CS221 Exam Solutions

CS221
Spring 2019

Name: $\underbrace{\hspace{6cm}}_{\text{by writing my name I agree to abide by the honor code}}$

SUNet ID: ______________________

Read all of the following information before starting the exam:

- This test has 3 problems printed on 32 pages and is worth 180 points total. It is your responsibility to make sure that you have all of the pages.
- Only the printed (top) side of each page will be scanned so write all your answers on that side of the paper.
- Keep your answers precise and concise. We may award partial credit so show all your work clearly and in order.
- Don't spend too much time on one problem. Read through all the problems carefully and do the easier ones first. Try to understand the problems intuitively; it really helps to draw a picture.
- You cannot use any external aids except one double-sided $8\frac{1}{2}$" x 11" page of notes.
- Good luck!

Problem	Part	Max Score	Score
1	a	18	
	b	15	
	c	18	
	d	9	
2	a	13	
	b	10	
	c	11	
	d	16	
3	a	20	
	b	19	
	c	15	
	d	16	

Total Score: $\boxed{} + \boxed{} + \boxed{} = \boxed{}$

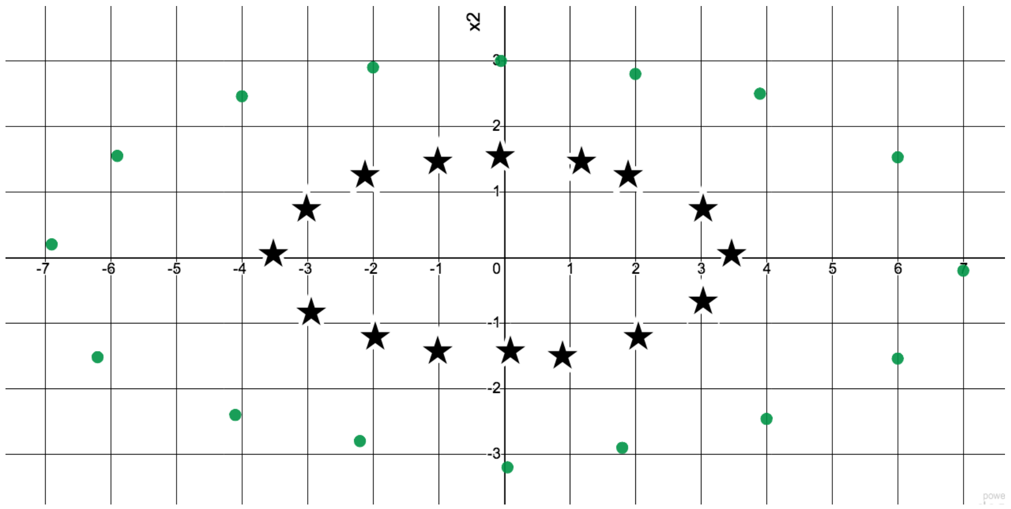
1. Machine Learning (*60 points*) In an alternate universe, the course staff of CS 221 does not release the mechanism by which final grades are computed. As an enterprising student, you decide to procrastinate by trying to predict final course grades.

You have obtained access to data points (x, y) for each person, where $x \in \mathbb{R}^6$ gives 6 pieces of information about each student's grades in previous classes and progress so far in CS 221 (you don't need to know what those are). You wish to design a model that predicts the student's final grade: A, B, C, D, or F.

(a) Binary classification. (*18 points, 6 points each*)

You start by trying to solve a simpler problem: only using the data $x^{(i)} = (x_1^{(i)}, x_2^{(i)})^T \in \mathbb{R}^2$ for each person i in the historical database, you wish to predict whether student i passes the class, label z. Thus, $z = 1$ when the student's final grade is in $\{A, B, C\}$ and they pass, and $z = -1$ when the student receives a $\{D, F\}$ and thus NC (no credit) for the class. You have N training examples, $x^{(1)}, \ldots, x^{(N)}$ (viewing them in $\mathbb{R}^2$) with associated pass/fail labels $z^{(1)}, \ldots, z^{(N)} \in \{-1, 1\}$.

(i) Before applying a machine learning algorithm, you plot $x^{(1)}, \ldots x^{(N)}$ and observe the following, where circle dots have label $z = 1$ and star dots have label $z = -1$:



Based on this plot, give an expression for one additional *polynomial feature* $\phi(x^{(i)})$, you would add to the vector $x^{(i)}$ to train on, that would likely improve the performance of a machine learning model that used a *linear predictor*, so $x^{(i)} := (x_1^{(i)}, x_2^{(i)}, \phi(x^{(i)}))^T$.

Solution $x_1^2/49 + x_2^2/9$

(ii) After playing around with the data, you realize you have many poorly classified points where z and your prediction $\hat{z} = \theta^T x$ are far from each other. Thus, you wish to penalize large losses linearly, but wish to use a squared loss for non-outlier data points. Given this background, construct a continuous, differentiable, loss function ℓ_1 in the piecewise format below:

$$\ell_1(z, \widehat{z}) = \begin{cases} ? & |z - \widehat{z}| < 1 \\ ? & \text{otherwise} \end{cases}$$

What loss function $L(\theta)$ (with respect to the penalty ℓ_1) do you minimize to find the optimal predictor θ ?

Solution $L(\theta) = \sum_{i=1}^{N} \ell_1(z^{(i)}, \theta^\top x^{(i)})$ where

$$\ell_1(z, \widehat{z}) = \begin{cases} \frac{1}{2}(z - \widehat{z})^2 & |z - \widehat{z}| < 1 \\ |z - \widehat{z}| - \frac{1}{2} & \text{otherwise} \end{cases}$$

(iii) With a step size of $\eta = 0.1$ and θ initialized to all zeros, manually compute one step of a training update using stochastic gradient descent if you randomly select data point $x^{(1)} = (0.9, 0.5), z^{(1)} = 1$ (you do not need to use the feature from (i)).

Solution We compute the gradient of $\ell_1(z^{(j)}, \theta^\top x^{(j)})$ with respect to θ as

$$\nabla \ell_1(z^{(j)}, \theta^\top x^{(j)}) = \begin{cases} -x^{(j)} \cdot (z^{(j)} - \theta^\top x^{(j)}) & |z^{(j)} - \theta^\top x^{(j)}| < 1 \\ -x^{(j)} \cdot \operatorname{sign}(z^{(j)} - \theta^T x^{(j)}) & \text{otherwise} \end{cases}.$$

Then we update $\theta := \theta - \eta \cdot \nabla \ell_1(z^{(j)}, \theta^\top x^{(j)})$, so $\theta = (0.09, 0.05)$

(b) Multiclass classification (*15 points, 5 points each*)

Now you are ready to work with the more complex data set. You are given the data $x \in \mathbb{R}^6$ for each student described above, and wish to predict their final letter grade y. This label comes in the form of a *one-hot vector*, i.e., for each student with data $x \in \mathbb{R}^6$, they have label $y \in \mathbb{R}^5$ where if the student got an A, $y = [1, 0, 0, 0, 0]$, if the student got a B, $y = [0, 1, 0, 0, 0]$, etc.

(i) Using a linear predictor as before, you wish to find Θ such that $\Theta^\top x^{(i)} = \widehat{y}$ is approximately $y^{(i)}$, where $y^{(i)}$ is the true grade of student i with data $x^{(i)}$. What are the dimensions of Θ? What is the loss function $L(\Theta)$ on training data $x^{(1)}, \ldots, x^{(N)}$ if you wish to minimize the norm of the 6-dimensional loss vector (computed component-wise using loss ℓ_1 from part (a))?

