3 Jaccard Similarity and k-Grams

We will study how to define the distance between sets, specifically with the Jaccard distance. To illustrate and motivate this study, we will focus on using Jaccard distance to measure the distance between documents. This uses the common "bag of words" model, which is simplistic, but is sufficient for many applications.

We start with some big questions. This lecture will only begin to answer them.

- Given two homework assignments (reports) how can a computer detect if one is likely to have been plagiarized from the other without *understanding* the content?
- In trying to index webpages, how does Google say a webpage is similar to a keyword. How does Google avoid listing duplicates or mirrors?
- How does a computer quickly understand emails, for either detecting spam or placing effective advertisers? (If an ad worked on one email, how can we determine which others are similar?)

The key to answering these questions will be convert the data (homeworks, webpages, emails) into an object in an *abstract space* that we know how to measure distance, and how to do it efficiently. The most obvious abstract space is Euclidean space $\mathbb{R}^d$. An object $v \in \mathbb{R}^d$ can be thought of as a d-dimensional point (or vector) $v = (v_1, v_2, \ldots, v_d)$. The notation of a list of objects, separated by commas, inside parenthesis (and) represents an *ordered set*; that is $(a, b) \neq (b, a)$. The Euclidean distance between two points $v, u \in \mathbb{R}^d$ is measured

$$d_{\mathsf{E}}(u, v) = \|u - v\| = \sqrt{\sum_{i=1}^{d} (v_i - u_i)^2}.$$

This is the common *straight line* distance. We will return to this later, as it will not be immediately useful for distances between documents. Instead we will use a different abstract distance between (unordered) *sets*.

3.1 Sets and Distances

A *set* is a (unordered) collection of objects $\{a, b, c\}$. We use the notation as elements separated by commas inside curly brackets { and }. They are unordered so $\{a, b\} = \{b, a\}$.

Although we are interested in a "distance," we will actually focus on a dual notion of a *similarity*. A distance $d(A, B)$ has the properties:

- it is small if objects A and B are close,
- it is large if they are far,
- it is (usually) 0 if they are the same, and
- it has value in $[0, \infty]$.

On the other hand, a similarity $s(A, B)$ has the properties:

- it is large if the objects A and B are close,
- it is small if they are far,
- it is (usually) 1 if they are the same, and
- it is in the range $[0, 1]$.

Often we can convert between the two as $d(A, B) = 1 - s(A, B)$, however sometimes it is better to use $d(A, B) = \sqrt{s(A, A) + s(B, B) - 2s(A, B)}$. Both restrict the distance to be a bounded (non infinite) domain, that can be converted with a $\tan$ map if one desires.

3.1.1 Jaccard Similarity

Consider two sets $A = \{0, 1, 2, 5, 6\}$ and $B = \{0, 2, 3, 5, 7, 9\}$. How similar are A and B?

The *Jaccard similarity* is defined

$$\begin{aligned}\mathsf{JS}(A, B) &= \frac{|A \cap B|}{|A \cup B|} \\ &= \frac{|\{0, 2, 5\}|}{|\{0, 1, 2, 3, 5, 6, 7, 9\}|} = \frac{3}{8} = 0.375\end{aligned}$$

More notation, given a set A, the *cardinality* of A denoted $|A|$ counts how many elements are in A. The *intersection* between two sets A and B is denoted $A \cap B$ and reveals all items which are in *both* sets. The *union* between two sets A and B is denoted $A \cup B$ and reveals all items which are in *either* set.

Confirm that JS satisfies the properties of a similarity.

Generalized set similarities. To fully generalize set similarities (at least those that are amenable to large-scale techniques) we introduce a third set operation. The *symmetric difference* between two sets A and B is denoted $A \triangle B = (A \cup B) \setminus (A \cap B)$. Note that $\setminus$ is called *set minus* and $A \setminus B$ is all of the elements in A, except those also in B. Thus the symmetric difference of A and B describes all elements in A or B, but not in both.

We now consider the follow class of similarities. We use $\overline{A \cup B} = [n] \setminus (A \cup B)$, where $[n]$ is a superset that all sets A and B we consider a subsets from.

$$S_{x,y,z,z'}(A, B) = \frac{x|A \cap B| + y|\overline{A \cup B}| + z|A \triangle B|}{x|A \cap B| + y|\overline{A \cup B}| + z'|A \triangle B|}.$$

Now we can define several concrete instances.

