COMP/LING 445/645
Problem Set 5

There are several types of questions below. For programming questions, please put your answers into a file called `pittard-victoria.clj`. Be careful to follow the instructions exactly and be sure that all of your function definitions use the exact names, number of inputs, input types, number of outputs, and output types as requested in each question. Please make sure all other variables are also named as indicated.

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 `pittard-victoria.clj`. You can find information about installing and using Clojure here `https://clojure.org/`.

For the code portion of the assignment, it is crucial to submit a standalone file that runs. Before you submit `pittard-victoria.clj` to us via email, make sure that your code executes correctly without any errors when run at the command line by typing `clojure 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 like produces an error, the code portion of the assignment will receive a 0.

For questions involving answers in English or mathematics or a combination of the two, put your answers to the question in an **Answer** section like in the example below. You can find more information about LaTeX here `https://www.latex-project.org/`.

Once you have answered the question, please compile your copy of this LaTeX document into a PDF and submit (i) the compiled PDF (ii) the raw LaTeX file and (iii) your `pittard-victoria.clj` file via email to *both* timothy.odonnell@mcgill.ca and savanna.willerton@mail.mcgill.ca. The problem is set is due before 16:05 on Wednesday, December 18, 2019.

Problem 0: This is an example question using some fake math like this $L = \sum_0^\infty \mathcal{G}\delta_x$.

Answer 0: Put your answer right under the question like this $L = \sum_0^\infty \mathcal{G}\delta_x$.

Background: Throughout this problem set we will be working with the Hidden Markov Model. In an HMM, there is a sequence of hidden states $c^{(1)}, ..., c^{(n)}$, and a sequence of observed words, $w^{(1)}, ..., w^{(n)}$. The set of all possible hidden states is S, and the set of all possible words is O. Thus $c^{(i)} \in S$ and $w^{(i)} \in O$ for all i. We defined the distribution on hidden states and words as follows in class:

$$\Pr(W^{(1)}, \dots, W^{(k)}, C^{(1)}, \cdots, C^{(k)}) = \prod_{i=1}^{k} \Pr(W^{(i)} \mid C^{(i)} = c^{(i)}, \vec{\theta}_{O,c^{(i)}}) \Pr(C^{(i)} \mid C^{(i-1)} = c^{(i-1)}, \vec{\theta}_{T,c^{(i-1)}}) \tag{1}$$

Note that we are assuming (for the time being) that the parameters $\vec{\theta}_{T,c^{(i-1)}}$ and $\vec{\theta}_{O,c^{(i)}}$ are fixed, i.e. there is no uncertainty about their value.

We will assume throughout that there are three hidden states (categories) and three words. These are defined by:

```
(def hidden-states '(Start N V))
```

```
(def vocabulary '(Call me Ishmael))
```

Note that, to make the exercises less painful, we are not including a `stop` state. There are three state transition distributions that we need to define, one for each of the hidden states: $\Pr(C^{(i)} \mid C^{(i-1)} = Start)$, $\Pr(C^{(i)} \mid C^{(i-1)} = N)$, and $\Pr(C^{(i)} \mid C^{(i-1)} = V)$. As stated in the definition of the HMM probabilities above, these distributions are specified by the parameters $\vec{\theta}_{T,Start}$, $\vec{\theta}_{T,N}$, and $\vec{\theta}_{T,V}$:

```
(def theta-transition-Start '(0 0.9 0.1))

(def theta-transition-N '(0 0.3 0.7))

(def theta-transition-V '(0 0.8 0.2))

(def theta-transition-dists-1
  (list theta-transition-Start theta-transition-N theta-transition-V))
```

This means that the hidden state N has probability 0.3 of transitioning to hidden state N, and probability 0.7 of transitioning to V. Note that none of the states ever transition to the `start` hidden state. This means that the `start` state can only occur as the first hidden state in a sequence, and never afterwards.

We also need to specify the observation distribution for the hidden states N and V (the hidden state `start` cannot produce observations, so it is ignored here). They are defined by:

```
(def theta-observation-N '(0.1 0.5 0.4))

(def theta-observation-V '(0.8 0.1 0.1))

(def theta-observation-dists-1
   (list theta-observation-Start
        theta-observation-N
        theta-observation-V))
```

This means, for example, that the hidden state N has probability 0.4 of emitting the word *Ishmael*. The following helper functions will be useful for writing your code.

```
(defn dist-lookup [state states dists]
  (if (= state (first states))
     (first dists)
     (dist-lookup state (rest states) (rest dists))))


(defn logsumexp [log-vals]
  (let [mx (apply max log-vals)]
    (+ mx
      (Math/log2
         (apply +
               (map (fn [z] (Math/pow 2 z))
                    (map (fn [x] (- x mx)) log-vals)))))))

(defn logscore-categorical [outcome outcomes params]
  (if (= outcome (first outcomes))
    (Math/log2 (first params))
    (logscore-categorical outcome (rest outcomes) (rest params))))
```

