CS4248 Natural Language Processing
Tutorial 1: Words

In this tutorial, we'll practice three technical topics from our Week 02 lecture: Regular Expressions, Edit Distance and Byte Pair Encoding. We'll also have a look at morphology; a concept that reoccurred across Week 01 and 02.

Please come to your tutorial session, with your attempts to these questions. It's fine to come without a working solution but to get the most out of tutorial, you should attempt them so that you have practice before solutions can be shared. Extension exercises marked with a "**" or "***" are slightly more advanced and you (and your tutorial leader) may not have time to go over them. You're welcomed to discuss these on the forum among yourselves (Let's keep an active forum!)

We may ask some students to share their solutions so that we know where we might struggle when solving these problems. Don't worry about the correctness of your solution and it is totally fine if you don't have an answer. Our target is to contribute a tutorial together that everyone can learn something from. Thank you!

1. **What module are you taking? (a.k.a. Regular Expressions)**

   Let's practice some regexes, by turning them loose on our own corpus, the NUS School of Computing website. Let's visit the page containing SoC's schedule of module mounting here:

   `https://www.comp.nus.edu.sg/cug/soc-sched/`

   Then write regexes for each of the following tasks:

   1. Match all of the modules from the Computer Science (CS) department.
   2. Match all senior level (4000 level) modules from CS department. CS4248 should be there!
   3. Match all numeric parts of module codes (e.g. "2010") offered by IS department.
   4. Match all CS modules that are NOT CS4248. (So sad!)

   ***To think about*: Which of these regular expressions are finite state machines (FSM) that are of the form that were introduced in lecture. Why? Why not?

   Explanation: Answers:

   1. CS[0-9]+[A-Z]*: "CS" matches the characters CS literally (case-sensitive); [0-9] matches a number; + matches the previous token (in this case, a number) for at least one time, as many times as possible, giving back as needed (greedy); [A-Z] Matches an upper case letter; * matches the previous token between zero and unlimited times, as many times as possible, giving back as needed (greedy)
   2. CS4[0-9]+[A-Z]*: CS4 matches any modules starting with CS4, [0-9]+ matches one to unlimited numbers, [A-Z]* matches zero to unlimited upper case letters.
   3. (?<=IS)[0-9]+: (?<=IS) is the positive lookbehind for "IS": we are looking for things following "IS". [0-9]+ matches one to unlimited numbers.
   4. (?!CS4248)CS[0-9]+[A-Z]*: (?!CS4248) is the negative lookahead for "CS4248", we are looking for things NOT followed by CS4248.

2. **A Pop Star's Nemesis (a.k.a. Minimum Edit Distance)**

   In the latter half of Week 02, we also covered Minimum Edit Distance (MED) as a means of knowing how many operations can transform one string to another.

   Spelling correction is a real problem, as you may know. The name "Britney Spears" (a famous pop music star, in case you've been hiding under a rock) has many variants according to Google.

   Dance Dance Revolution: Guess how many variants Google thinks it has for this rock star. Then access (not "acress", haha) the below page and find out for yourself.

   `http://archive.google.com/jobs/britney.html`

   Then please calculate the edit distance between following pairs of strings. Show your work and give your backtrace as well. Use a substitution cost of "2" as in lecture, and disallow transposition operations.

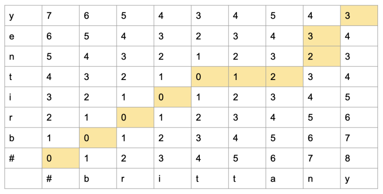
   1. "brittany" and her real first name.
   2. "britey" and her real first name.
   3. "rbitney" and her real first name.

   ***A little more.* Do your answers (MED score and backtrace) change if your substitution cost is "1"? How about "1.8"? How about if you allow transposition at a cost of "1"?

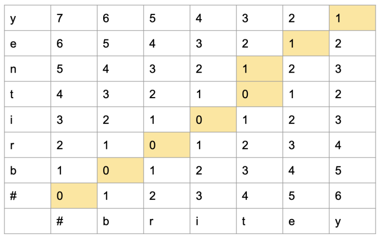
   Explanation: Answer:

   1. britney ⇒ brittany: 3, where 3 = insert(t) + insert(a) + del(e).
   2. britney ⇒ britey: 1, where 1 = del(n).
   3. britney ⇒ rbitney: 2, where 2 = del(b) + insert(b).

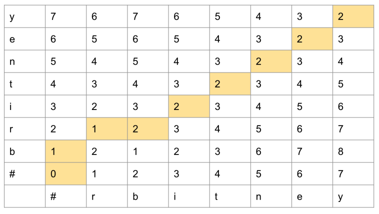
   Note that the target word is on the x-axis.
   For the backtracers table, click this link for a bigger one.



y	7	6	5	4	3	4	5	4	**3**
e	6	5	4	3	2	3	4	**3**	4
n	5	4	3	2	1	2	3	**2**	3
t	4	3	2	1	**0**	**1**	**2**	3	4
i	3	2	1	**0**	1	2	3	4	5
r	2	1	**0**	1	2	3	4	5	6
b	1	**0**	1	2	3	4	5	6	7
#	**0**	1	2	3	4	5	6	7	8
	#	b	r	i	t	t	a	n	y



y	7	6	5	4	3	2	**1**
e	6	5	4	3	2	**1**	2
n	5	4	3	2	**1**	2	3
t	4	3	2	1	**0**	1	2
i	3	2	1	**0**	1	2	3
r	2	1	**0**	1	2	3	4
b	1	**0**	1	2	3	4	5
#	**0**	1	2	3	4	5	6
	#	b	r	i	t	e	y



y	7	6	7	6	5	4	3	**2**
e	6	5	6	5	4	3	**2**	3
n	5	4	5	4	3	**2**	3	4
t	4	3	4	3	**2**	3	4	5
i	3	2	3	**2**	3	4	5	6
r	2	**1**	**2**	3	4	5	6	7
b	**1**	2	1	2	3	6	7	8
#	**0**	1	2	3	4	5	6	7
	#	r	b	i	t	n	e	y

Figure 1: Backtracers for Q2.

