Due Monday, March 18, 2019

East Asian Language Tokenization

For this project you have a choice between implementing Thai tokenization or Japanese tokenization.

Thai Tokenization

You will take a finite state approach to Thai tokenization. This is a simplified FSA that will work on the examples provided, but may not work on every Thai sentence.

./thai_tokenizer.py <input_file> <output_file>

Characters in Thai can be categorized as follows:

	Category	Members	Unicode
V1	Preposed vowel	เแโใไ	0E40, 0E41, 0E42, 0E43, 0E44
C1	Initial consonant	กขฃคฅฆงจ ฉชซฌญฎฏฐ ฑฒฑณดตถ ทธนบปผฝพ ฟภมยรฤลฦ วศษสหฬอฮ	0E01 - 0E2E
C2	Clustered consonant	รลวนม	0E19, 0E21, 0E23, 0E25, 0E27
V2	Super/subscript vowel	◌ิ ◌ี ◌ึ ◌ื ◌ุ ◌ู ◌ั ◌็	0E31, 0E34, 0E35, 0E36, 0E37, 0E38, 0E39, 0E47
T	Superscript tone mark	◌่ ◌้ ◌๊ ◌๋	0E48, 0E49, 0E4A, 0E4B
V3	Postposed vowel	าอยว	0E22, 0E27, 0E2D, 0E32
C3	Final consonant	งนมดบกยว	0E01, 0E07, 0E14, 0E19, 0E1A, 0E21, 0E22, 0E27

Every word has a consonant from the list of initial consonants, but characters from the other categories may or may not appear in a word. They appear in the following order:

V1? C1 C2? V2? T? V3? C3?

The states and transitions are as follows:

	v1	c1	c2	v2	t	v3	c3	#
q0	q1	q2	-	-	-	-	-	-
q1	-	q2	-	-	-	-	-	-
q2	q7	q8	q3	q4	q5	q6	q9	-
q3	-	-	-	q4	q5	q6	q9	-
q4	q7	q8	-	-	q5	q6	q9	-
q5	q7	q8	-	-	-	q6	q9	-
q6	q7	q8	-	-	-	-	q9	-
q7	-	-	-	-	-	-	-	q1
q8	-	-	-	-	-	-	-	q2
q9	-	-	-	-	-	-	-	q0

If the state becomes 7 or 8, the FSA has encountered a new starting vowel or consonant, insert a space **before** the character that initiated the transition. From state 7 or 8, transition to state 1 or 2 (respectively), before looking at the next character.

If the state becomes 9, the FSA has encountered a final consonant, insert a space **after** the character that initiated the transition. From state 9, transition to state 0.

Note that there are overlaps in the categories! (The characters in C2, C3, and V3 overlap with C1). To deal with this, accept the characters in the following order:

	Character Class Order
q0	V1, C1
q1	C1
q2	C2, V2, T, V3, C3, V1, C1
q3	V2, T, V3, C3
q4	T, V3, C3, V1, C1
q5	V3, C3, V1, C1
q6	C3, V1, C1
q7	
q8	
q9	

The sample_out.txt file contains the correctly tokenized output of the in.txt file. Please check your program output against this file.

Japanese Tokenization

For the Japanese tokenization project, you will use the MaxMatch algorithm to read in a file with lines of Japanese text without spaces, find the word boundaries, and output a file with the same lines of text with spaces added between words.

./japanese_tokenizer.py <input_file> <output_file>

The MaxMatch algorithm uses a dictionary (japanese_wordlist.txt). Starting from the beginning of the sentence, match the longest consecutive string of characters that exists in the dictionary. Once you have found that longest string, insert a space and continue. If no matches exist in the dictionary, consider the character a one character word, insert a space and continue.

The sample_out.txt file contains the correctly tokenized output of the in.txt file. Please check your program output against this file.

gold_standard.txt contains the ideal tokenization. The MaxMatch algorithm makes mistakes, so don't expect your tokenization output to match this. When you're done implementing MaxMatch, compare the output of your file to the gold_standard. Make a file (by hand or programmatically) named evaluation.txt that contains the following:

# of sentences tokenized correctly: <# of lines of output that match gold_standard>

# of sentences tokenized incorrectly: <# of lines of output that don't match gold_standard>

accuracy: <# correct / total # of sentences>