Problem 1: Suppose that at time t the hidden state is $c^{(t)}$. Then we can compute the probability that the next hidden state is $c^{(t+1)}$ and the next observed word is $w^{(t+1)}$. This probability is equal to:

$$\begin{aligned}&\Pr(W^{(t+1)} = w^{(t+1)}, C^{(t+1)} = c^{(t+1)} \mid C^{(t)} = c^{(t)})\\ =&\Pr(W^{(t+1)} = w^{(t+1)} \mid C^{(t+1)} = c^{(t+1)}) \Pr(C^{(t+1)} = c^{(t+1)} \mid C^{(t)} = c^{(t)})\end{aligned}$$

Write a procedure `score-next-state-word` which computes the log of this probability. It should take five arguments: the current hidden state, the next hidden state, and next observed word, the hidden state transition distributions, and the observation distributions. A call to `(score-next-state-word current-hidden next-hidden next-observed theta-transition-dists theta-observation-dists)` should return the log probability of the next hidden state and next observed word, given the current hidden state.
Hint: use the functions `dist-lookup` and `logscore-categorical` defined above.

Answer 1: Answer in `pittard-victoria.clj`.

Problem 2: Suppose that at time t the hidden state is $c^{(t)}$. Then we can compute the *marginal* probability of the word $w^{(t+1)}$ at time $t+1$ given the hidden state at time t. This probability is equal to:

$$\begin{aligned}&\Pr(W^{(t+1)} = w^{(t+1)} \mid C^{(t)} = c^{(t)}) &(2)\\ =&\sum_{c^{(t+1)}} \Pr(W^{(t+1)} = w^{(t+1)}, C^{(t+1)} = c^{(t+1)} \mid C^{(t)} = c^{(t)}) &(3)\\ =&\sum_{c^{(t+1)}} \Pr(W^{(t+1)} = w^{(t+1)} \mid C^{(t+1)} = c^{(t+1)}) \Pr(C^{(t+1)} = c^{(t+1)} \mid C^{(t)} = c^{(t)}) &(4)\end{aligned}$$

That is, we sum over all of the possible hidden states that could appear at time $t+1$.

Using the procedure `score-next-state-word` that you wrote in Problem 1, write a function `compute-next-observation-marginal`, which takes four arguments: `current-state`, `next-observation`, `theta-transition-dists`, and `theta-observation-dists`. When `(compute-next-observation-marginal current-state next-observation theta-transition-dists theta-observation-dists)` is called, it should return the marginal probability of the next observed word given the current hidden state.

Hint: use `logsumexp`.

Answer 2: Answer in `pittard-victoria.clj`.

Problem 3: Suppose that at time t the hidden state is $c^{(t)}$. Then we can compute the probability of the next k hidden states and next k observed words given the hidden state at time t. This probability is equal to:

$$\begin{aligned}&\Pr(W^{(t+1)}, \ldots, W^{(t+k)}, C^{(t+1)}, \cdots, C^{(t+k)} \mid C^{(t)} = c^{(t)}) &(5)\\ =&\prod_{i=t}^{t+k-1} \Pr(W^{(i+1)} = w^{(i+1)} \mid C^{(i+1)} = c^{(i+1)}) \Pr(C^{(i+1)} = c^{(i+1)} \mid C^{(i)} = c^{(i)}) &(6)\\ =&\Pr(C^{(t+1)} = c^{(t+1)} \mid C^{(t)} = c^{(t)}) \Pr(W^{(t+1)} = w^{(t+1)} \mid C^{(t+1)} = c^{(t+1)}) &(7)\\ &\cdot \Pr(W^{(t+2)}, \ldots, W^{(t+k)}, C^{(t+2)}, \cdots, C^{(t+k)} \mid C^{(t+1)} = c^{(t+1)}) &(8)\end{aligned}$$

Note that the equation shows that there is a recursive formula for this probability.

Write a procedure `score-next-states-words` which computes this probability. The function should take five arguments: the current hidden state, a list of k hidden states, a list of k observed words, the transition distributions, and the observation distributions. When `(score-next-states-words current-hidden next-hidden-states next-words theta-transition-dists theta-observation-dists)` is called, it should return the log probability that the k hidden states and k observed words will appear immediately following the current hidden state.

You should use recursion to write the function.

Answer 3: Answer in `pittard-victoria.clj`.

