives each time that file is updated, and sends its representatives (and process ID number) to the central process using sockets when enough changes have occurred since its last communication. On each update, the system time, number of local data points, and local score are recorded.
- `central.py` contains a single function `central`, which receives data over a socket from several local processes. When this happens, it selects from its received representatives those that best represent the contents of its data file and writes them to logs, along with system time, number of data points, and local score.

- `makeplots.py` can be called from the command line and reads central and oracle logs to produce plots of their raw scores and the ratio of the oracle score to each central process' score, per update.
- `greedy.py` contains several functions implementing the greedy algorithm of Mirza & Soleiman's 2013 paper "Distributed submodular maximization: Identifying representative elements in massive data". The two that are called by other components are:
  - `greedy`, which returns the representatives and their score for a dataset.
  - `score`, which only returns the score. This is used to score the central representatives on the complete dataset.

Several pre-existing components are imported (details can be found in function documentation):

- socket: sends messages between local and central machines
- json: encodes data to be sent over wire
- itertools.islice: allows access to parts of files
- random.randint: generates random integers for machine selection
- random.random: generates random floats between 0 and 1, for deletion
- time.sleep: allows pause before and between updates
- time.time: allows system time in logs
- os.rename: allows swap file for deleting entries
- csv.reader: separates data elements without splitting on commas within quoted strings, as are found in our data
- multiprocessing: starts local, central, and oracle simulations
- copy: for manipulating list values