Lab01: Tweets' similarity

Abdelkrime Aries

We want to implement a small program to detect similar tweets. Our target population is Algerians (Algerian tweets) which use Arabic, Berber, English, French and Arabizi. One use case is to detect spammers and robots since they tend to repeat their tweets with some minor adjustments.

1 Tweets' description

The tweets were downloaded from `https://www.kaggle.com/datasets/didamarouane/algerian-tweets` (4134 tweets). We deleted line breaks in tweets so they will be more manageable. some tweets were deleted because they contain just links and/or tags, others for being inappropriate or too redundant/short. Some were deleted because they were in Spanish, Chinese, etc. 2455 tweets remain.

1.1 Language

Let's consider the combination between each spoken language and writing system as a separate language. In this case, Algerians use these different languages in this dataset (alphabetical order): Algerian dialect (Arabizi: in Latin script), Algerian dialect (Arabic script), Berber (Latin script), English (Arabic script), English (Latin script), French (Arabic script) and Standard Arabic. Sometimes, these are all mixed into one tweet.

1.2 Links, emails and tags

All links are shortened as `https://t.co/` followed by an ID combined from basic Latin letters (uppercase and lowercase), numbers and underscores. Example:

`https://t.co/SSjjlxm8n5`

Emails are Latin letters (uppercase and lowercase), numbers, dots and underscores followed by @ followed by Latin letters (uppercase and lowercase), numbers, dots and underscores; then a dot then a string of letters.

Tags can be user tags; starting with @ followed by a string of basic Latin letters (uppercase and lowercase), numbers and underscores. They can be hash tags; starting with # followed by a string of anything but the space.

`@Honeygain_App #ReferralCode BAKLI434`

2 Program description

Here, different functions are described; either those implemented or not. You have to understand how they are implemented to respond on the different questions.

2.1 Word similarity

This function calculates the similarity between two words based on their letters. This similarity is based on Levenstein distance; the less the words are distant, the more they are similar. Given two words w_1 and w_2, it is calculated as follows:

$$sim(w_1, w_2) = \frac{\max(|w_1|, |w_2|) - levenstein(w_1, w_2)}{\max(|w_1|, |w_2|)}$$

This must be implemented.

2.2 Tweet normalization

First, spaces are doubled (for each space, we add another). Then, the all letters are transformed into lowercase (for Latin script). Also, Tashkiil and Tatweel are deleted (For Arabic script). When normalizing the text, you can add as much consecutive spaces as you want since **split** function will ignore them. Use **re.sub** to complete this task (normally no more than 25 operations is needed)

The first normalization step is to replace:

- Mails with a token **[MAIL]**
- User tags with a token **[USER]**
- Hash tags with a token **[HASH]**
- Links with a token **[LINK]**

The second step is to normalize words (without tokenization). Let's start with Latin based scripts (French, English, Berber):

- Replace accentuated **e**'s and **a**'s with **e** and **a** respectively.
- The ending **s** must be deleted (plural; even if it is not a plural).
- French suffixes (ir, er, ement, ien, iens, euse, euses, eux) must be deleted.
- English suffixes (ly, al) must be deleted. If the word ends with **ally**, we delete just **ly**.
- Berber suffixes (ya, en) must be deleted. For example: iggarzen → iggarz, arnuyas → arnu
- English contractions must be transformed into their origin. Such as: it's → it is, don't → do not
- French contractions must be transformed into their origin. Such as: qu'ont → que ont, s'abstenir → se abstenir, p'tit → petit.

DZ Arabizi has some Arabic rules as well as rules specific to Algerian population. The difference, it is written in Latin script. These are the rules which must be implemented:

- Negation must be deleted (ma...ch). For example, mal9itch → l9it
- Suffix **k, km** variations must be deleted. ywaf9ek → ywaf9, ya3tik → ya3ti, 3ndk → 3nd, 3ndkm → 3nd
- Suffixes **a, i, o, ou** must be deleted when the radical is two letters or more. This must be after the last rule in case of suffixes **ak, ik, ok, ouk** For example, yetfarjou → yetfarj, fhamto → fhamt, mousiba → mousib, wladi → wlad
- Suffixes **h, ha** must be deleted when the radical is two letters or more. For example, khatih → khati, katiha → khati

Standard Arabic and Dz Arabic are written using Arabic script.

- Algerian negation must be split. For example, مَا نلبسوش → مَانلبسوش, مَا نلبسو → منلبسوش, مَا نلبسو
- **Al** qualifier variants (ال، للّ، فل، وَال، ول، بل، بَال) must be deleted if the rest is 2 or more letters. For example, البحر → بحر, للّبحر → بحر, فلبحر → بحر, وَالبحر → بحر, ولبحر → بحر,
- Standard Arabic plural suffixes (ين، ون، ات) as well as a borrowed suffix from French (ال). For example,

سونتر → سونترَال, إِتِحَاد → إِتِحَادَات, يجر → يجرون, رَايح → رَايحين

- Arabic object pronouns (ني، ك، ه، هَا، نَا، كمَا، كم ، كن، همَا، هم، هن) which are suffixes, as well as وَا must be deleted if the rest if 2 letters or more. For example, بَالهَا → بَال, بَالكم → بَال, يرموَا → يرم,
- Some other Arabic and Algerian suffixes must be deleted (ا، و، ي، ة) if the rest if 2 letters or more. For example, رَايحَا → رَايح, قَالو → قَال, سكنة → سكن

This must be implemented.

2.3 Get similar word

Given a word w and a list of words $[w'_1, \cdots, w'_n]$, the most similar word w'_s is found using the function you completed earlier.

This is already implemented.

2.4 Tweets' similarity

Given two tweets $T = [w_1, \cdots, w_n]$ and $T' = [w'_1, \cdots, w'_m]$, their similarity is calculated as follows:

$$sim(T, T') = \frac{\sum_{w \in T} \max_{w' \in T'} sim(w, w') + \sum_{w' \in T'} \max_{w \in T} sim(w', w)}{n + m}$$

This is already implemented.

3 Questions

Answer these questions at the start of your code, as comments.

1. What are the problem(s) with normalization in our case (Algerian tweets)?
2. Why word similarity is based on edit distance and not vectorization such as TF?
3. Why tweets similarity is proposed as such? (not another formula such as the sum of similarity of the first tweet's words with the second's divided by max length)
4. Technical question: why blanks are being duplicated before using regular expressions in our case?

4 Evaluation

- Duration: 1h (You have to return your work at the end of session)
- Grade
  - **word similarity grade** (2pts) = no detail (it is so simple).
  - **tweet normalization grade** (12pts) = mails, tags and links (3pts = 4*0.75pt) + words' (8pts = 16*0.5pts) + number of substitution regular expressions (1pt = [25, 27, 29, 31] * 0.25pt).
  - **questions grade** (4pts) = 1pt for each question.
  - **In time grade** (2pts): after the deadline, each late minute is -0.25. So, 8 minutes then you will get 0.