Problem 4: Suppose the at time t the hidden state is $c^{(t)}$. Then we can compute the marginal probability of the next k observed words given this hidden state, i.e. summing out all of the possible settings of the hidden states. This probability is equal to:

$$\Pr(W^{(t+1)}, \ldots, W^{(t+k)} \mid C^{(t)} = c^{(t)}) \tag{9}$$

$$= \sum_{c^{(t+1)}, \cdots c^{(t+k)}} \Pr(W^{(t+1)}, \ldots, W^{(t+k)}, C^{(t+1)}, \cdots, C^{(t+k)} \mid C^{(t)} = c^{(t)}) \tag{10}$$

$$= \sum_{c^{(t+1)}, \cdots c^{(t+k)}} \prod_{i=t}^{t+k} \Pr(W^{(i+1)} = w^{(i+1)} \mid C^{(i+1)} = c^{(i+1)}) \Pr(C^{(i+1)} = c^{(i+1)} \mid C^{(i)} = c^{(i)}) \tag{11}$$

$$= \sum_{c^{(t+1)}} \Pr(C^{(t+1)} = c^{(t+1)} \mid C^{(t)} = c^{(t)}) \Pr(W^{(t+1)} = w^{(t+1)} \mid C^{(t+1)} = c^{(t+1)}) \tag{12}$$

$$\cdot \Pr(W^{(t+2)}, \ldots, W^{(t+k)}) \mid C^{(t+1)} = c^{(t+1)}) \tag{13}$$

The final equation is very important. It shows that the marginal probability of the words following hidden state $c^{(t)}$ can be computed in a recursive manner. Write a function `compute-next-words-marginal`, which computes the log marginal probability of a sequence of observed words given the current hidden state. The function should take four arguments: the current hidden state, a list of words, the transition distributions, and the observation distributions. When `(compute-next-words-marginal current-hidden next-words theta-transition-dists theta-observation-dists)` is called, it should return the log of the marginal probability of the list of words given the hidden state, as defined in the equation above.

You should use recursion to write your function.

Answer 4: Answer in `pittard-victoria.clj`.

Problem 5: Find a sequence of two words $w^{(1)}, w^{(2)}$ such that the marginal probability of this sequence is different than the marginal probability of its reverse. That is, find $w^{(1)}, w^{(2)}$ such that:

$$\Pr(W^{(1)} = w^{(1)}, W^{(2)} = w^{(2)} \mid C^{(0)} = Start) \neq \Pr(W^{(1)} = w^{(2)}, W^{(2)} = w^{(1)} \mid C^{(0)} = Start)$$

Use your solution to Problem 4 to confirm that your sequences have different probabilities, and explain why this is true.

Answer 5: Code used in obtaining answer in `pittard-victoria.clj`.
The sequence of words "Call me" has a different marginal probability than its reverse, "me Call". The marginal probability of a sequence of words depends on two key factors: the probability of the word occurring (constant each time the word appears) and the transition probability between the previous state to the current one (or sum of probabilities, should several options exist). These factors are multiplied together for every probability. Thus, the probability of words 2-k occurring considers the probability that the previous word was a noun

or a verb, and that the current word is a noun or a verb (four probabilities total), and then factors in the probability that the word occurs given these states. The only time where variation occurs, is thus in the first word, as the initial state 0 is Start. Start has a different set of transition probabilities, and as they are multiplied with the probability of a word occurring, the following holds true:
"Call me" is the product of the probability that Call is first (aka start –> noun or verb, multiplied by noun or verb –> Call), and me is second (aka noun or verb –> noun or verb, multiplied by noun or verb –> me).
"me Call" is the product of the probability that me is first (aka start –> noun or verb, multiplied by noun or verb –> me), and Call is second (aka noun or verb –> noun or verb, multiplied by noun or verb –> Call).
Thus, as the start transition distribution has different values to the sum of the other transition distributions, the word order causes Call me and me Call to have different marginal probabilities. This is confirmed by calling `compute-next-words-marginal` on both Call me and me Call, yielding marginal probabilities (log probabilities) of approximately -4.2270 and -1.9002, respectively.