- *Jaccard Similarity* is defined

$$\mathsf{JS}(A, B) = S_{1,0,0,1}(A, B) = \frac{|A \cap B|}{|A \cap B| + |A \triangle B|} = \frac{|A \cap B|}{|A \cup B|}.$$

- *Hamming Similarity* is defined

$$\mathsf{Ham}(A, B) = S_{1,1,0,1}(A, B) = \frac{|A \cap B| + |\overline{A \cup B}|}{|A \cap B| + |\overline{A \cup B}| + |A \triangle B|} = 1 - \frac{|A \triangle B|}{|[n]|}.$$

- *Andberg Similarity* is defined

$$\mathsf{Andb}(A, B) = S_{1,0,0,2}(A, B) = \frac{|A \cap B|}{|A \cap B| + 2|A \triangle B|} = \frac{|A \cap B|}{|A \cup B| + |A \triangle B|}.$$

- *Rogers-Tanimoto Similarity* is defined

$$\mathsf{RT}(A, B) = S_{1,1,0,2}(A, B) = \frac{|A \cap B| + |\overline{A \cup B}|}{|A \cap B| + |\overline{A \cup B}| + 2|A \triangle B|} = \frac{|[n]| - |A \triangle B|}{|[n]| + |A \triangle B|}.$$

- *Sørensen-Dice Similarity* is defined

$$\mathsf{S\text{-}Dice}(A, B) = S_{2,0,0,1}(A, B) = \frac{2|A \cap B|}{2|A \cap B| + 1|A \triangle B|} = \frac{2|A \cap B|}{|A| + |B|}.$$

For JS, Ham, Andb, and RT, the distance $D(A, B) = 1 - S(A, B)$ is a *metric*, and these four are amenable to LSH. We will discuss these topics later. See Chierichetti and Kumar (*LSH-Preserving Functions and their Applications*, SODA 2012) for more details.

3.2 Documents to Sets

How do we apply this set machinery to documents?

Bag of words vs. k-Grams The first option is the *bag of words* model, where each document is treated as an unordered set of words.

A more general approach is to *shingle* the document (or create k-grams). This takes consecutive words and group them as a single object. A *k-gram* is a consecutive set of k words. So the set of all 1-grams is exactly the bag of words model. An alternative name to *k-gram* is a *k-shingle*; these mean the same thing.

D_1 : `I am Sam.`

D_2 : `Sam I am.`

D_3 : `I do not like green eggs and ham.`

D_4 : `I do not like them, Sam I am.`

The $(k = 1)$-grams of $D_1 \cup D_2 \cup D_3 \cup D_4$ are: {`[I], [am], [Sam], [do], [not], [like], [green], [eggs], [and], [ham], [them]`}.

The $(k = 2)$-grams of $D_1 \cup D_2 \cup D_3 \cup D_4$ [1] are: {`[I am], [am Sam], [Sam Sam], [Sam I], [am I], [I do], [do not], [not like], [like green], [green eggs], [eggs and], [and ham], [like them], [them Sam]`}.

The set of k-grams of a document with n words is at most $n - k$. The takes space $O(kn)$ to store them all. If k is small, this is not a high overhead. Furthermore, the space goes down as items are repeated.

Character level. We can also create k-grams at the character level. The $(k = 3)$-character grams of $D_1 \cup D_2$ are: {`[iam], [ams], [msa], [sam], [ami], [mia]`}.

The $(k = 4)$-character grams of $D_1 \cup D_2$ are: {`[iams], [amsa], [msam], [sams], [sami], [amia], [miam]`}.

Modeling choices.

- **White space?** Should we include spaces, and returns? Sometimes. `plane has touch down` versus `threw a touchdown`.
- **Capitalization?** `Sam` versus sam. Can help distinguish proper nouns.
- **Punctuation?** May be indication of education level, or dialects. For instance English is punctuated differently in US and India. Punctuation is used differently in new articles (very proper style), blogs (more informal), and twitter (what is punctuation?).
- **Concatenation?** Should we concatenate consecutive documents as above, or not let k-grams span separate documents?
- **Characters vs. Words?** Long enough grams with characters can simulate words, but will have more *false positives*. Can pick up other dialect patterns. But is less interpretable.

