Pixie Explanation

Pixie algorithm is powered by the visual discovery platform “Pinterest” which is a scalable real-time recommendation engine. This algorithm considers a bi-partite graph structure (Contains two types of nodes and have connections between the types and not with the same types). Pixie works with the concept of randomly walking through the graph and suggesting a material that the user could be interested in. Conventional recommendation systems like collaborative filtering heavily rely on matrix on matrix computations which is obviously expensive in terms of both time and space (Lot of unused spaces will be present). Whereas, Pixie fully online and in-memory allows real-time personalization. Briefly, Pixie algorithm starts from a node and keeps moving randomly in a graph (If it is weighted, bias will be based on weight and tie will be broken randomly) and recommendations will be given based on node visit counts.

Pixie considers weighted user interactions and accepts multi-query inputs and hence it represents a user’s wide range of interests. Pixie has been implemented with an early stop feature during a random walk, where the navigation will be stopped if the pin has been visited enough times only if the ranking is unlikely to change. This algorithm can handle more than 1000 queries per second as it is highly parallelized as it has very low latency.

In conclusion, Pixie is a combination of randomized algorithms, graph theory and customization techniques to provide relevant recommendations to users without training which is useful for real world data.

Algorithm

- Execute the random walk for the number of times the walk length is inputted.
    - As it is a user based approach, start the walk from the user and get the connected movies from the user. Among the connected movies, get the maximum rating and if there is only one movie with the maximum rating (say 4), move to that movie. If there are more than one movie with rating 4 (highest), randomly move to any of the highest ranked movies.
    - If the current standing node is a movie (we are having movie name and user id and hence if the current node is a string then it must be a movie) then add to dictionary and increment count, if the movie is already visited just increment the visit count of that movie.
    - Repeat the loop with the randomly found (highest rated) node as the current node until the walk length is completed.
- Once the walk is completed, sort the movies in the visited dictionary based on visit count in descending order. Get only the number of movies to be recommended from the array and convert to data frame and display.