Problem 6: In the next several problems, we will be using your solution to Problem 4 to perform Bayesian inference in the HMM. Our goal will be to compute the posterior distribution over hidden states given a sequence of words. Recall that Bayes' Rule tells us how to calculate this posterior distribution:

$$\Pr(C^{(1)} = c^{(1)}, \cdots, C^{(t)} = c^{(t)} \mid W^{(1)} = w^{(1)}, \ldots, W^{(t)} = w^{(t)}) = \tag{14}$$

$$\frac{\Pr(C^{(1)} = c^{(1)}, \cdots, C^{(t)} = c^{(t)}) \Pr(W^{(1)} = w^{(1)}, \ldots, W^{(t)} = w^{(t)} \mid C^{(1)} = c^{(1)}, \cdots, C^{(t)} = c^{(t)})}{\Pr(W^{(1)} = w^{(1)}, \ldots, W^{(t)} = w^{(t)})} \tag{15}$$

Write a procedure `compute-hidden-prior`, which takes two arguments: a list of k hidden states and the hidden state transition distributions. When `(compute-hidden-prior hidden-states theta-transition-dists)` is called, it should return the log prior probability of the hidden state sequence, i.e. the log value of

$$\Pr(C^{(1)} = c^{(1)}, \cdots, C^{(k)} = c^{(k)})$$

.

Answer 6: Answer in `pittard-victoria.clj`.

Problem 7: In this problem, we will continue with the Bayesian calculation from Problem 6. Write a procedure `compute-likelihood-of-words`, which takes three arguments: a list of k hidden states, a list of k words, and the observation distributions. When `(compute-likelihood-of-words hidden-states words theta-observation-dists)` is called, it should return the log probability of the k words being generated from the hidden states, i.e. the log value of $\Pr(W^{(1)} = w^{(1)}, \ldots, W^{(k)} = w^{(k)} \mid C^{(1)} = c^{(1)}, \cdots, C^{(k)} = c^{(k)})$.

Answer 7: Answer in `pittard-victoria.clj`.

Problem 8: We are finally ready to solve our Bayesian inference problem. Write a procedure `compute-hidden-posterior`, which takes four arguments: a list of k hidden states, a list of k words, the hidden state transition distributions, and the observation distributions. When `(compute-hidden-posterior hidden-states words theta-transition-dists theta-observation-dists)` is called, it should return the log posterior probability of the k hidden states given the k words, i.e. it should return the log value of $\Pr(C^{(1)} = c^{(1)}, \cdots, C^{(k)} = c^{(k)} \mid W^{(1)} = w^{(1)}, \ldots, W^{(k)} = w^{(k)})$.
Hint: Use your solutions to Problems 4, 6, and 7.

Answer 8: Answer in `pittard-victoria.clj`.

Problem 9: Suppose that we use the function `compute-next-words-marginal` that you wrote in Problem 4, in order to compute the marginal probability of a sequence of k observed words. How many times is compute-next-words-marginal called during the execution of the program? What does this mean about the run-time of the program?
Hint: your formula should involve the number of hidden states and the length k of the sequence of words.

Answer 9: Using the function `compute-next-words-marginal` to compute the marginal probability of a sequence of k words calls the function `compute-next-words-marginal` n^k times throughout the program, where n is the number of states and k is the number of words. This was confirmed by running the function with added print statements. It makes logical sense, as it maps over hidden-states (n times) and in this map call, calls itself. The base case occurs when next-words is empty (k words), thus causing the function to be called n^k times (in other words, n times for each of k words). This indicates a runtime of O(n^k), assuming the other function calls (not recursive) occur trivially. In other words, the runtime of the program increases exponentially with the number of words.

Problem 10: In class, we introduced *memoization* as a method for sharing computation and/or reducing the run-time of programs. Memoization refers to a technique where we intercept calls to a function with some arguments, check if the function has been called with those arguments before and, if not, call the function on those arguments, get the result, store it in a table and return it. On subsequent calls to that function with those arguments, we simply look up the result in the table and avoid recomputing the value. In Clojure, functions can be memoized by a call to `memoize`.

Using `memoize`, define a memoized version of your function `compute-next-words-marginal`. Name this function `compute-next-words-marginal-mem`.

Answer 10: Answer in `pittard-victoria.clj`.

Problem 11: Suppose that we use the memoized function `compute-next-words-marginal-mem` that you wrote in the previous problem, in order to compute the marginal probability of a sequence of k observed words. How many times is `compute-next-words-marginal-mem` called during the execution of the program? How does this differ from the non-memoized version of the procedure?

Answer 11: In the memoized function, there are nk recursive calls when computing the marginal probability of a sequence of k observed words (where n hidden states exist). This results in less recursive calls, and thus a faster O(nk) runtime (pseudopolynomial rather than exponential) when compared to the non-memoized function. This is because memoization keeps track of results, thus enabling fast lookup of calls that are repeated (for example, repeated words, as hidden-states is constant). To further compare the two functions, while runtime is initially similar (i.e. 2 states and 2 words, as both $2 * 2$ and 2^2 equal 4), it increases at drastically different rates with the number of states (in this case, constant at 3) and words (i.e. 3 states and 8 words, as $3 * 8 = 24$ (memoized), while $3^8 = 6561$ (non-memoized)).