**SIMILAR ITEMS:**

**Overview:**

Given a collection of objects {O1, O2, … On}

Given 2 objects from the collection, Oi and Oj, how similar are they?

Compute similarity between them

Given Oi, find all objects that are similar to Oi

Lets assume we can compute similarity in time O(|Oi| + |Oj|)

Solve by computing similarity of Oi with all objects

O(Total size of all objects)

We will build a data structure to achieve this

Three Basic Questions:

Can we reduce time to compute?

How much memory do we need to store objects?

How can we “Group” similar objects?

Obvious way is to compute list of objects that are similar for each Oi

Takes O(n2) (Bad)

Objects could take the form of documents, images, audio files, and user profiles

We will assume documents for most examples

**Similarity Method:**

1. Extract features from objects
2. Represent each object as a point/vector in a high dimensional space
3. Define similarity between vectors
4. Apply dimensionality reduction (Compresses the data)
5. Locality Sensitive Hashing (LSH)

Dimensionality reductions means you are mapping points to a low-dimensional space

Want to preserve the similarity (approximately)

**Extracting Features:**

Feature is a characteristic of an objects

IE. Feature of a person could uniquely identify that person

For documents, features might be: words, author, Size, Language, Doc. Type/Title/Topic

Suppose our feature is a word

Common words make bad features for uniquely identifying documents

Start by removing all STOP words (common words)

Which words are STOP words depend on the document (no universal set)

Some words are derived from a ROOT/STEM words (IE. fish and fishing)

Location of words is also important

Steps you could follow:

1. Remove stop words
2. Remove punctuation
3. STEMMING: for each word, find its stemp and replace with STEM

Can view document as set of words, or multiset of words

Set of words: {Hash, Table}

Mutliset of words: {hash, hash, table, table}

There is no single answer for what works best when extracting features (it depends on the document)

**Location of Words (Word Order):**

Suppose we have two docs, Da and Db, and two sets of words, Sa and Sb

Suppose also that the two sets are close to each other

Can we say Da and Db are very similar?

We also want to capture the meaning of the words

Use a shingle or k-shingle where k is an integer

**Shingle:**

Suppose our document contains the words “james bond”

If you look at all 4-shingles, they are “jame”, “ames”, “mes\_”, “es\_b”, etc…

Sometimes shingles are better at capturing the meaning

Often used to detect plagiarism

If you take only 1-shingles, you would reduce all docs to sets of 26 letters (not very good for distinguishing)

Number of shingles that is best depends on the document

Sometimes, can look at word shingles instead of character shingles (4 words in a row, instead of 4 characters)

**Terms (AKA Features):**

Suppose we have made our choices for extracting terms from the document, and we have extracted them

Suppose we have document set D = {D1, D2, … Dn}

The key = {T1, T2, … Tm} is the set of all terms in the document collection

**Vector Representation:**

Now we view each document as a vector

TFij = frequency of ith term in the jth document

It can be binary = 1 if Ti is in Dj, and 0 if it is not

Given a document d, Vd = <TF1d, TF2d, … TFmd>

This is the vector representation of the document

**Similarity:**

If the distance between the vectors is small, then the documents are similar

Euclidean-distance:

V = <a1, a2, … an>

L2 of a vector V is the distance of V to the origin

L2(V) = sqrt(a12 + a22 + … an2)

One way of measuring similarity is to measure the angle between the Euclidean-distances, instead of the distance between the two points

For two vectors U and V:

Cos(angle) = (U dotProduct V) / (L2(U)\*L2(V))

This is called Cosine-similarity

Jaccard-Similary of two vectors U and V:

= (U dotProduct V) / (L2(U)2 + L22(V) – U dotProduct V)

Jaccard-Similarity of two sets, S1 and S2:

= |S1 and S2| / |S1 or S2|