Locality-sensitive Hashing and MinHash
The Problem

- Nearest-neighbor has prohibitive test complexity
  - Time
  - Space

- Instance of general problem:
  - given a large database $D$, and some distance function how do we make it efficient to:

  Find neighbors $N_D(q)$ for any given query example $q$
Key Ideas

- **Hash functions**
  - `<data> => small bit string`
  - Typically, designed such that distinct `<data>` rarely maps to same bit string
  - We will explore hash functions that map *similar* data to *similar* hash strings
    - “locality sensitive”

- **Similarity functions**
  - We’ll start with cosine similarity between vectors $x, y$
    \[
    \cos(x, y) = \frac{x \cdot y}{\|x\| \|y\|}
    \]
Semantic Relatedness

- Several measures have been proposed

- One that works well: “Milne-Witten”

$$SR_{MW} (x, y) \propto \text{fraction of Wikipedia in-links to either } x \text{ or } y \text{ that link to both}$$
Ad-hoc Reference Systems

Country

music

Rock

music

Hip

hop

music
Ad-hoc Reference Systems

Category: Grammy Award winners
From Wikipedia, the free encyclopedia
Ad-hoc Reference Systems
Ad-hoc Reference Systems

Tim McGraw
Ad-hoc Reference Systems

Tim McGraw

Jay-Z
How to use for nearest neighbor?

- Use $n$ hash functions $H_i$
- For query example $q$, retrieve lists $L_{ik} = \{x : H_i(x) = H_i(q)\}$
  - E.g., data set has 25,600 examples, say $H_i$ is 8 bits, each list then has avg. 100 elements.
  - Compute more expensive distance computation on union of those $n$ lists
Pros/Cons

- Works well when nearest neighbor is fairly near
  - Consider what lists look like given large #s of irrelevant attributes

- Works best for one NN, not kNN for largish k

- LSH has “uniform resolution” across whole input space
  - Usually not the best choice
  - Data-dependent methods can help