Similar Item Search Service Design Document

My similar item search service is implemented in Python 3 and makes use of the Fastavro, Numpy and Flask libraries as well as the k-d tree data structure. The service primarily consists of the following parts:

- ## Search Index

  Keeps track of all items in the database and allows for efficient lookup and iteration over items in given categories. This is accomplished by maintaining in the index:

  - A dictionary mapping each category to an ordered list of ids of all items in the category. Each category list is coupled (by index) with a Numpy memory mapped 2D matrix containing the embedding vectors for corresponding items in each row. This allows efficiently iterating over item ids in each category and retrieving the embeddings associated with the item ids by reading them on demand from disk storage rather than having to keep all embeddings in RAM (the uncompressed embeddings vectors are quite large).
  - Another dictionary mapping item ids to the position of the corresponding item's record in the avro data file. This dictionary allows for efficiently reading in item records and information from disk storage (by seeking to appropriate positions in the data file) without keeping the entire database in RAM.

  Once the search index has been built, it's attributes are serializable, allowing it to be saved to disk and quickly loaded into memory for future runs.

- ## Search Component

  The search component of my system maintains an index object as described above and uses the index to construct a k-d tree for each category of items. The k-d trees enable the component to efficiently query the most similar items from a given category to a given item in the database. The component is represented as a dictionary mapping categories to corresponding k-d trees, along with the index which enables accessing item embedding data from disk. Once the component has been built, its attributes are serializable, allowing it to be saved to disk and quickly loaded into memory for future runs.

- ## Search Service

  The search service is implemented as an HTTP server application designed with Flask. It listens for HTTP requests on port 13000 and serves JSON formatted responses for similar item queries as well as queries for random items from the database. The app also serves a user interface that can be accessed from a web browser for visualizing query results.

Further improvements

Further improvements that could be made to the service include:

- Porting more code to Cython (C python extensions) in order to optimize speed, especially the code for building the search component's trees and making queries.
- Supporting efficient disk access of embedding vectors directly from avro input data file. Currently, separate Numpy memmap files are maintained by the service because the module has better built in support for efficient on disk operations in Python. This, of course, requires a potentially undesirable amount of extra available disk storage.
- A naïve formula is currently used to convert Euclidean distances provided by the k-d trees in the search component to similarity values. A more robust method for computing similarities could be added, and also alternate methods for deriving similarity could be supported (e.g. Hamming distance, Cosine similarity…).