TWEET TOKENIZER

The tokenizer is built using regular expressions at large and string manipulation.

1)The tokenizer first finds matches for the different formats of dates and replaces them with the required format like CF:D:yyyy-mm-dd. In case of only '4-digit Year' reference, the program checks if the previous words are 'in' or 'from' and if found so, then replaces this year with required format.
Eg: 'originated in 2015' will be tokenized as ['originated', 'in', 'CF:D:2015:??:??']
Similar things happen in case of reference of a month.
Eg: July becomes 'CF:D:????:07:??'

2) The tokenizer then finds matches for time formats and replaces them with canonical format. For this regular expression is used. A list of abbreviations of all available time zones is maintained in the program
Eg: '05:30 pm IST' becomes 'CF:T:1730:IST'
'IST 12 pm' becomes 'CF:T:1200:IST'

3) Tokenizer finds matches for usernames and hashtags with the help of regular expression
Eg: '@Daniel-Radcliffe' remains a single token and is not tokenized further
'#Potter_Heads' remains a single token

4) Clitics are handled by considering the tense of the words coming after the clitic

Eg: 'I'd tweeted' becomes 'I had tweeted' but 'I'd go' becomes 'I should go'
'It's raining' becomes 'It is raining' but 'It's rained' becomes 'It has rained'
'rock'nroll' becomes 'rock and roll'
'can't' becomes 'can not', 'ain't' becomes 'are not', 'shan't' becomes 'shall not'

In case of 'd clitic, if the next words after 'd is a past participle (i.e. either ending in -ed or from the list of exceptional past participles), then 'd will be tokenized as 'had'. Else, if word is "I'd" or "We'd", then 'd becomes "should" else it becomes 'would'.

In case of n't clitic, the exceptions are hardcoded in the program in the form of dictionary like {'ain't' :'are not'},etc.

For 's clitic, the program takes into consideration the 'ing' form of the verb appearing after 's.

5) Hyphenated strings are handled using isTitle() methodology.
Eg: US-Japan becomes ['US', '-', 'Japan'] because 'Japan' contains a captalised first letter,whereas, US-based remains ['US-based'] because 'based' does not contain a capitalised first letter

6) Strings like iGuess are split into 'i' and 'Guess' using CamelCase concept. If in a word, a Capitalised letter appears in between, the tokenizer splits the word on the capitalised alpabet.

Eg: 'TheWindowSill' becomes ['The', 'Window', 'Sill'] because the word gets split on W and S as they are capitalised.

7) Regular expression is used to extract URL from the tweet and does not undergo any further splitting.

8) Abbreviations are matched using a regular expression and the space between the characters is removed to make it a single word.
Eg: ‘B. E.’ becomes ‘B.E.’
‘U. K.’ becomes ‘U.K.’

After all the above replacements, the line is split on spaces.
The output of this split is then split on punctuation marks and the resulting list of tokens is then written to file ‘output.txt’.

Interesting dataset used:

1) Fetched famous and most-liked tweets from wikipedia and prepared a dataset.