LING/COMP 445, LING 645
Problem Set 3

Name: Jan Tiegges , **McGill ID**: 261180937
Collaborators:

Due before 4:35 PM on Wednesday, October 18, 2023

Please enter your name and McGill ID above. There are several types of questions below.

- For questions involving answers in English or mathematics or a combination of the two, put your answers to the question in an answer box like in the example below. You can find more information about LaTeX here `https://www.latex-project.org/`.
- For programming questions, please put your answers into a file called `ps3-lastname-firstname.clj`. Be careful to follow the instructions exactly and be sure that all of your function definitions use the precise names, number of inputs and input types, and output types as requested in each question.

  For the code portion of the assignment, **it is crucial to submit a standalone file that runs**. Before you submit `ps3-lastname-firstname.clj`, make sure that your code executes correctly without any errors when run at the command line by typing `clojure ps3-lastname-firtname.clj` at a terminal prompt. We cannot grade any code that does not run correctly as a standalone file, and if the preceding command produces an error, the code portion of the assignment will receive a 0.

  To do the computational problems, we recommend that you install Clojure on your local machine and write and debug the answers to each problem in a local copy of `ps3-lastname-firstname.clj`. You can find information about installing and using Clojure here `https://clojure.org/`.

  A template Clojure file will be provided with the helper functions described below.

Once you have entered your answers, please compile your copy of this LaTeX into a PDF and submit

(i) the compiled PDF renamed to `ps3-lastname-firstname.pdf`
(ii) the raw LaTeX file renamed to `ps3-lastname-firstname.tex` and
(iii) your `ps3-lastname-firstname.clj`

to the Problem Set 3 folder under 'Assignments' on MyCourses.

Example Problem: This is an example question using some fake math like this $L = \sum_0^\infty \mathcal{G}\delta_x$.

Example Answer: Put your answer in the box provided, like this:

Example answer is $L = \sum_0^\infty \mathcal{G}\delta_x$.

MAIN PROBLEM SET

Problem 1: In these exercises, we are going to be processing some natural linguistic data, the first paragraph of Moby Dick. We will first write some procedures that help us to manipulate this corpus. We will then start analyzing this data using some probabilistic models.

We'll start by defining the variable `moby-word-tokens`, the tokens in our corpus as a Clojure list. This variable is defined for you already in the provided template Clojure file.

```
(def moby-word-tokens '(CALL me Ishmael . Some years ago never mind how long
  precisely having little or no money in my purse , and nothing particular to
  interest me on shore , I thought I would sail about a little and see the
  watery part of the world . It is a way I have of driving off the spleen ,
  and regulating the circulation . Whenever I find myself growing grim about
  the mouth whenever it is a damp , drizzly November in my soul whenever I
  find myself involuntarily pausing before coffin warehouses , and bringing up
  the rear of every funeral I meet and especially whenever my hypos get such
  an upper hand of me , that it requires a strong moral principle to prevent
  me from deliberately stepping into the street , and methodically knocking
  people's hats off then , I account it high time to get to sea as soon as I
  can . This is my substitute for pistol and ball . With a philosophical
  flourish Cato throws himself upon his sword I quietly take to the ship .
  There is nothing surprising in this . If they but knew it , almost all men
  in their degree , some time or other , cherish very nearly the same feelings
  toward the ocean with me .))
```

In the template Clojure file, we have also defined the function `member-of-list?`, with the following code.

```
(defn member-of-list? [w l]
  (if (empty? l)
    false
    (if (= w (first l))
      true
      (member-of-list? w (rest l)))))
```

It has two arguments, `w` and `l`, and returns `true` if `w` is a member of the list `l`, and `false` otherwise. For example, `(member-of-list? 'a '(a ship is))`, will return `true`, and `(member-of-list? 'the '(a ship is))` will return `false`.