Solution $L(\theta) = \sum_{i=1}^{N} \sqrt{\sum_{j=1}^{5} (\ell_1(y_j^{(i)}, (\Theta^\top x^{(i)})_j)^2}$

(ii) You wish to interpret your prediction $\widehat{y} \in \mathbb{R}^5$ as a probability vector (with the entries denoting the probability of the student receiving each of the possible final course grades). Describe why this interpretation is flawed in the current prediction model and propose a mechanism to post-process the predictions so that this interpretation. Note: you should view negative probabilities as equivalent to 0 probability.

Solution The prediction $\widehat{y}$ may not have nonnegative entries that sum to 1. We can use $\widehat{y} := \frac{(y)_+}{\mathbf{1}^\top (y)_+}$ or we can also accept softmax $\widehat{y} = (\frac{e^{y_i}}{\sum_{i=1}^5 e^{y_i}})$

(iii) When working in probability space, a common loss function between two probability vectors $y, \widehat{y} \in \mathbb{R}^5$ is the *Kullback–Leibler (KL) divergence*:

$$\ell_{KL}(y, \widehat{y}) = \sum_{i=1}^{5} y_i \log\left(\frac{y_i}{\widehat{y}_i}\right)$$

What is the gradient of the loss function $L(\Theta)$ computed using a linear predictor, the ℓ_{KL} loss, and training data $(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})$? Note: you can assume logarithms are base e, no post-processing of predictions, and can give the gradient element-wise.

Solution Let Θ_i^T be the ith row of Θ. Then,

$$L(\Theta) = \sum_{j=1}^{N} \sum_{i=1}^{5} y_i^{(j)} \log\left(\frac{y_i^{(j)}}{\Theta_i^T x^{(j)}}\right)$$

Then to compute $\nabla_\Theta L(\Theta)$, we compute the partial derivative

$$\frac{\partial}{\partial \Theta_{ik}} L(\Theta) = \sum_{j=1}^{N} y_i^{(j)} \cdot \frac{1}{\frac{y_i^{(j)}}{\Theta_i^T x^{(j)}}} \cdot -\frac{y_i^{(j)} x_k^{(j)}}{(\Theta_i^T x^{(j)})^2} = -\sum_{j=1}^{N} \frac{y_i^{(j)} x_k^{(j)}}{\Theta_i^T x^{(j)}}$$

Then $\nabla_\Theta L(\Theta)$ is an 5×6 matrix with ik-entry as given above by the partial derivative.

(c) Loss functions (*18 points, 2 points each*)

We have studied several loss functions in this course:

(a) $\ell_{sq}(z, \widehat{z}) = (z - \widehat{z})^2$
(b) $\ell_{0-1}(z, \widehat{z}) = \mathbf{1}[z \neq \widehat{z}]$
(c) $\ell_{abs}(z, \widehat{z}) = |z - \widehat{z}|$,
(d) ℓ_1
(e) ℓ_{KL}

In each of the following situations, select the most appropriate loss function choice:

(i) You wish to penalize most incorrect predictions quadratically but not outliers.

(a) *(b)* *(c)* *(d)* *(e)*

(ii) Incorrect predictions that are very far from the true label are much worse than incorrect predictions close to the true value.

(a) *(b)* *(c)* *(d)* *(e)*

(iii) You wish to use your loss function to compute the number of examples predicted incorrectly in binary classification

(a) *(b)* *(c)* *(d)* *(e)*

(iv) Incorrect predictions far from the true value should be penalized more than incorrect predictions close to the true value, but data is very noisy.

(a) *(b)* *(c)* *(d)* *(e)*

(v) You wish to have (ideally) 0 predictions very far away from the true label.

(a) *(b)* *(c)* *(d)* *(e)*

(vi) Incorrect predictions are bad proportional to their distance to the true value.

(a) *(b)* *(c)* *(d)* *(e)*

(vii) You wish for the predictions to always have strictly positive entries.

(a) *(b)* *(c)* *(d)* *(e)*

(viii) Incorrect predictions far from the true value are equally as bad as incorrect predictions close to the true value.

(a) *(b)* *(c)* *(d)* *(e)*

(ix) You wish to pick parameters to maximize the log likelihood of the data.

(a) *(b)* *(c)* *(d)* *(e)*

Solution

(i) $(d) : \ell_1$

(ii) $(a) : \ell_{sq}$

(iii) $(b) : \ell_{0-1}$

(iv) $(d) : \ell_1$

(v) $(a) : \ell_{sq}$

(vi) $(c) : \ell_{abs}$

(vii) $(e) : \ell_{KL}$

(viii) $(b) : \ell_{0-1}$

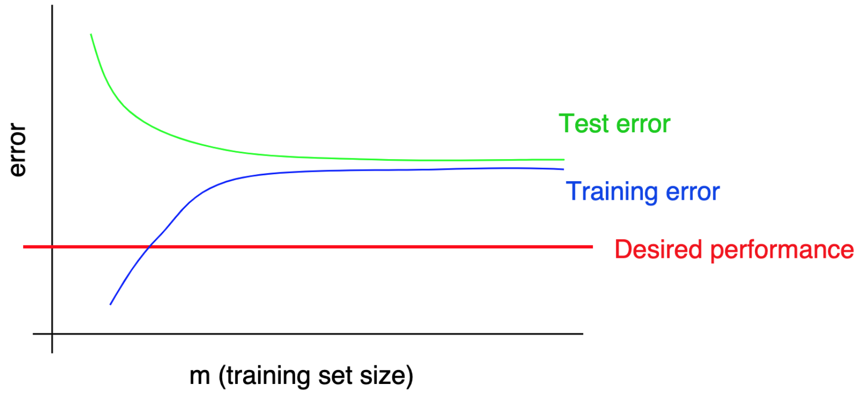
(ix) $(e) : \ell_{KL}$

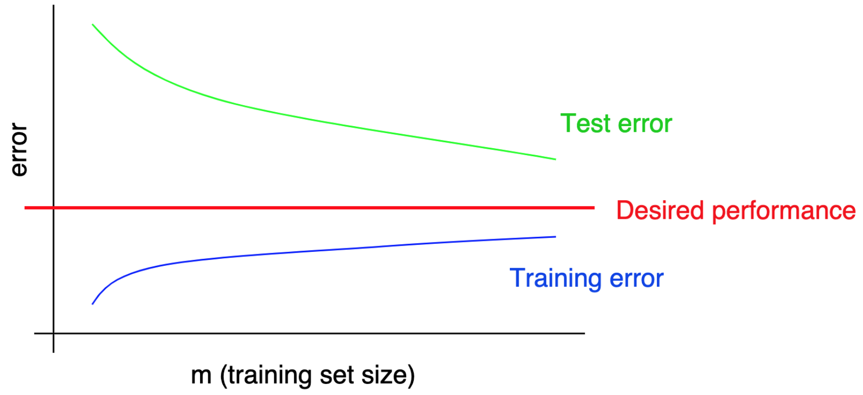
(d) Performance (*9 points, 3 points each*)

(i) You train a machine learning model $L(\theta)$ obtained in 1(a) and find that it has a training accuracy of 0.95 and a test accuracy of 0.46. What does this imply about the model's performance? Provide one way $L(\theta)$ can be adjusted to increase the test accuracy.