- **How large should k be?** General rule: probability of (almost all) k-grams is low, so a collision is meaningful.
  For word-k-grams: emails $k = 2$ or 3 (small documents), research articles $k = 3$ or 4 (large documents), news articles, blog posts (in between).
  In English there are 27 characters (26 letters + 1 whitespace). With $k = 5$ there are $27^5 \approx 14$ millions

[1] Note here we treat the union of documents as concatenation. e.g. $D_1 \cup D_2 =$ `I am Sam. Sam I am.` This is not always typical for much larger documents.

possible k-grams. (Maybe in practice closer to 20^5 since some letters (e.g. `z`, `q`, `x` are rarely used.)

- **Count replicas?** Typically *bag of words* counts replicas, but *k-gramming* does not.
- **Stop words?** Words like {`a, you, for, the, to, and, that, it, is, ...`} are very common, and called *stop words*. Sometimes omit these (typically in bag of words). In shingling can be effective to say use $k = 3$ where the first word must be a stop word: `the pizza oven`.

There are many variations of these methods. Natural Language Processing (NLP) studies these variations, but also focuses on finding much richer representations of bodies of text. Identifying all nouns and verbs, and disambiguating words with multiple meanings `went to the retreat` versus `the troops had to retreat`.

3.3 Jaccard with k-Grams

So how do we put this together. Consider the $(k = 2)$-grams for each D_1, D_2, D_3, and D_4:

D_1 : `[I am], [am Sam]`

D_2 : `[Sam I], [I am]`

D_3 : `[I do], [do not], [not like], [like green], [green eggs], [eggs and], [and ham]`

D_4 : `[I do], [do not], [not like], [like them], [them Sam], [Sam I], [I am]`

Now the Jaccard similarity is as follows:

$$\begin{aligned}
\mathsf{JS}(D_1, D_2) &= 1/3 \approx 0.333 \\
\mathsf{JS}(D_1, D_3) &= 0 = 0.0 \\
\mathsf{JS}(D_1, D_4) &= 1/8 = 0.125 \\
\mathsf{JS}(D_2, D_3) &= 0 = 0.0 \\
\mathsf{JS}(D_3, D_4) &= 2/7 \approx 0.286 \\
\mathsf{JS}(D_3, D_4) &= 3/11 \approx 0.273
\end{aligned}$$

Next time we will see how to use this special abstract structure of sets to compute this distance (approximately) very efficiently and at extremely large scale.

3.4 Continuous Bag of Words

Sometimes it is useful to obtain statistics for individual words from text corpuses. This will allow us to compare words in ways we will discuss later in the course. The simplest way is called *continuous bag of words (CBOW)*. For any instance of a word in a corpus (for instance "like"), it collects in a set (with multipilicity) all other words used within a distance r of that word. For large corpuses $r = 5$ has been suggested. Then a word's representation is the combination of all of these sets.

Distance between words are then calculated commonly in two ways. The first is Jaccard distance between the sets.

The second it to represent the set of all possible words as elements of a vector: each coordinate corresponds with a distinct word. Then for the representation of an instance of a word in the CBOW model, we build a vector v_{word}, initially all 0s. Each other word in a neighborhood set has its coordinate in v_{word} set to 1 (or c if it occurs c times in the neighborhood set). Then the vector representation of that word from the corpus, is the average of all vectors for each instance of that word. Finally, we can use the *cosine distance* (discussed in a later lecture, or just Euclidean distance) to determine the distance between words.

Example Consider the text corpus:
`I am Sam Sam I am I do not like green eggs and ham I do not like them Sam I am`
Then using the example word "like" and the parameter $r = 2$, the representation of the first and second instances are:

$$v^1_{\text{like}} = (0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0)$$
$$v^2_{\text{like}} = (0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1)$$

where the each coordinate is associated with the following words

$$(\texttt{I}, \texttt{am}, \texttt{Sam}, \texttt{do}, \texttt{not}, \texttt{like}, \texttt{green}, \texttt{eggs}, \texttt{and}, \texttt{ham}, \texttt{them}).$$

In reality, the above vectors are much longer, with the many other words used in English.

The average of the two vectors is then:

$$(v^1_{\text{like}} + v^2_{\text{like}})/2 = v_{\text{like}} = (0, 0, 0.5, 1, 1, 0, 0.5, 0.5, 0, 0, 0.5)$$

Note that the other words `do` and `not` occur commonly with `like`.