

CS221 Fall 2018 Homework 3

SUNet ID: 05794739

Name: Luis Perez

Collaborators:

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

Problem 1

- (a) To show that this greedy algorithm is suboptimal, simply consider the following set-up. First, we work with a 1-gram model, and consider the input:

“theseccount”

The greedy algorithm will compare the following on the first iteration:

$u(\text{“t”})$
 $u(\text{“th”})$
 $u(\text{“the”})$
 $u(\text{“thes”})$
 $u(\text{“these”})$
 $u(\text{“thesecc”})$
 $u(\text{“thesecco”})$
 $u(\text{“theseccou”})$
 $u(\text{“theseccount”})$

From the above, it’s reasonable to have our 1-gram model such that $u(\text{“the”}) < u(\text{“these”})$, and both of these will obviously have lower cost than the other non-English words. Therefore, on the first iteration, our greedy algorithm will select the split:

“the seccount”

On the second iteration, the algorithm will consider:

$u(\text{“s”})$
 $u(\text{“se”})$
 $u(\text{“sec”})$
 $u(\text{“secco”})$
 $u(\text{“seccou”})$
 $u(\text{“seccount”})$

Note that none of these are English words, and therefore we define them to have extremely high cost. For the sake of simplicity, we'll have $u(\text{"secount"})$ have the lowest cost amongst the above, but still have an extremely high cost since it's not an English word. As such, the final output of our algorithm will be:

"the secount"

With cost $u(\text{"the"}) + u(\text{"secount"})$. However, note that the optimal split point would actually be:

"these count"

with cost $u(\text{"these"}) + u(\text{"count"})$. We note that:

$$u(\text{"these"}) + u(\text{"count"}) < u(\text{"the"}) + u(\text{"secount"})$$

mainly because of an extremely high cost associated with $u(\text{"secount"})$.

(b) In "submissions.py".

Problem 2

(a) We simply consider the sequence of vowel-free words:

mnysvl.

Let us define first possibleFills:

$$\begin{aligned}\text{possibleFills}(\text{mny}) &= \{\text{money}, \text{many}\} \\ \text{possibleFills}(\text{s}) &= \{\text{is}, \text{sea}\} \\ \text{possibleFills}(\text{vl}) &= \{\text{evil}, \text{veil}\}\end{aligned}$$

We now consider how our greedy algorithm will proceed above.

- The algorithm will first pick "many", given that $\text{bigramCost}(-\text{BEGIN}-, \text{money}) > \text{bigramCost}(-\text{BEGIN}-, \text{many})$, since in English, more sentences begin with "many".
- Next, it will select "sea", since $\text{bigramCost}(\text{many}, \text{sea}) < \text{bigramCost}(\text{many}, \text{is})$ given that "many is" is grammatically incorrect.
- Next, it will select "veil" since $\text{bigramCost}(\text{sea}, \text{veil}) < \text{bigramCost}(\text{sea}, \text{evil})$, which is a somewhat reasonable assumption.

In this case, the greedy algorithm will therefore end with the output "Many sea veil" with a relatively high bigram cost, due to high cost of (many, sea), and (sea, veil).

However, note that the optimal reconstruction would have been "Money is evil" given the significantly lower costs of (money, is) and (is, evil).

(b) In "submissions.py".

Problem 3

- (a) We present the formal definition of the search problem below. We can define the states as consisting of the tuples (w, text) where w is the fully-vowelized word which was previously selected to be part of our output (or a special token signifying the start, if nothing was previously selected), and text is the sequence of characters that remain to be split into words and for vowels to be inserted. More formally, let our input query be $q = b_0b_1 \cdots b_n$, a query string of n constants where b_0 is a special token specifying the start of a string. Then our states are:

$$\text{States}(b_0b_1 \cdots b_n) = \{(b_0, b_1b_2 \cdots b_n)\} \cup \bigcup_{i=1}^n \bigcup_{j=i}^n \{(w, b_jb_{j+1} \cdots b_n) \mid w \in \text{possibleFills}(b_ib_{i+1} \cdots b_{j-1}b_j)\}$$

Given the above definition of our states, it's easy to see what our start state s_{start} is:

$$s_{\text{start}} = (b_0, b_1b_2 \cdots b_n)$$

recalling that b_0 is our special token specifying the start of a string.

We continue by defining our $\text{Actions}(s)$ for some state $s = (w, b_ib_{i+1} \cdots b_n)$. Intuitively, the actions simply consist of two things – picking a “split point” $i \leq j \leq n$ in the remaining characters such that we will have two substrings $b_i \cdots b_j$ and $b_{j+1} \cdots b_n$ (which is the empty string when $j+1 > n$). We can formalize the above by defining our action space as:

$$\text{Actions}((w, b_ib_{i+1} \cdots b_n)) = \bigcup_{j=i}^n \{p_j \mid p_j \in \text{possibleFills}(b_ib_{i+1} \cdots b_j)\}$$

We continue expanding on the above by explicitly defining our successor function, which can be formally defined as:

$$\text{Succ}((w, b_ib_{i+1} \cdots b_n), p_j) = (p_j, b_{j+1}b_{j+2} \cdots b_n)$$

Next, we need to formally define the $\text{Cost}(s, a)$ of taking a particular action in state s . This can be directly defined for our problem as:

$$\text{Cost}((w, b_ib_{i+1} \cdots b_n), p_j) = \text{bigramCost}(w, p_j)$$

since it is just the cost of selecting p_j as the next word.

Finally, we now simply need to consider the end test. This too can easily be defined as follows:

$$\text{IsEnd}((w, b_ib_{i+1} \cdots b_n)) = i > n$$

since we reach an end state whenever we have no further characters to split (ie, we have an empty string).

(b) In “submissions.py”.

(c)

- (d)
- Yes, UCS is a special case of A^* . We can see this by noting that UCS is simply A^* where we have the same states, actions, starting state, end states, and costs. However, for UCS, we must have that the heuristic function h is given by $h(s) = 0$. Therefore UCS is a less generic algorithm.
 - Yes, BFS is a special case of UCS. BFS is simply UCS where we have the same states, actions, starting state, and end states. However, for BFS, we have that $\text{Cost}(s, a) = 1$ always (ie, our cost function is one). Therefore, BFS is a special case of UCS. Therefore BFS is a less generic algorithm.