Solution Add regularization by $\alpha||\theta||_2$ for $\alpha > 0$ a hyperparameter.
Credit was not given to answers that suggested using less features in the training set or early stopping as those are not adjustments to $L(\theta)$. Though L1 (Lasso) regularization was accepted as an answer if explained correctly or a correct equation was given. Just stating 'add regularization' was not sufficient

(ii) You use various subsets of the data and plot your training error as a function of training set size m. Explain whether each of the following plots demonstrates overfitting (overly complex model) or underfitting (insufficiently complex model), and give a 1-2 sentence explanation for each.





Solution The first image has high bias as it has high training and testing error (underfitting), and the second has high variance as it overfits the training data (overfitting)

(iii) Explain in 1 sentence what effect (if any) each of the following will have on approximation error and estimation error:

i. Using a smaller set of features

ii. Getting more training examples

Solution A smaller set of features will create a simpler model with decreased variance and complexity, and thus lower estimation error. However, approximation error will likely increase with our reduced hypothesis class size.

More training examples will lead to less overfitting, which will also lower variance and give a better estimation error. However, approximation error will likely stay the same because the hypothesis class size will stay the same

2. Pokémon (*50 points*) A famous Pokémon trainer, Jaebum, is trying to capture as many as Pokémons he can find on a $N \times M$ grid.

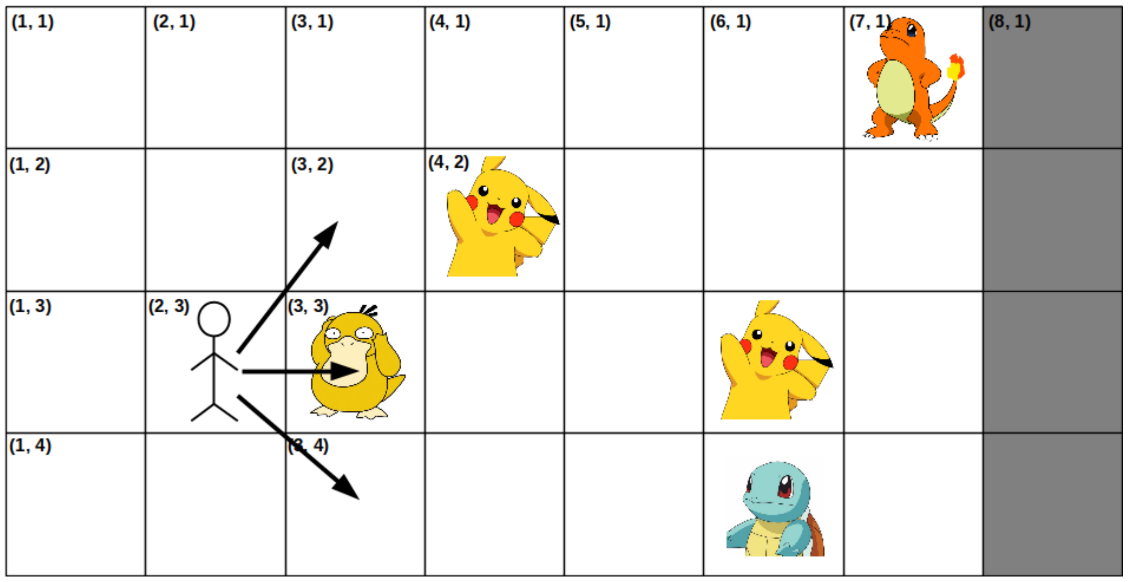


Figure 1: Pokémons on the 4×8 grid; Last column (colored in grey) is the end point.

As you can see, Pokémons reside in cells and at most one Pokémon can reside in a single cell.

At every time step, Jaebum can only move right or diagonal to the right as shown in the picture. In other words, if Jaebum is at (x, y), the next place he can be is either one of $(x+1, y), (x+1, y-1), (x+1, y+1)$ if that position is within the grid.

If he arrives at a cell where the Pokémon resides, he can decide whether to capture the Pokémon or not. If he decides to capture the Pokémon, since he is a very skillful Pokémon trainer, he can always capture a Pokémon with 100% probability.

Since Pokémons are heavy, **the cost of moving at each step is equal to the sum of the weights of all currently captured Pokémons**[1]. The weight of the Pokémon at location (x, y) is given by $W(x, y)$.

For example, suppose Jaebum has captured Pokémons at $(3, 3)$ and $(4, 2)$. Now, if he tries to move from $(4, 2)$ to $(5, 1)$, then the cost of moving will be $W(3, 3) + W(4, 2)$. And then if he moves from $(5, 1)$ to $(6, 1)$, the cost of doing so will also be $W(3, 3) + W(4, 2)$ since he is only carrying those two Pokémons.

His journey ends when he reaches the Mth column of the grid (i.e (M, y) for any $1 \leq y \leq N$). For convenience, there is no Pokémon in the last column.

The size of the grid is $N \times M$, where N is the number of rows and M is the number of columns. Jaebum's initial location is (x_{start}, y_{start}) and there are no Pokémon at the initial location.

[1] You can think of it as the energy required to carry Pokémons

a. Uniform Cost Search (*13 points*)

Jaebum needs to capture at least P Pokémons where P is some integer $P > 0$. To make his journey as easy as possible, Jaebum wants to minimize the total cost of carrying Pokémons around. We are going to formulate the search problem that finds the path that minimizes the total cost of carrying captured Pokémons while he can capture at least P Pokémons.

You may use the function $\textsc{IsPoké}(x, y)$ which returns 1 if there exists a Pokémons at (x, y) and returns 0 otherwise. Also, $W(x, y)$ will return 0 when there is no Pokémon at (x, y). Otherwise, it will return the weight of Pokémon at (x, y).

For convenience, below is the set of every possible action.

$$\textsc{Actions} = \{(a_y, a_c) | a_y \in \{-1, 0, 1\} \text{ and } a_c \in \{\text{Capture}, \text{Not Capture}\}\}$$

The way to interpret this is that **your y coordinate (the row) changes by a_y first and then do what a_c says at the new location.**

For example, if you are at $(3, 2)$ and your action is $(0, \text{Capture})$, then you can capture Pikachu at $(3 + 1, 2 + 0) = (4, 2)$. On the other hand, if you are at $(2, 3)$, then the action $(-1, \text{Capture})$ won't be a valid action because there is no Pokémon at $(2+1, 3-1) = (3, 2)$.

Keep in mind that you move first, and then decide to capture or not capture. Because we know that Jaebum will always move one unit to the right, we don't have to specify the difference of x in the action.

From here, we are going to formulate the problem so that we can use the *uniform cost search* algorithm to solve it.

(i) [2 points] Define the state for the search problem and specify what the start state is.

Solution State can be defined as the position of Jaebum, the number of Pokémons he carries and the total weight of Pokémons in the bag at that moment.

Solution Thus, the start state will be $(x_{start}, y_{start}, 0, 0)$

(ii) [1 points] Define $\textsc{IsEnd}(s)$.

Solution Let $s = (s_x, s_y, s_n, s_w)$. Then, $s_x = M$ and $s_n \geq P$

(iii) [3 points] Given an action a, formulate a function $\textsc{IsValidAction}(s, a)$ that tells us whether taking an action a at the state s is valid. Here, a can be an arbitrary action in set $\textsc{Actions}$.

Solution Let $s = (s_x, s_y, s_n)$. Then, $\textsc{IsValidAction}((s_x, s_y, s_n, s_w), a)$ returns true when following is true.

