Text similarity

- In essence, the goal is to compute how ‘close’ two pieces of text are in (1) meaning or (2) surface closeness.
- The first is referred to as **semantic similarity** and the latter is referred to as **lexical similarity**. Although the methods for *lexical similarity* are often used to achieve *semantic similarity* (to a certain extent), achieving true *semantic similarity* is often much more involved.

Lexical or Word Level Similarity

- how similar two pieces of text are at the surface level. For example, how similar are the phrases *“the cat ate the mouse”* with *“the mouse ate the cat food”* by just looking at the words? On the surface, if you consider only word level similarity, these two phrases (with determiners disregarded) appear very similar as 3 of the 4 unique words are an exact overlap.

- This notion of similarity is often referred to as lexical similarity. It typically does not take into account the actual meaning behind words or the entire phrase in context.

Granularity

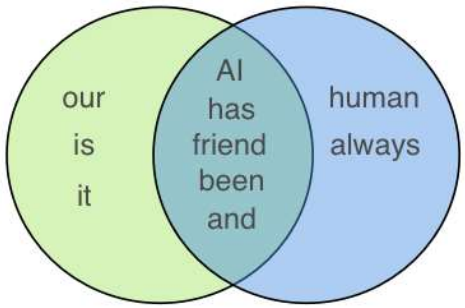
- Another point to note is that lexical similarity can be computed at various granularity. You can compute lexical similarity at the *character level*, *word level* (as shown earlier) or at a *phrase level* (or lexical chain level) where you break a piece of text into a group of related words prior to computing similarity. Character level similarity is also known as string similarity/matching and is commonly used to determine how close two strings are. For example how close are the names *'Kavita Ganesan'* and *'Kavita A Ganesan'* ? Pretty close! You can use the common metrics outlined below for string similarity or you can use edit distance type of measures to quantify how dissimilar two strings are. In essence, you are trying to compute the minimum number of operations required to transform one string into the other.

Jaccard Similarity

- Jaccard similarity or intersection over union is defined as **size of intersection divided by size of union of two sets.** Let's take example of two sentences:
- **Sentence 1:** AI is our friend and it has been friendly
  Sentence 2: AI and humans have always been friendly
- In order to calculate similarity using Jaccard similarity, we will first perform **lemmatization** to reduce words to the same root word. In our case, "friend" and "friendly" will both become "friend", "has" and "have" will both become "has". Drawing a Venn diagram of the two sentences we get:

Jaccard Similarity

- For the above two sentences, we get Jaccard similarity of **5/(5+3+2) = 0.5** which is size of intersection of the set divided by total size of set.



- The more you normalize, pre-process and filter your text (e.g. stem, remove noise, remove stop words), the better the outcome of your text similarity measure using simple measures such as Jaccard.

Jaccard Similarity

- One thing to note here is that since we use sets, “friend” appeared twice in Sentence 1 but it did not affect our calculations — this will change with Cosine Similarity.

Cosine Similarity

- Cosine similarity calculates similarity by measuring **the cosine of angle between two vectors**. This is calculated as:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Cosine Similarity

- **With cosine similarity, we need to convert sentences into vectors**. One way to do that is to use **bag of words with either TF** (term frequency) **or TF-IDF** (term frequency- inverse document frequency). The choice of TF or TF-IDF depends on application and is immaterial to how cosine similarity is actually performed — which just needs vectors.

Cosine Similarity

- Let's calculate cosine similarity for these two sentences:
- **Sentence 1:** Bring mangoes
  Sentence 2: Mangoes are sweet
- **Step 1**, we will calculate Term Frequency using Bag of Words:
- Our vocabulary is [mangoes bring sweet] (considering "are" are stop-word)

sent 1: [1, 1, 0]
sent 2: [1, 0, 1]

Cosine Similarity

- Cosine similarity between the two sentences is:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

(1*1 + 1*0 + 0*1)/sqrt((1^2 + 1^2))* sqrt((1^2 + 1^2)) = 1/2 = 0.5

Differences between Jaccard Similarity and Cosine Similarity:

- Jaccard similarity takes only **unique set of words** for each sentence / document while cosine similarity takes **total length of the vectors**. (these vectors could be made from bag of words term frequency or tf-idf)
- This means that if you repeat the word “friend” in Sentence 1 several times, cosine similarity **changes** but Jaccard similarity does not. For ex, if the word “friend” is repeated in the first sentence 50 times, cosine similarity drops to 0.4 but Jaccard similarity remains at 0.5.
- Jaccard similarity is good for cases where duplication does not matter, cosine similarity is good for cases where duplication matters while analyzing text similarity. For two product descriptions, it will be better to use Jaccard similarity as repetition of a word does not reduce their similarity.

Where is lexical similarity used?

- **Clustering** – if you want to group similar texts together how can you tell if two groups of text are even similar?
- **Redundancy removal** – if two pieces of texts are so similar, why do you need both? You can always eliminate the redundant one. Think of duplicate product listings, or the same person in your database, with slight variation in the name or even html pages that are near duplicates.
- **Information Retrieval** –given a query, retrieve all relevant documents that contain the given query