The template Clojure file contains a skeleton for the function `get-vocabulary`, which you must implement. This function takes two arguments, `word-tokens` and `vocab`, and it should return a list of all unique words occurring in `word-tokens`. For example, if `word-tokens` is `'(the ship is the ship)`, then get-vocabulary should return `'(the ship is)`. Implement this function by filling in the missing parts of this provided code.

When you call `(get-vocabulary moby-word-tokens '())`, you will get back a list of all of the unique words occurring in `moby-word-tokens`. Give this the name `moby-vocab`.

Note: there are a lot of choices that go into processing text when doing work with a corpus. That is not the point of this problem set. To make things easier:

- Don't worry about case. So, treat `With` and `with` as different words.
- Don't worry about punctuation. Treat `.` as a word just like any other. Also, note that commas are treated as whitespace in Clojure so they will be ignored by your code. We left them in the list `moby-word-tokens` just for readability.

Answer 1: Please put your answer in `ps3-lastname-firstname.clj`.

Problem 2: Define a function get-count-of-word. This function should take three arguments, w, word-tokens, and count, where w is a word, word-tokens is a list of words, and count is a number.

When you call (get-count-of-word w word-tokens 0), the function should return the number of occurrences of the word w in the list word-tokens. For example

- (get-count-of-word 'the (list 'the 'the 'whale) 0) should return 2.
- (get-count-of-word 'the (list 'the 'whale) 0) should return 1.

Write get-count-of-word as a recursive function. You can use the count argument to accumulate the words counted so far.

Answer 2: Please put the answer in ps3-lastname-firstname.clj.

Problem 3: In the template Clojure file, we have provided a function get-word-counts, which takes two arguments, vocab and word-tokens, where vocab is assumed to be a list of the unique words that occur in the list word-tokens.

```
(defn get-word-counts [vocab word-tokens]
  (let [count-word
        (fn [w] (get-count-of-word w word-tokens 0))]
    (map count-word vocab)))
```

This function returns the number of times each word in vocab occurs in word-tokens. For example, suppose vocab is '(whale the is), and word-tokens is '(the is whale is). Then the function will return the list (1 1 2), corresponding to the number of times 'whale, 'the, and 'is occur in word-tokens, respectively.

Use this function and the other variables we have defined, to define a variable named moby-word-frequencies. This variable should contain the number of times each word in moby-vocab occurs in moby-word-tokens.

Answer 3: Please put your answer in ps3-lastname-firstname.clj.

Problem 4: In class we defined the functions normalize, flip, and sample-categorical. These functions will be very useful for us, and are included below as well as in the Clojure template file.

```
(defn flip [p]
  (if (< (rand 1) p)
    true
    false))

(defn normalize [params]
  (let [sum (apply + params)]
    (map (fn [x] (/ x sum)) params)))

(defn sample-categorical [outcomes params]
  (if (flip (first params))
    (first outcomes)
    (sample-categorical (rest outcomes) (normalize (rest params)))))
```

We have also provided a function that returns a particular probability distribution, the *uniform distribution*. The uniform distribution is the distribution which assigns equal probability to every possible outcome. The function create-uniform-distribution takes a single argument, outcomes, which is a list of length n. The function returns a list containing the number $1/n$ repeated n times. For example, if outcomes is `'(the a every)`, then this function will return `'(1/3 1/3 1/3)`. This list can be interpreted as a probability distribution over the outcomes, which assigns equal probability to each of them.

```
(defn create-uniform-distribution [outcomes]
  (let [num-outcomes (count outcomes)]
    (map
      (fn [x] (/ 1 num-outcomes))
      outcomes)))
```

Using functions `create-uniform-distribution` and `sample-categorical`, write a function `sample-uniform-BOW-sentence` that takes two arguments: a number `n` and a list `vocab`, and returns a sentence of length `n`. Each word in the sentence should be generated independently from the uniform distribution over vocab. For example, if `n` is `4` and `vocab` is `'(the a every)`, a possible return value for this function is `'(a the the a)`.