- $1 \leq s_x + 1 \leq M$ and $1 \leq s_y + a_y \leq N$
- a_c is *Capture* only when $\textsc{IsPoké}(s_x + 1, s_y + a_y)$ is 1.

Otherwise, it returns false.

(iv) [7 points] Finally, formulate $\textsc{Succ}(s, a)$ and $\textsc{Cost}(s, a)$. Note that you will be given only actions that are valid for the state s.

Solution

Solution Let $s = (s_x, s_y, s_n, s_w)$. For convenience, let's define $(x_{next}, y_{next}) = (s_x + 1, s_y + a_y)$. Then,

$$\textsc{Succ}((s_x, s_y, s_n, s_w), a) = \begin{cases} (x_{next}, y_{next}, s_n + 1, s_w + W(x_{next}, y_{next})) & \text{if } a_c \text{ is } \textit{Capture} \\ (x_{next}, y_{next}, s_n, s_w) & \text{if } a_c \text{ is } \textit{Not Capture} \end{cases}$$

Also,

$$\textsc{Cost}((s_x, s_y, s_n, s_w), a) = s_w$$

b. Relaxed Heuristics (*10 points*)

Jaebum decided to use the A^* algorithm to enhance his search speed, which needs some heuristic function. The heuristic function that he came up with is as follows:

$$h(s) = s_w \cdot (M - s_x),$$

where s_w is the weight of his bag at the state s and s_x is the x coordinate of the state s. Is this a consistent heuristic? If so, prove it. If not, show a counter-example.

Solution This is a consistent heuristic. Here is the prove. First, obviously, $h(s_{end}) = 0$ as $s_x = M$ at the end state. Now, we have to show that following is true.

$$\textsc{Cost}(s, a) + h(\textsc{Succ}(s, a)) - h(s) \geq 0 \tag{1}$$

There are two possible cases. First, think of the case when a_c is *Capture*. Let's define the weight of the Pokémon at the successive location as W'. Then, we get

$$\begin{aligned}\textsc{Cost}(s, a) + h(\textsc{Succ}(s, a)) - h(s) &= s_w + (s_w + W')(M - (s_x + 1)) - s_w(M - s_x) \\ &= s_w - s_w + W'(M - s_x - 1) \\ &= W'(M - s_x - 1).\end{aligned}$$

Since $s_x < M$, we can see that $\textsc{Cost}(s, a) + h(\textsc{Succ}(s, a)) - h(s) \geq 0$. Finally, consider when a_c is *Not capture*. Then, $h(\textsc{Succ}(s, a)) - h(s) = -s_w$ and $\textsc{Cost}(s, a) = s_w$. Hence, $\textsc{Cost}(s, a) + h(\textsc{Succ}(s, a) - h(s) = 0$. Thus, we can see that $h(s)$ always satisfy the condition 1. Therefore, $h(s)$ is a consistent heuristic.

c. MDP (*11 points*)

Since carrying Pokémons is cumbersome, Jaebum brought K Poké balls. Any Pokémon captured with a Poké ball weighs zero. He no longer has to worry about the weight of captured Pokémons [2].

Instead, now he tries to focus on the value of captured Pokémons, which he wants to maximize. Since he only has K Poké balls, the maximum number of Pokémons he can capture is K. His goal is to reach the end point with the total value of captured Pokémons as large as possible.

Furthermore, because of the unstable nature of the Poké ball, for each Pokémon, there is some probability p that the Poké ball is unable to contain the Pokémon. In that case, the Pokémon will run away and we will no longer be able to capture it again. However, we can still reuse the Poké ball. Also, once the Poké ball successfully captures the Pokémon, then the Pokémon stays in the Poké ball forever.

Formulate this problem using an MDP. **For this problem, please assume that the size of the grid is** $1 \times M$. In other words, Jaebum only moves to its horizontal right, ignoring diagonal moves.

The value of the Pokémon at $(x, 1)$ is denoted as $R(x)$. Also, the probability that the Poké ball fails to capture Pokémon at $(x, 1)$ is $p(x)$.

Since we know that Jaebum only moves horizontally, we don't need to specify directions anymore as he will always move right. All we need to care about is whether Jaebum will try to capture a Pokémon on the next location or not. In other words, ACTIONS will be represented as:

$$\text{Actions} = \{\text{Capture}, \text{Not Capture}\}$$

Just like before, the action chosen denotes what Jaebum does after he moves. If he is at $(3, 1)$ and the action is *Capture*, then that means he will be capturing a Pokémon after he arrives at $(4, 1)$.

[2] Each Poké ball ignores the law of conservation of mass.

(i) [2 points] Please specify how to define a **minimal form of the state** and what the initial state is. Note that Jaebum is at $(1, 1)$ in the beginning.

Solution The state will be (x, k) where x is the current location of Jaebum and k is remaining number of Poké balls. Thus, the initial state will be $(1, K)$

(ii) [3 points] Define the transition probability, $T(s, a, s')$. You may assume that the action given to this function is always valid (i.e., Jaebum does not try to capture when there is no Pokémon).

Solution

$$\begin{aligned}
&T((s_x, s_k), \text{Capture}, (s_x + 1, s_k)) = p(s_x + 1) \\
&T((s_x, s_k), \text{Capture}, (s_x + 1, s_k - 1)) = 1 - p(s_x + 1) \\
&T((s_x, s_k), \text{Not Capture}, (s_x + 1, s_k)) = 1
\end{aligned}$$

(iii) [3 points] Define the reward $\textsc{Reward}(s, a, s')$.

Solution

$$\begin{aligned}
&R((s_x, s_k), \text{Capture}, (s_x + 1, s_k)) = 0 \\
&R((s_x, s_k), \text{Capture}, (s_x + 1, s_k - 1)) = R(s_x + 1) \\
&R((s_x, s_k), \text{Not Capture}, (s_x + 1, s_k)) = 0
\end{aligned}$$

(iv) [1 points] What is the value of discount factor γ?

Solution $\gamma = 1$

(v) [2 Points] Suppose we use value iteration to compute the optimal values for the above MDP. If you can choose the order you iterate the states in the value iteration, how many times do we have to iterate over all states in order to converge? Please explain.

Solution If we iterate from $x = M$ to $x = 1$, then the value iteration will converage after the first iteration. This is because the $V_{opt}(x, k)$ only depends on $V_{opt}(x + 1, k)$ and $V_{opt}(x + 1, k - 1)$. In other words, the dependency graph is acyclic.

d. Value Iteration (*16 points*)
Consider the 1×8 grid like below.

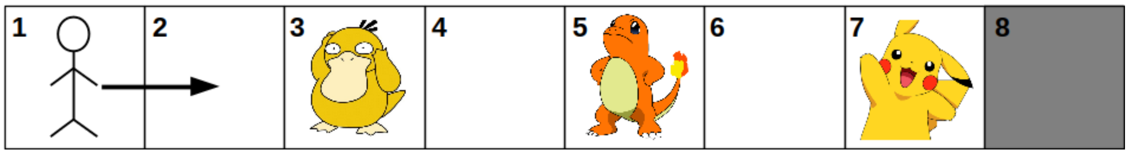


Figure 2: Pokémons on the grid

Let's define $p(3) = p_3, p(5) = p_5, p(7) = p_7$ and $R(3) = r_3, R(5) = r_5, R(7) = r_7$.
Jaebum starts his journey with just one Poké ball. Please answer the following questions.

