Homework 3: Text Reconstruction

Brandon McKinzie

Problem 1: Word Segmentation

In word segmentation, you are given as input a string of alphabetical characters ([a-z]) without whitespace, and your goal is to insert spaces into this string such that the result is the most fluent according to the language model.

(a) *Consider the following greedy algorithm: Begin at the front of the string. Find the ending position for the next word that minimizes the language model cost. Repeat, beginning at the end of this chosen segment.*

Show that this greedy search is suboptimal. In particular, provide an example input string on which the greedy approach would fail to find the lowest-cost segmentation of the input.

In creating this example, you are free to design the n-gram cost function (both the choice of n and the cost of any n-gram sequences) but costs must be positive and lower cost should indicate better fluency. Note that the cost function doesn't need to be explicitly defined. You can just point out the relative cost of different word sequences that are relevant to the example you provide. And your example should be based on a realistic English word sequence — don't simply use abstract symbols with designated costs.

Let's consider a unigram model (n=1). Given a starting index i into the original character sequence s, this model will scan from i to $i + L - 1$, where L is the length of the original character sequence. It will choose to end the word at

$$j = \arg\min_j -\ln p(s_{i:j}) \tag{1}$$

This model makes its segmentation decisions one-by-one and then moves on (i.e. it is unable to revise previous decisions). Consequently, it will perform poorly when trying to segment compound words or any words that contain common/frequently-used word substrings.

For example consider the input $s := cathedral$. Nearly all english corpora will assign a much higher unigram probability to the substring *cat* compared to the word *cathedral*. Therefore, the greedy model will begin by segmenting s into $[cat, hedral]$. It may continue on to segment this into $[cat, he, dral]$. Even more unfortunate is that it ends up having to decide between very low probability segmentations amongst the substring *dral*, which will substantially add to the total cost. In summary, the model was unable to realize, while making its first decision, that although

$$Pr(cat) > Pr(cathedral)$$

it is also true that

$$Pr(cathedral) >> Pr(cat)Pr(he)Pr(dral)$$

(regardless of how it decides to segment *dral*).

Problem 2: Vowel Insertion

Now you are given a sequence of English words with their vowels missing (A, E, I, O, and U; never Y). Your task is to place vowels back into these words in a way that maximizes sentence fluency (i.e., that minimizes sentence cost). For this task, you will use a bigram cost function.

You are also given a mapping possibleFills that maps any vowel-free word to a set of possible reconstructions (complete words). For example, possibleFills('fg') returns set(['fugue', 'fog']).

(a) *Consider the following greedy-algorithm: from left to right, repeatedly pick the immediate-best vowel insertion for current vowel-free word given the insertion that was chosen for the previous vowel-free word. This algorithm does not take into account future insertions beyond the current word.*

Show, as in question 1-a, that this greedy algorithm is suboptimal, by providing a realistic counter-example using English text. Make any assumptions you'd like about possibleFills and the bigram cost function, but bigram costs must remain positive.

Similar to problem 1(a), this model will suffer from its inability to revise its past decisions given updated information (future context). For example, given input query "`lk vr thr`", the model might first be presented with the following candidate bigrams:

- (-BEGIN-, look)
- (-BEGIN-, like)
- (-BEGIN-, leak)

and so on. Its decision will be determined by the minimum bigram cost and it wont take into consideration possible future decisions. This may result in, for example, the suboptimal "`like over three`" instead of the likely best option (based on my intuition) of "`look over there`".

Problem 3: Putting it all Together

We'll now see that it's possible to solve both of these tasks at once. This time, you are given a whitespace- and vowel-free string of alphabetical characters. Your goal is to insert spaces and vowels into this string such that the result is as fluent as possible. As in the previous task, costs are based on a bigram cost function.

(a) *Consider a search problem for finding the optimal space and vowel insertions. Formalize the problem as a search problem; what are the states, actions, costs, initial state, and end test? Try to find a minimal representation of the states.*

- **States**. The states are essentially a combination of the states for the two subproblems. As with the segmentation subproblem, we'll need to keep track of where we are in the original string. As with the insertion subproblem, we'll need to store the previous (filled-in) word. Our states will then take the form (w_{prev}, i).
- **Actions**. At any given step, we need to decide where the next word segmentation boundary will go, and which vowels to insert into this newly-segmented word.
- **Initial state**. (-BEGIN-, 0).
- **End test**. That our current index has reached the end of the string (the index of the special token -END-, if we were to use one).

(c) *Let's find a way to speed up joint space and vowel insertion with A*. Recall that one way to find the heuristic function h(s) for A* is to define a relaxed search problem P_{rel} where $Cost_{rel}(s, a) \leq Cost(s, a)$ and letting $h(s) = FutureCost_{rel}(s)$.*

Given a bigram model b (a function that takes any (w',w) and returns a number), define a unigram model u_b (a function that takes any w and returns a number) based on b.

Use this function u_b to help define P_{rel}.

One example of a u_b is $u_b(w) = b(w, w)$. However this will not lead to a consistent heuristic because $Cost_{rel}(s, a)$ is not guaranteed to be less than or equal to $Cost(s, a)$ with this scheme.

Explicitly define the states, actions, cost, start state, and end state of the relaxed problem and explain why h(s) is consistent.

Note: Don't confuse the u_b defined here with the unigram cost function u used in Problem 1.

Hint: If u_b only accepts a single w, do we need to keep track of the previous word in our state?

In order to ensure $u_b(w) < b(w', w)$, we can simply define

$$u_b(w) = \min_{w'} b(w', w) \tag{2}$$

which by definition no longer depends on storing the previous word. Therefore, our state can be just i, the current index into the query word.

(d) *We defined many different search techniques in class, so let's see how they relate to one another.*

Is UCS a special case of A? Explain why or why not.*

Yes. We saw this in lecture. UCS is the special case of A* where $h(s) = 0$.

Is BFS a special case of UCS? Explain why or why not.

Yes. BFS is the special case of UCS where all costs are the same constant $c \geq 0$.