Note that this is a bag of words model, as defined in class. That is, we assume every element of the list is generated independently. We will call this the uniform bag of words model.

Answer 4: Please put your answer in `ps3-lastname-firstname.clj`.

Problem 5: Define a function `compute-uniform-BOW-prob`, which takes two arguments, `vocab` and `sentence`. `vocab` is the list of all words in the vocabulary, and `sentence` is a list of observed words. The function should return the probability of the sentence according to the uniform bag of words model.

For example, if `vocab` is `'(the a every)`, and sentence is `'(every every)`, then the function should return the number $\frac{1}{9}$.

Answer 5: Please put your answer in `ps3-lastname-firstname.clj`.

Problem 6: Using `sample-uniform-BOW-sentence` and `moby-vocab`, sample a 3-word sentence from the vocabulary of our Moby Dick corpus. This will be a sample from the uniform bag of words model for this vocabulary. Repeat this process a handful of times. For each of these 3-word sentences, use `compute-uniform-BOW-prob` to compute the probability of the sentence according to the uniform bag of words model. Are the different sentences you sampled assigned different probabilities under this model? Explain why this is (or isn't) to be expected.

Answer 6: Please put your answer in the box below.

The sentence probability is always identical $\frac{1}{2744000}$. This is due to the fact, that the probability among the different words in the vocabulary is uniformly distributed, meaning that no matter which words are being sampled, the probability will be unaffected. Would we sample words from outside the vocabulary, we could achieve a probability of 0 as alternative, but as we sample from the vocabulary only, we will always achieve the same probability.

Problem 7: In class we looked at a more general version of the bag of words model, in which different words in the vocabulary can be assigned different probabilities. We defined a function `sample-BOW-sentence`, which returns a sentence sampled from the bag of words model that we have specified. Below we have included a slight variant of the function which we defined in class. Previously the variables vocabulary and probabilities were defined outside of the function. In the current version, they are passed in as arguments. The function is identical otherwise.

```
(defn sample-BOW-sentence [len vocabulary probabilities]
  (if (= len 0)
    '()
    (cons (sample-categorical vocabulary probabilities)
          (sample-BOW-sentence (- len 1) vocabulary probabilities))))
```

The function `sample-BOW-sentence` allows us to sample a sentence given arbitrary probabilities for the words in our vocabulary. Let's make use of this power and define a distribution over the vocabulary which is better than the uniform distribution. We will use the word frequencies for our Moby Dick corpus to *estimate* a better distribution.

Above we defined the variable `moby-word-frequencies`, which contains the frequency of every word that occurs in our Moby Dick corpus. Using `normalize` and `moby-word-frequencies`, define a variable `moby-word-probabilities`. This variable should contain probabilities for every word in `moby-vocab`, in proportion to its frequency in the text. A word which occurs 2 times should receive twice as much probability as a word which occurs 1 time.

Answer 7: Please put your answer in `ps3-lastname-firstname.clj`.

Problem 8: Using `sample-BOW-sentence`, sample a 3-word sentence from a bag of words model, in which the probabilities are set to be those in `moby-word-probabilities`. Repeat this process at least three times, and write down the sentences that you collect through this process.

Answer 8: Please put the output sentences in the box below.

the stepping I methodically precisely the hand it almost

Problem 9: Define a function `lookup-probability`, which takes three arguments, `w`, `outcomes`, and `probs`. `probs` represents a probability distribution over the elements of `outcomes`. For example, if outcomes is `'(the a every)`, then `probs` may be `'(0.2 0.5 0.3)`. The first number in `probs` is the probability of the first element of outcomes, the second number in probs is the probability of the second element of outcomes, and so on.

`lookup-probability` should look up the probability of the element `w`. For example, if `w` is `'the`, then look-up probability should return `0.2`. If `w` is `'a`, then `lookup-probability` should return `0.5`. If `w` is not in the list of outcomes, the function should return that its probability is zero.