(i) [10 points] We would like to make our optimal policy to specify *"Try to capture the Pokémon at $x = 3$"*. When will this be true? Find an **if and only if condition** in terms of $p_3, p_5, p_7, r_3, r_5, r_7$ and prove that your condition is correct to get full credit. (*Hint : Your solution might not use all of these variables.*)

Solution

$$\begin{aligned}
V(2,1) &= \max(p_3 V(3,1) + (1-p_3)(r_3 + V(3,0)), V(3,1)) \\
V(3,1) &= V(4,1) \\
V(4,1) &= \max(p_5 V(5,1) + (1-p_5)(r_5 + V(5,0)), V(5,1)) \\
V(5,1) &= V(6,1) \\
V(6,1) &= \max(p_7 V(7,1) + (1-p_7)(r_7 + V(7,0)), V(7,1)) \\
V(7,1) &= V(8,1) = 0
\end{aligned}$$

Also, we know that $V(n,0) = 0$ for any n. Thus, we get

$$\begin{aligned}
V(6,1) &= (1-p_7)r_7 \\
V(4,1) &= \max(p_5(1-p_7)r_7 + (1-p_5)r_5, (1-p_7)r_7)
\end{aligned}$$

To make capturing Pokémon at $x = 3$ as an optimal policy, then

$$p_3 V(3,1) + (1-p_3)(r_3 + V(3,0)) > V(3,1)$$

must be satisfied. Hence, because $V(3,0) = 0$, the above becomes

$$(1-p_3)r_3 > (1-p_3)V(3,1) \rightarrow r_3 > V(3,1)$$

Now, we know that $V(3,1) = V(4,1) = \max(p_5(1-p_7)r_7 + (1-p_5)r_5, (1-p_7)r_7)$. Thus, the condition is

$$r_3 > p_5(1-p_7)r_7 + (1-p_5)r_5 \text{ and } r_3 > (1-p_7)r_7$$

(ii) [2 points] Suppose $p_5 = p_7 = 0.5$ and $r_7 = 8, r_5 = 2$. Assuming that we still have a Poké ball, will the optimal policy tell us to try to capture the Pokémon at $x = 5$ or not?

Solution $V(4,1) = \max(0.5 \times 0.5 \times 8 + 0.5 \times 2, 0.5 \times 8) = \max(3, 4)$ Thus, the best policy is not try capturing it.

(iii) [4 points] Jaebum has written the code to run the value iteration. Now, he just realized that the Pokémons can run away with some probability before he gets there. The probability that the Pokémon at x disappears before reaching is $f(x)$. Note that this event is independent of any other events.

Jaebum wants to reuse his existing value iteration code. How would you set new $p'(x)$ and $R'(x)$ in terms of $p(x)$ and $R(x)$ to incorporate this? **Note that Jaebum starts with 1 Poké ball.**

Solution We can interpret *disappearing* as an *unsuccessful capture*. That means, for Jaebum to successfully capture the Pokémon at x, then first Pokémon should not disappear before getting there and second the Pokémon should not break out from Poké ball. Hence, the probability of capturing a Pokémon decreases to $(1 - f(x))(1 - p(x))$ Let's define the new problem's p as $p'(x)$. Then, we get

$$1 - p'(x) = (1 - f(x))(1 - p(x)) \rightarrow p'(x) = p(x) + f(x) - f(x)p(x)$$

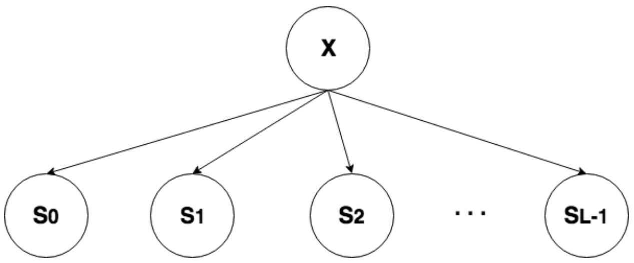
Note that because $0 \le p(x), f(x) \le 1$, $0 \le p'(x) \le 1$ is always satisfied. Hence $p'(x)$ is a valid probability. Also, we don't need to modify $r(x)$.

3. DNA Sequence Analysis (*70 points*) You've been hired by Bayenomics, a new genomics startup applying AI techniques to gene sequencing problems. Your team is investigating a rare genetic disease called *Markosis* and hopes that a statistical analysis of various DNA fragments will yield new insights into the disease.

We define a DNA fragment of length L to be a sequence of L characters, each in the set $\{$A, C, G, T$\}$. For instance, "ACCGAT" is a DNA fragment of length 6, and "GGCT" and "TGGAGACG" are DNA fragments of lengths 4, and 8 respectively.

a. Unigram Models and Bayes' Rule (*20 points*)
Biologists at Bayenomics have collected N DNA fragments, each of length L, some *regular*, and some associated with the disease Markosis, which we call *defective*. They want to develop a Naive Bayes model for predicting whether a given fragment is regular or defective. In the Naive Bayes' model, each character in a fragment is generated independently at random from a probability distribution that depends only on whether the fragment is regular or defective.

Let p_a, p_c, p_g, p_t, represent the unigram probabilities of A, C, G, and T respectively in a regular fragment, and similarly let q_a, q_c, q_g, q_t represent the unigram probabilities for defective fragments. The graphical model representing our Naive Bayes model is shown below.



As an example, if $L = 4$, if we selected a regular fragment at random, the likelihood of the fragment being "AGTA" would be P["AGTA" | regular] $= p_a p_g p_t p_a = p_a^2 p_g p_t$. Similarly, we have P["AGTA" | defective] $= q_a q_g q_t q_a = q_a^2 q_g q_t$.

Let X be the random variable denoting whether a fragment is regular or defective, which we will denote by $X =$ R or $X =$ D respectively. Suppose a randomly drawn fragment from our database has P[X = R] $= z_r$, i.e. the *prior* probability of a fragment being regular is z_r. Given a randomly sampled fragment S, we'd like to be able to predict whether it is regular or defective.

(i) Suppose we pick a fragment S at random from our database, and suppose S contains l_a A's, l_c C's, l_g G's, and l_t T's.
Write down an expression for the probability that this fragment is defective. You should express your answer in terms of $l_a, l_c, l_g, l_t, p_a, p_c, p_g, p_t, q_a, q_c, q_g, q_t$, and z_r. No justification is necessary. [4 points]

Solution By Bayes' rule, we have: $P[D|S] = P[S|D]\ P[D]/P[S] = \frac{P[S|D]P[D]}{P[S|D]P[D]+P[S|R]P[R]}$

$$= \frac{(1-z_r)q_a^{l_a} q_c^{l_c} q_g^{l_g} q_t^{l_t}}{z_r p_a^{l_a} pc^{l_c} p_g^{l_g} pt^{l_t} + (1-z_r)q_a^{l_a} q_c^{l_c} q_g^{l_g} q_t^{l_t}}$$

We call a fragment S *likely-regular* if $P[R|S] > P[D|S]$, and *likely-defective* if $P[D|S] > P[R|S]$.

(ii) Suppose L $=$ 3, and suppose our probabilities are $p_a = 0.4, p_c = 0.3, p_g = 0.2, p_t = 0.1, q_a = 0.1, q_c = 0.2, q_g = 0.3, q_t = 0.4,$ and $z_r = 0.9$.

