Text Normalization

Section 1: Descriptions of the System

1.1: The design

The tweets are one kind of special text corpus where we may have many expressions which are not standard expression. The design of the system is meant to handle the following characteristics of tweets:

- The special sequences: urls, the topic tags, the user tags, concatenation of several words, etc.
- The misspelling(deliberated or not) of words

For the two characteristics, what I do is to build a system of 2 main steps:

1. Preprocessing: remove the urls, detect and mark the topic tags and user tag, split the concatenated sequences.
2. Misspelling correction: For the words that are not correct (not in the dictionary), correct it using both context information and formal similarity information.

1.2: The building process

The main idea is to firstly build the overall pipeline, then fill the detailed implementations:

1. Build the "preprocess.py" file.
2. Build the pipeline where "system.py".
    1. It does nothing to the preprocessed sentences: it just take the input as output, the purpose is to check whether the bash file works well.
    2. Add the function to detect the positions in the sentences where the words are not in the dictionary.
    3. Add the function to get the context vector of the target positions.
    4. Get the candidates replacements using both vector similarity and Levenshtein distance.
    5. Add the 'STATE' parameter to specify the way to select the candidates (manual or auto).

1.3: The components

The system is implemented in python.

1. For the preprocessing part, I used one existing library ekphrasis (https://github.com/cbaziotis/ekphrasis), the code in "preprocess.py" is a modified version of https://github.com/cbaziotis/ekphrasis/blob/master/ekphrasis/examples/example.py.

2. For the misspelling correction part, I used the library context2vec (https://github.com/orenmel/context2vec), the code in "system.py" is developed by myself, but I referred to the https://github.com/orenmel/context2vec/blob/master/context2vec/train/train_context2vec.py.

Section 2: Situations Handling

2.1: Situations well handled

For the simple misspelling, such as "survivor —> survivar", the system can easily detect and correct it. The system considered both context information and formal similarity information, so the recovery of simple misspelled words in a relatively clear context is good.

2.2: Situations not properly handled

1. In the system, I didn't consider the impact of the keyboard: 'q' is more likely to be misspelled to 's' than 'p' because 'q' is closer to 's' in the 'qwerty' keyboard. One possible solution is to attach different penalty in the computation of Levenshtein distance: larger penalty for the replacement between two 'far-away' letters.

2. In the system, I didn't consider the frequent situation where people use the repeated vowels. For example, "liiiiiiiiike" means "like", but in our system, this transformation will be counted 9 units of Levenshtein distance. One possible solution is try to "merge" the characters (in this case, 'i') which repeat too many times.