Assignment 5: Emoji space

In this assignment you will use a technique know as latent semantic analysis to learn the Emoji similarities space.

NOTE

Assignment duration is 2 weeks. Hand it in by Sunday, April 23, 11:59pm.

1 Load/acquire data

1.1 Tweets

To work well, latent semantic analysis requires a lot of data. I recommend using a dataset that is rich in Emojis and contains at least 100,000 tweets. If your dataset from the previous assignment falls short of these specifications, consider collecting more tweets using your `my_streamer`.

1.2 Emojis

Load the new Emoji list. In my experience the Emoji in row 2283 results in an error. In case you make the same experience remove it. E.g., `emoji_ids = emoji_ids[-2283,]`.

2 Create term-document matrix

First eliminate non-occuring Emojis and tweets without Emojis, which will help greatly in speeding up the process. Then fill the term-document matrix with raw frequencies. Details below.

2.1 Eliminate non-occuring Emojis

Iterate over all Emojis and identify whther they occur in any of the tweets. To do this collapse the tweets into a single text and use `stri_count_fixed()` to count the frequency of each Emoji. Conveniently, the functions pattern argument takes a vector, so the calculation can be done in a single command. Note: only use the utf-8 representation. Then in a next step create a new object containing only those Emojis that occur in the text.

2.2 Eliminate tweets without Emojis

Iterate over all tweets and identify if they contain Emojis. To do this create a search pattern that contains all Emojis using `|` (pipe). Specifically, collapse all Emojis into a single string using the pipe character as the collapse argument. Then create new object containing only the tweets that contain Emojis. Useful for later: store the number of Emojis (total not unique) per tweets in an additional object.

2.3 Create term-frequency matrix

Create a new matrix `td_mat` with as many rows as the number of remaining emojis and as many columns as the number of remaining tweets. Iterate over the tweets and use `stri_count_fixed()` to fill the columns of `td_mat`. Again, use the vectorized version as it is much faster.

3 Process term-frequency matrix

Transform to *tf-idf*. That is, transform the raw frequency in each cell f_{ij} to $tfidf_{ij} = tf_{ij} * idf_{ij}$ where $tf_{ij} = \frac{f_{ij}}{n_j} = \frac{f_{ij}}{\sum_i f_{ij}}$ and $idf_i = log\frac{N_d}{\sum_j I(f_{ij}>0)}$. The indicator function $I(\cdot)$ is 1 if argument is true and 0 if argument is false.

3.1 Determine number of Emojis per tweets

If you have stored the numbers of Emojis in the tweets in (2.2) extract the counts for the remaining tweets. Alternatively calculate the column sums of the `td_mat` using `colSums()` - almost as easy and fast as the first option. Finally divide the number 1 by the resulting vector, which will give you the inverse of the number of number of Emojis (terms) per tweets (document) $\frac{1}{n_j}$. You will need this in a second.

3.2 Determine number of tweets per Emoji

Calculate the row sums of the binarized `td_mat` that ccarries a 1 if $f_{ij} > 0$, and 0 if $f_{ij} = 0$. The easiest way to do this is to use a logical operation on the entire matrix, i.e., `td_mat > 0` and then `rowSums()`. Finally, divide the number of documents N_d by the resulting row sums and take the logarithm using `log` to compute idf_i.

3.3 Calculate *tf-idf*

Calculate tf-idf as $tfidf_{ij} = f_{ij} * \frac{1}{n_j} * idf_i$. To do this it is convenient to calculate first the outer product (using `%o%`) of the two vectors $\frac{1}{n_j}$ and idf_i. The outer product creates from two vectors of length m and n a $m \times n$ matrix. Thus, by using `n_doc_byemoji %o% n_emoji_bydoc` we can create a matrix, called e.g., `tmp_mat` , that matches the dimensions of `td_mat`. Finally, to calculate the product of `td_mat` and `tmp_mat` to obtain `tfidf_mat`, i.e., `tfidf_mat = td_mat * tfidf_mat`.

4 Singular value decomposition

Apply the singular value decomposition and represent Emojis in s-dimensional space.

4.1 Compute svd

Download and load package `corpcor`. Execute `emoji_svd = fast.svd(tfidf_mat)`. This may take a few minutes. Save object for later purposes.

4.2 Represent emojis in fewer dimensions

Determine Emoji represention based on fewer dimensions. Emoji representation is calculated as $T_s = T\Sigma_s$, where Σ_s is Σ with all but the s most import singular values replaced by 0. In R, we simply calculate `emoji_repr = emoji_svd$u[,1:s] * emoji_svd$d[1:s]`. Choose s to be some value around 300. The result is our learned Emoji space in s dimensions.

5 Plot result

To plot the emojis we need to project the sD space into the 2D plane. A crude way to do this is to simply ignore the less important s-2 dimensions by just using the first two. A better way is to calculate the similarity between the Emojis using the *cosine* similarity and then reduce the inverse similarity space using, e.g., tSNE (t-distributed stochastic neighbor embedding).

5.1 Calculate cosine similarities

Install and load the `lsa` package. Apply `cosine()` to our emoji representation `emoji_repr`. Because `cosine()` calculates similarities between columns, use the transpose `t(emoji_repr)`.

5.2 Translate cosine into distance

First rescale cosines to be between 0 and 1 using `(1 + x) / 2`. Then translate cosines into distances by taking the inverse. E.g., calculate `1 / (x + b)` where be is a constant used to limit the range of distances. That is, some cosines will be of very small value. If we take the inverse of very small values the result will be extreme, which will work against us in producing a well organized 2D representations. Thus, `b` should take some value such that the difference between the largest and smallest distance is not too large. I've used `b=.5`.

5.3 Apply `tsne`

Install and load `tsne` package. Apply `tsne` to matrix of distances. The result is a matrix with two columns containing the x and y positions.

5.4 Plot and post result

Plot the emojis on a 2D canvas. For instance, start a canvas using `plot.new()`, then plot a canvas using `plot.window()` where `xlim` and `ylim` is equal to the ranges of the x and y coordinates. Then plot the Emojis using either my function from Assignment 3 or the Emoji font that others have been using.