List all fragments S of length 3 that are *likely-defective*. What fraction of the set of all possible fragments of length 3 are *likely-defective*? No proof is necessary. [7 points]

Hint: Note that $P[D \mid S] > P[R \mid S]$ if and only if $\frac{P[D \mid S]}{P[R \mid S]} > 1$. What character combinations make this happen?

Solution Applying Bayes' rule we see that we need:
$\frac{P[D \mid S]}{P[R \mid S]} > 1. \iff \frac{P[S \mid D]P[D]}{P[S \mid R]P[R]} > 1. \iff \frac{P[S \mid D]}{P[S \mid R]} > 9$

$\iff \frac{q(S_1)q(S_2)q(S_3)}{p(S_1)p(S_2)p(S_3)} > 9.$

Consider how each character affects the ratio on the left hand side: T multiplies it by 4, G multiplies it by 3/2, C multiplies it by 2/3 and A multiplies it by 1/4.

If we have only 1 T, the left hand side would be at most $4/1 \ * 3/2 \ * 3/2 = 9$, but we need it strictly greater than 9. So we need at least 2 T's. 2 T's and 1 A also won't work as that makes the left hand side 4. Thus the only possible character combinations are 3 T's; 2 T's and 1 C; and 2 T's and 1 G. This yields the fragments {"TTT", "TTC", "TCT", "CTT", "TTG", "TGT", "GTT"}, which is 7 out of 64 total fragments of length 3.

In part (ii), we see that for the given probabilities, the fraction of fragments S that are *likely-defective* is quite small, and this is primarily due to the fact that the prior z_r is quite large. You might be tempted to conclude that whenever z_r is large, there are more *likely-regular* fragments than *likely-defective* fragments. Surprisingly, this is not true!

(iii) Prove the following statement: for any value of $z_r < 1$ and any length $L > 0$, there exists a nonzero setting of the probabilities $p_a, p_c, p_g, p_t, q_a, q_c, q_g, q_t$, such that there are more *likely-defective* fragments of length L than *likely-regular* fragments of length L. [9 points]
Hint: As in part (ii), consider the ratio $\frac{P[D \mid S]}{P[R \mid S]}$.

Solution As in the part (ii) we start by looking at the ratio $\frac{P[D \mid S]}{P[R \mid S]} = \frac{(1-z_r)}{z_r} \prod_{i=1}^{L} \frac{q(S_i)}{p(S_i)}$.

Intuition: to make this expression greater than 1, we need to counteract the effect of a large prior z_r by making the product $\prod_{i=1}^{L} \frac{q(S_i)}{p(S_i)}$ large: specifically, we need it to be larger than $\frac{z_r}{1-z_r}$. We can't make *all* the ratios $\frac{q_a}{p_a}, \frac{q_c}{p_c}, \frac{q_g}{p_g},$ and $\frac{q_t}{p_t}$ large, since the q's and p's both sum to 1, as they are probability distributions. We do the next best thing and make 3 of these ratios large: let $q_a = q_c = q_g = q_t = 1/4$, and let $p_a = p_c = p_g = \epsilon, p_t = 1 - 3\epsilon$, for some small ϵ.

This makes the numerator of the fraction, namely $\prod_{i=1}^{L} q(S_i)$, always $(1/4)^L$. Now observe that for **any** fragment S containing a character in {A, C, G}, the denominator $\prod_{i=1}^{L} p(S_i)$ is *at most* ϵ. So for any fragment S other than "TTT...T" we have:

$\prod_{i=1}^{L} \frac{q(S_i)}{p(S_i)} = \frac{(1/4)^L}{\prod_{i=1}^{L} p(S_i)} \geq \frac{(1/4)^L}{\epsilon}$.

We can now make this last fraction arbitrarily large by making ϵ arbitrarily small. In particular, selecting $\epsilon < \frac{1-z_r}{4^L z_r}$ makes this fraction greater than $\frac{z_r}{1-z_r}$. This holds true for **all** fragments other than the all-T's fragment, meaning there is just 1 *likely regular* fragment and $4^L - 1$ *likely defective* fragments.

Notes on part (a):
Most students had no trouble with part (i). Part (ii) is an interesting case study in the effectiveness of pruning, a recurring theme in several parts of the course (e.g. alpha-beta pruning in game trees, AC3 in CSPs, etc.) There's a "brute force" way to solve part (ii) by simply testing all 64 fragments, but this is prohibitively time consuming. Instead, applying Bayes' rule, thinking about the resulting fraction (called the likelihood ratio) and noting that it is invariant to character permutations allows you to very rapidly prune the search space to just a few character combinations.

Many students understood the core ideas behind (ii), with roughly 20% finding all 7 correct fragments and several students finding at least a few correct fragments. Interestingly, many students successfully found {TTT, TTG, TGT, GTT} but missed TTC and its permutations. A surprisingly large fraction of students did not get the total number of fragments of length 3 correct.

Part (iii) builds on the ideas of part (ii) in that it's all about understanding the interaction between prior probability and the likelihood ratio. While many students understood the basic idea that we need to somehow control the ratios $\frac{q_a}{p_a}, \frac{q_c}{p_c}, \frac{q_g}{p_g}$, and $\frac{q_t}{p_t}$, very few students got this perfectly correct. A common error was not realizing that not all of these ratios can be large since the q's and p's must sum to the same value.

b. Markov Models and Recursion (*19 points*)
Scientists at Bayenomics realize that a more granular model of regularity versus defectiveness would better model the biological reality. Rather than classifying entire fragments as regular or defective, each *position* within a fragment can either be regular or defective. You propose a new *character-level* fragment model as follows. Associated with any fragment S of length L is a *hidden array* H, also of length L. $H[i]$ indicates whether the ith position of S is regular or defective.

Formally, H is generated by a Markov Chain defined by the following rules:

1. $H[0] = \text{``}R\text{''}$
2. $H[i] \in \{\text{``}R\text{''}, \text{``}D\text{''}\}\ \forall\, 1 \le i \le L-1$,
3. $\forall i \ge 1, P(H[i+1] = \text{``}R\text{''} | H[i] = \text{``}R\text{''}) = x$, and $P(H[i+1] = \text{``}D\text{''} | H[i] = \text{``}D\text{''}) = y$, where x and y are probabilities in (0, 1).

Basically, H is an array with "R"s and "D"s, the first character is always "R", and subsequent characters are generated stochastically based on the previous character.

(i) Let H be a randomly generated hidden array of length L as described above. For an index k, where $0 \le k \le L-1$, let e_k denote the probability that $H[k]$ is defective. That is, $e_k = P[H[k] = \text{``}D\text{''}]$. Note that $e_0 = 0$ since H[0] is always "R".

Define a recurrence that expresses e_{k+1} in terms of e_k, x, and y. No proof is necessary. [4 points]

Solution Let E_k be the event that $H[k] = \text{``}D\text{''}$. Then:
$P[E_{k+1}] = P[E_k]P[E_{k+1}|E_k] + P[\overline{E_k}]P[E_{k+1}|\overline{E_k}]$
$= e_k y + (1-e_k)(1-x)$
$= (x+y-1)e_k + (1-x)$.

The recurrence of part (i) allows us to compute e_k in $O(k)$ time, but you realize we can do even better.