Answer 9: Please put your answer in `ps3-lastname-firstname.clj`.

Problem 10: Using `lookup-probability`, define a function `compute-BOW-prob` which takes three arguments, `sentence`, `vocabulary`, and `probabilities`. The arguments `vocabulary` and `probabilities` are used to define a bag of words model with the associated probability distribution over vocabulary words. The function should compute the probability of the sentence (which is a list of words) according to the bag of words model.

This function is a generalization of the function `compute-uniform-BOW-prob` that you defined above.

Answer 10: Please put your answer in `ps3-lastname-firstname.clj`.

Problem 11: In problem 8, you collected a number of 3-word sentences. These sentences were generated from a bag of words model in which the probabilities were set to those in `moby-word-probabilities`, which reflect the relative frequency of the words in the Moby Dick corpus. Use `compute-BOW-prob` to compute the probability of these sentences according to the bag of words model. How does your answer differ from problem 6?

Choose one of the 3-word sentences that you have generated. Can you construct a different sentence which has the same probability according to the bag of words model? When computing the probability of a sentence under a bag of words model, what information about the sentence suffices to compute this probability?

Answer 11: Please put your answer in the box below.

Sentence Probabilities:

1. the stepping I: 9.858×10^{-6}
2. methodically precisely the: 1.095×10^{-6}
3. hand it almost: 4.381×10^{-7}
4. stepping it almost: 4.381×10^{-7}

We no longer use the uniform distribution but a probability distribution based on the frequency of the words in the corpus, meaning that the probabilities of the individual words can differ because they occur with different frequency. Therefore, we do not always get the same probability for our sentence as in the uniform case. However, some words still have the same frequency in the corpus and therefore the same probability, which is why we can generate different sentences with the same probability. This can be observed with the sentences *hand it almost* and /textitstepping it almost, which both have the same probability, because the words *hand* and *stepping* occur equally often in the corpus (1 time). In general, the number of occurrences of each word in a sentence in the corpus is sufficient for calculating the probability in a bag of words model, since we can simply calculate 1 divided by the multiplication of these numbers.

LONG FORM READING QUESTION:

(This section is optional for students in LING/COMP 445, but must be completed if taking LING 645.)

You must answer this question on your own.

Elman's (1991) seminal work is an early example of how neural networks, here recurrent neural networks, can be used to model cognitive representations and learning processes. This work is part of a theoretical tradition called Connectionism which came from psychology and cognitive science and posited that all behaviours, especially linguistic, could be explained via simple neural connections, here examplified by neural network models. Elman discusses the distinctions between traditional computational models of language and neural network models. (1) Name and describe at least two of these distinctions. (2) Given that extracting information about linguistic representation from neural network models requires probing techniques, what is a limitation of this modeling approach? (Your answer should be 1/2 a page to a page in length.)

Answer: Please put your answer in the box below.

Traditional computational language models are often based on explicit rules or algorithms used to generate or analyze sentences. In contrast, neural network models use connectionist learning, where the network learns to make predictions or produce outputs based on patterns in the input data. This learning is often implicit and distributed across the network, rather than being explicitly programmed. The authors claim that the latter is however not necessarily a no-rule system, since it is unclear what exactly is meant by a rule. They therefore argue that the difference lies mainly in the nature of the rules used and what kind of information counts as explicitly present. Furthermore, traditional language models rely on discrete and abstract symbols representing linguistic units such as words and phrases. Neural network models, on the other hand, create representations corresponding to the activation behaviours of the individual nodes. Such representations are not bound to specific linguistic units, but result from the learning process of the network.

Probing techniques involve testing the behavior of the network on specific tasks to draw conclusions about the nature of the learned linguistic representation or tasks. However, it can be difficult to design these tests to accurately capture the relevant linguistic phenomena, and it can be difficult to interpret the results of these tests. Moreover, it is not always clear how the network's behavior on these tests generalizes to its behavior on other tasks or in other contexts.