3. **Weighted edit distance**:

   Let's say the weight of substitution error equals to the distance of the letters on the standard terribly-inefficient QWERTY keyboard, e.g. $sub(Q, A) = 1$ and $sub(Q, E) = 2$. Letters that share any part of edges count as adjacent (on my keyboard, $sub(S, E) = 1$, but $sub(E, F) = 2$. Recalculate the pairs above.

   ***You're really Dvorak-ing it now.* How about on a Dvorak keyboard?

   Explanation: Weighted edit distance answers:
   There will be no changes compared the above one. This is because there is no useful optimization by using the Sub() weights on keyboard. Click this link for details.

4. **I need Morph- ine (a.k.a. Morphology)**

   Ing-glish Morph-o-logy. We learned a bit about morphemes in Week 01. Please indicate which one of the operations are considered *stemming* (chopping off affixes) and which are considered *lemmatization* (reducing to a dictionary form).

   1. only ⇒ onli
   2. the ⇒ the
   3. best ⇒ good
   4. examples ⇒ exampl
   5. are ⇒ be
   6. accepted ⇒ accept

   ****Do try this at home.* We mentioned that stemming systems differ, since they are rule-based systems. You can try the tokenizers for NTLK[a] and SpaCy[b]. Can you find words that differ in their base stemming? (Try this and you can legitimately write those keywords on cyour résumé, HAHA).

   [a] `https://www.nltk.org/`
   [b] `https://spacy.io/`

   Explanation: **Answers:**

   Stemming: 1, 2, 4, 6. We can see in 1 and 4, we even reduce the words to more "stemmed" forms.

   Lemma: 2, 3, 5, 6. We can see for these examples, words have big changes, NOT merely chopping off the affixes.

   "the" is a special case, since it is already the most basic form, hence the output is still "the" for both stemming and lemma.

   "accepted" is also a special case, it falls into both categories.

5. **Don't 😟 be 😄:**

We might consider emoji as a foreign language that is used by aliens 👽 who prefer visual communication. Sometimes, one emoji can be seen as the composition of a *stem* and an *affix*, e.g., 🤓 $\Rightarrow$ 😀 + 👓.

Please find the stems and affixes from the candidate lists.

Candidate stem list: 😀, 🧑, 😙, 🐻, 🏳️;

Candidate affix list: 🚀, 🌈, 💻, 👓, 🕶️, 🎉, 🏀, ❄️

1. 😎 $\Rightarrow$? + ?
2. 🧑‍💻 $\Rightarrow$? + ?
3. ⛹️ $\Rightarrow$? + ?
4. 🧑‍🚀 $\Rightarrow$? + ?
5. 🥳 $\Rightarrow$? + ?
6. 🐻‍❄️ $\Rightarrow$? + ?
7. 🏳️‍🌈 $\Rightarrow$? + ?

***To* 🤔 *about...*: What do you think right way to describe the categorization of the affixes in the above patterns? Prefix or suffix? Bound or free? (No right answers here; really just for you to 🤔. Hmmmm...).

Explanation: Emoji Morphology answers:

1. 😎 = 😀 + 🕶️
2. 🧑‍💻 = 🧑 + 💻
3. ⛹️ = 🧑 + 🏀
4. 🧑‍🚀 = 🧑 + 🚀
5. 🥳 = 😙 + 🎉
6. 🐻‍❄️ = 🐻 + ❄️
7. 🏳️‍🌈 = 🏳️ + 🌈

6. **Byte Me (a.k.a. Byte Pair Encoding)**

Let's practice some BPE-ing. Consider a corpus consisting of the words:

{*target pasta star star pasta star*}

1. **Learning:** Execute BPE token learning on the corpus, where $k = \infty$. Show all merges.

2. **Segmentation:** Execute the BPE token segmentation for following new words, according to your learned vocabulary in Step 1.

   New words: *tapas*, *stata*.

****Once byte-n, twice shy.* BPE is originally a compression algorithm, where the compressed message is sent along with the merges (dictionary of tokens). How much space is saved by using a k of 3 and 6 (You may have to make some assumptions to be able to calculate this)? Why do you think you get the result that you get?

Explanation: For merges:

t a ⇒ ta
s ta ⇒ sta
sta r ⇒ star
star _⇒ star_
p a ⇒ pa
pa sta ⇒ pasta
pasta _⇒ pasta_
ta r ⇒ tar
tar g ⇒ targ
targ e ⇒ targe
targe t ⇒ target
target _⇒ target_

For segmentation:

tapas ta, pa, s, _
stata sta, ta, _
Thanks Chris for the notebook