(ii) Using your result from part (i), find an explicit formula for e_k in terms of x, y, and k. No proof is necessary. [5 points]

Solution If $(x+y-1) = 0$ we have $e_k = 1-x$ for all $k > 0$ and our answer is simply $1-x$. Otherwise, denote $(x+y-1) = a, (1-x) = b$. Then $e_{k+1} = ae_k + b$. Since $e_0 = 1$, we compute $e_1 = b, e_2 = ab + b = b(a+1), e_3 = a^2b + ab + b = b(a^2+a+1)$. Following this pattern, more generally we have $e_k = (a^{k-1} + ... + 1)b = \frac{(1-a^k)b}{1-a}$. Plugging in our values for a and b, the final answer is $\frac{(1-(x+y-1)^k)(1-x)}{2-x-y}$.

Let's call an array H with no consecutive "D"s *isolated*, and *non-isolated* otherwise. For example, the array ["D", "R", "R", "D"] is an *isolated* array of length 4, and so is the array ["R", "R", "D", "R"], since neither of them contain consecutive "D"'s. On the other hand, the arrays ["R", "R", "D", "D"] and ["D", "D", "R", "D"] are non-isolated since they both contain consecutive "D"s.

(iii) What is the probability that an array H of length L, randomly generated by the stochastic process defined above, is isolated? Implement the function below that returns the answer to this question. Include a brief explanation of how your code works. As a reference, our solution is 6 additional lines of code, but it's very compact so don't worry if you deviate significantly from this. [10 points]

Hint: Let $f(n, D)$ and $f(n, R)$ denote the probability that an array of length n is isolated and ends with "D" or "R" respectively. Find recurrences for $f(n, D)$ and $f(n, R)$.

```
def isolatedProb(L, x, y):
    """
    Args:
        L: length of the array being generated.
        x and y: transition probabilities defined earlier

    Returns:
        Probability that an array of length L generated by above Markov chain is iso
    """
    # Your code starts here
```

```
        # Your code ends here
```

Briefly explain your code here:

Solution Let $f(n, D)$ denote the probability that an array of length n is isolated and ends with “D”, and define $f(n, R)$ analogously. The key observation is that if an isolated array ends with “D”, then its second last character must be “R”. On the other hand, if the last character is “R”, the second last character could be either “R” or “D”. This yields the following recurrence relations:
(i) $f(n, D) = f(n-1, R) \cdot (1-x)$
(ii) $f(n, R) = f(n-1, R) \cdot x + f(n-1, D) \cdot (1-y)$
Finally, the result we seek is just $f(L, D) + f(L, R)$.

This leads to a very short implementation as follows:

```
        def isolatedProb(L, x, y):
            """
            Args:
                L: length of the array being generated.
                x and y: transition probabilities defined earlier
            Returns:
                Probability that array of length L
                generated by above Markov Chain is isolated.
            """
            # Recursive wrapper
            def wrapper(n, prevChar):
                if n == 1:
                    return 1. if prevChar == "R" else 0.
                return wrapper(n-1, "R") * (1-x) if prevChar == "D" else wrapper(
                n-1, "R") * x + wrapper(L-1, "D") * (1-y)

            return 1. if L <= 1 else wrapper(L, "R") + wrapper(L, "D")
```

Notes on part (b)

Recurrence relations are a tool we've seen repeatedly in this course, including in search algorithms, MDPs, game tree algorithms, CSPs, and inference in Bayesian networks. All three parts of this question illustrate the incredible power and versatility of recursion. Part (i) demonstrates how recursion allows us to compute e_k in linear time, rather than the exponential time complexity of listing all paths of length k and adding their individual probabilities. Part (ii) shows that amazingly, we can actually do this in *sublinear* time via the closed form solution, as raising something to the kth power can be done in $O(log k)$ time (something to think about: why is this true?)

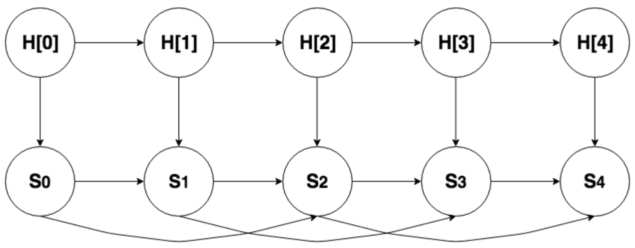
Part (iii) shows how we can respond to a complicated probabilistic query about a Hidden Markov Model in just 6 lines of Python code. A dynamic programming solution by memoizing (i.e. caching) intermediate results and computing the wrapper values in a bottom-up manner is also possible, and may be slightly more time efficient (something else to think about: what is the time complexity of our provided algorithm?).

c. n-grams and the EM algorithm (*15 points*)
Let's now modify our Naive Bayes model from part (a) in two ways:

1. We incorporate the hidden array H defined in part (b)
2. We use n-grams of window size k instead of unigrams

Now, the probability of a character ch occurring at position i in a fragment S is given by the distribution:
$P(S_i = ch \mid H[i], S_{i-1}, S_{i-2}, ..., S_{i-k})$.
Here, as in the homework, we introduce a special token BEGIN and define S_j to be BEGIN for $j < 0$. Thus the distribution of the character S_j depends on **both** the corresponding character in the hidden array $H[j]$ **and** the previous k characters in S.

Below is a graphical model illustrating our new n-gram model for $L = 5, k = 2$.



Observe that for this Bayesian network, the probability of a full assignment of characters to S and H is:
$\prod_{i=1}^{L-1} P[H[i] \mid H[i-1]] \prod_{i=0}^{L-1} P[S_i \mid H[i], S_{i-1}, ..., S_{i-k}]$.

The transition probabilities for H are known, but the emission probabilities
$P[S_i = ch \mid H[i], S_{i-1}, S_{i-2}, ..., S_{i-k}]$ are unknown. We therefore turn to the EM algorithm. We've seen how to do EM in HMMs, using the forward-backward algorithm for the E-step. We can't do exactly the same thing here since our n-gram model is not quite an HMM, due to the edges between visible nodes. However, we can come up with a **modified forward-backward algorithm** for the E-step.

(i) Provide equations for the **E-step** of the EM algorithm for this model for inferring a distribution over hidden states by filling in the three blanks below. No justification is necessary. [7 points]

// Forward Pass
for $i = 1$ to $L - 1$:

$F_i(H[i]) =$ ____

```
// Backward Pass
for $i = L - 1$ downto 1:

$B_i(H[i]) =$ ______________________________

// Product - this step is done for you
for $i = 1$ to $L - 1$:
$M_i(H[i]) = F_i(H[i])B_i(H[i])$

// Get final probabilities
for i = L-1 downto 1:

$P[H[i] = \text{“}R\text{”}] =$ ________________________

$P[H[i] = \text{“}D\text{”}] = 1 - P[H[i] = \text{“}R\text{”}]$
```

Solution The algorithm is almost identical to the forward-backward algorithm, with the standard emission probabilities replaced by the emission probabilities in our model.

```
for i = 1 to L-1:
$F_i(H[i]) = \sum_{H[i-1]} F_{i-1}(H[i-1])\ P[H[i] \mid H[i-1]]\ P(S_i \mid H[i], S_{i-1}, S_{i-2}, ..., S_{i-k})$

for i = L-1 downto 1:
$B_i(H[i]) = \sum_{H[i+1]} B_{i+1}(H[i+1])\ P[H[i+1] \mid H[i]]\ P(S_{i+1} \mid H[i], S_i, S_{i-1}, ..., S_{i-k+1})$

for i = 0 to L-1:
$M_i(H[i]) = F_i(H[i])B_i(H[i])$

for i = L-1 downto 0:
$P[H[i] = \text{“}R\text{”}] = \frac{M_i(\text{“}R\text{”})}{M_i(\text{“}R\text{”})+M_i(\text{“}D\text{”})}$

$P[H[i] = \text{“}D\text{”}] = 1 - P[H[i] = \text{“}R\text{”}]$
```

(ii) Suppose that $L = 1000$. How many parameters are updated in the **M-step** if Laplace smoothing is used? Express your answer in terms of k (the n-gram window size). For what values of k are we guaranteed to update **more** parameters if Laplace smoothing is used than we would without Laplace smoothing? Briefly justify your answers. [8 points]

Solution The main thing to note is that with Laplace smoothing, **every** emission probability parameter is written to in the M-step, so our first task is to count the number of emission probability parameters in our model.

The emission probabilities we're estimating are of the form:
$P[S_i \mid H[i], S_{i-1}, ..., S_{i-k}]$.

Here, S_i can take 4 possible values, and each of $S_{i-1}, ..., S_{i-k}$ can take one of 5 values, namely BEGIN, A, C, G, T and H[i] can take one of 2 values ("R" or "D"). Thus the total number of parameters is $4 * 5^k * 2 = 8 * 5^k$.

Without Laplace smoothing, we only update emission probabilities corresponding to observed emissions, so we update at most $2L = 2000$ parameters (the factor of 2 comes from the fact that hidden nodes can have fractional counts for both "R" and "D" due to the E-step).

When $8 * 5^k > 2L$, we are guaranteed to update more parameters with Laplace smoothing than without. This happens when $k > log_5(2L/8) = log_5(500)$, which is between 3 and 4, so the answer is for all $k \geq 4$.

Notes on part (c)

The main point of this question was to test your understanding of what EM and forward-backward do. What do the forward and backward messages in forward-backward represent? How do you find the number of parameters in a Bayesian network? Which of these does the M-step update? These are all important details to understand about Bayesian networks and inference algorithms on them, and if you really understand these, this question is much more straightforward than it initially seems!

The E-step is a direct extension of the forward-backward algorithm. There's nothing fancy going on here: no recursion, dynamic programming, conditioning, or elimination algorithm. Just tweak the forward and backward messages to use our model's emission probabilities instead of the normal HMM emission probabilities.

There are admittedly a few tricky details to catch in the M-step question, so we've been generous with partial credit for solutions that hit the main ideas but don't catch the more subtle details (e.g. not handling BEGIN tokens, missing the factor of 2 due to the E step, etc.)

An interesting side note: the solution to the M-step problem shows that for large n-gram windows, specifically when $k >> log_5 L$, the time complexity of Laplace smoothing will be **much** higher than without using Laplace smoothing! This is due to a combinatorial explosion in the number of parameters. On the other hand, not using Laplace smoothing would lead to overfitting. This renders n-gram models with large window sizes impractical. To address this, there has been a lot of research on neural network based models for sequences such as LSTMs, which can effectively simulate having a large window size by learning to selectively "remember" past information.

d. Constraint Satisfaction meets HMMs (*16 points*)
The EM algorithm allows us to infer a posterior distribution over the hidden variables H. What we'd often like to do though in applications is to find the maximum likelihood setting of the hidden variables. This would in essence be our "best guess" of which parts of the DNA fragment are defective.

In this problem, we'll consider a constrained version of maximum likelihood. Specifically, we'll consider the task of finding an array H with maximum possible likelihood subject to the constraint that **no three consecutive characters of H are all "D".**

(i) We wish to model this constraint on H using factors. How many factors do we need to add? [1 point]

Solution We add one factor for each triple of consecutive characters in H, which yields $max\{0, L-2\}$ factors.

(ii) Suppose we run Gibbs sampling. In step i of the inner loop, we need to set our current guess of $H[i]$ to "R" with some probability, call it $p_{Gibbs}(R, i)$, and we set it to "D" with probability $1 - p_{Gibbs}(R, i)$.

Write down an expression for $p_{Gibbs}(R, i)$ in terms of the probabilities used to define our graphical model. Assume that setting $H[i]$ to "D" does not violate any constraint. You may also assume that $0 < i < L - 1$, that is, **you do not have to worry about edge cases where i is at one end of the array**. [7 points]

Solution The Gibbs sampling probability depends only on local factors and is proportional to the product of their weights. Hence for a nonterminal hidden node, we have:
$p_{Gibbs}(H, i) = \frac{g(\text{"}R\text{"})}{g(\text{"}R\text{"})+g(\text{"}D\text{"})}$, where

$$g(ch) = P[H[i] = ch \mid H[i-1]]\ P[H[i+1] \mid H[i] = ch]\ P[S_i \mid H[i] = ch,\ S_{i-1}, ..., S_{i-n}].$$

(iii) We know that a danger of running Iterated Conditional Modes is getting stuck in local optima. Suppose for this part of the question that all emission probabilities are equal to 1/4 (that is, each character is drawn from {A, C, G, T} uniformly at random regardless of the hidden state and previous k characters. Further suppose that $x = y > 0.5$. Provide 2 examples of arrays H with length 7 that are local optima but not global optima. A local optimum here is defined as an array H such that it is impossible to increase the weight of H by changing any one of its elements. No justification is necessary.
Note: Remember that $H[0]$ is always "R". [8 points]

Solution All emission probabilities are the same, so we only need to pay attention to the Markov Chain. Since the R–>R and D–>D transition probabilities are equal to each other and greater than the R–>D and D–>R transition probabilities, the only quantity that really matters here is the number of transitions in which the character stays the same. Thus the only global optimum is [R, R, R, R, R, R, R], since this is the only sequence in which the character stays the same in all transitions (remember that $H[0] = R$).

One local optimum is [R, D, D, D, D, D, D]. The weight of this assignment is 0 since there are more than 3 D's in a row. No matter what character is flipped, we'll still have 3 D's in a row, and the weight of our assignment will still be 0.

Another local optimum is [R, D, D, R, D, D, R]. None of the R's can legally flip to D's, and flipping any D to R keeps the weight the same.

Notes on part (d)
Part (ii) is fairly mechanical if you understand Gibbs sampling: just take the product of local factors and normalize. Part (iii) on the other hand requires a little creativity to construct examples of local optima.

Several algorithms we've seen in this class are inexact in that they do not guarantee an optimal solution, and many are susceptible to local optima. Examples of inexact algorithms include k-means, minimax with heuristic evaluation functions, beam search, particle filters, Gibbs sampling, iterative conditional modes, and neural networks. Getting some intuition for the set of local optima for an algorithm is a valuable skill in both engineering and research, since this can help determine the suitability of the algorithm for your task, and also possibly allow you to modify the algorithm to escape from local optima. In general, characterizing local optima can be an extremely difficult task: doing this for neural networks is an active area of research, and research on local optima for k-means led to the development of several improved k-means algorithms (such as kmeans++). Part (iii) gives you a chance to explore local optima for ICM on a very small test example.

For those who are curious, below are some other valid examples of local optima.
Any sequence with 2 consecutive D's:

R, D, D, R, R, R, R

R, R, D, D, R, R, R

R, R, R, D, D, R, R

R, R, R, R, D, D, R

R, R, R, R, R, D, D

Some sequences with pairs of separated consecutive D's:
R, D, D, R, R, D, D

R, R, D, D, R, D, D

{left blank for scratch work}

{left blank for scratch work}