Homework 5

Winston (Hanting) Zhang

Friday, October 28, by 23:00

General Instructions. The following assignment is meant to be challenging, and we anticipate that it will take most of you at least 10–15 hours to complete, so please allow yourself plenty of time to work on it. In fact, this assignment is probably the most difficult assignment of the semester so far. We highly recommend reading the entire assignment right away — you never know when inspiration will strike. Please provide a formal mathematical proof for all your claims, and present runtime guarantees for your algorithms using asymptotic (big-$O/\Omega/\Theta$) notation, unless stated otherwise. You may assume that all basic arithmetic operations (multiplication, subtraction, division, comparison, etc.) take constant time.

Collaboration. Please carefully check the collaboration policy on the course website. When in doubt, ask an instructor.

Consulting outside sources. Please carefully check the policy regarding outside sources on the course website. Again, when in doubt, ask an instructor.

Submission. Homework submission will be through the Gradescope system. Instructions and links have been provided through the course website and Piazza. The only accepted format is PDF. Starting with this homework, only typed solutions will be accepted, and we highly recommend producing your solutions using LaTeX (the text markup language we are also using for this assignment).

Recommended practice problems (do not hand in): KT Problems 6.2, 6.4, 6.6, 6.8, 6.11, 6.13, 6.16, 6.20, 6.24

Both problems in this homework are using quite a bit of probability. You may want to review those concepts from CSCI 170, and perhaps even a bit beyond.

Problem 1. [15]

Climate change has been affecting California heavily — as you can see almost daily, there are more and more severe wildfires and other consequences from year to year. The severity of wildfires is closely related with the amount of rain (or lack thereof) that has occurred in the preceding year, so scientists are trying to calculate the probability that 2023 will be an even dryer year than 2022.

Specifically, they posit the following model. There are n days that they are interested in, and for each day i, their meteorological model gives them a probability $p_i \in [0, 1]$ that it rains on day i. For simplicity, they assume that rain on different days happens independently.[1] They are also given the number k of drought days in 2022. Now, they want to calculate the probability that there will be *more* than k drought days in 2023.

Give (and analyze, of course) an algorithm with running time $O(n^2)$ which correctly computes the probability that more than k days will be drought days in 2023. You may assume that arithmetic operations take time $O(1)$, even when they involve numbers that are the products of up to n probabilities.[2] You should give a full algorithm and proof here.

Algorithm 1 Probability that more than k out of n days rain.

let $n \leftarrow p$.length
let $dp \leftarrow (n+1) \times (n+1)$ array of zeros
for $i = 0, \ldots, n$ **do**
  let $dp[i][0] \leftarrow \prod_{k=0}^{i} p_i$
for $j = 1, \ldots, n$ **do**
  let $dp[0][j] \leftarrow 0$
for $i = 1, \ldots, n$ **do**
  for $j = 1, \ldots, n$ **do**
    let $dp[i][j] \leftarrow (1 - p_i) \cdot dp[i-1][j-1] + p_i \cdot dp[i-1][j]$
return $\sum_{j=k+1}^{n} dp[n][j]$

Proposition 1. *Algorithm 1 computes the probability that more than k drought days out of n.*

Proof. **Correctness:** We define $dp[i][j]$ to be the probability that there will be *exactly* i out of j drought days. Then the problem can be reformulated into two steps, first computing dp, and then computing $\sum_{j=k+1}^{n} dp[n][j]$.

We think in terms of subproblems by doing casework on the result of the i^{th} day. If the i^{th} day rains, then the probability of having exactly j drought days is the probability that we had exactly j drought days in the previous $i-1$ days, i.e. $dp[i-1][j]$. Otherwise, if the i^{th} day is a drought day, then we want the probability that we had exactly $j-1$ drought days in the previous $i-1$ days, i.e. $dp[i-1][j-1]$. The first case happens with probability p_i and the second case happens with probability $1 - p_i$, thus we have the recurrence

$$dp[i][j] = (1 - p_i) \cdot dp[i-1][j-1] + p_i \cdot dp[i-1][j].$$

The base cases are when $i = 0$ or $j = 0$.

1. When $i = 0$, we are computing the probability that there are j drought days out of 0 days. This happens with probability 1 when $j = 0$ and probability 0 when $j > 0$. This is reflected in the second for loop.

2. When $j = 0$, we are computing the probability that there are 0 drought days out of j days. This means every day rains, which occurs with probability $\prod_{k=0}^{i} p_k$. This is reflected in the first for loop.

[1] Check your probability notes if you don't remember what that term means and implies.

[2] In reality, such numbers would likely need n decimal digits, and hence require time $\Theta(n)$ for arithmetic operations. You get to ignore this subtlety.

Now we proceed by induction on $i + j$ to show that $dp[n][k]$ is the probability that there are exactly k out of n drought days (most of the work has been done already, this is just a formality). Indeed, the base cases are when $i = 0$ or $j = 0$.

1. When $i = 0$, we are computing the probability that there are j drought days out of 0 days. This happens with probability 1 when $j = 0$ and probability 0 when $j > 0$. This is reflected in the second for loop.

2. When $j = 0$, we are computing the probability that there are 0 drought days out of j days. This means every day rains, which occurs with probability $\prod_{k=0}^{i} p_k$. This is reflected in the first for loop.

Assume for the sake of induction that for any $i' + j' < i + j$, $dp[i'][j']$ has the correct probability. Then by our analysis of the recurrence, we know that

$$dp[i][j] = (1 - p_i) \cdot dp[i-1][j-1] + p_i \cdot dp[i-1][j].$$

Since $(i-1) + (j-1) < i + j$ and $i - 1 + j < i + j$, it follows that $dp[i-1][j-1]$ and $dp[i-1][j]$ are correct. Thus $dp[i][j]$ must also be correct. This proves the induction.

In the end, recall that we are actually trying to compute the probability that more than k days are droughts. This is just the sum of the probabilities that exactly j out of n days are droughts, for $k < j \leq n$. Thus we return $\sum_{j=k+1}^{n} dp[n][j]$, and we've obtained our desired probability and Algorithm 1 is correct.

Termination: Algorithm 1 is not recursive and is just a bunch of loops, so after a finite amount of iterations it must terminate.

Runtime: Refer to the code above. The first loop has $n + 1$ iterations, and we can assume that the product of probabilities inside the loop takes $O(1)$ to compute, so the loop runs in $O(n)$. The second loop has $n + 1$ iterations and a $O(1)$ body again, so $O(n)$. The main double loop iterates n^2 times and the body is again $O(1)$, so this runs in $O(n^2)$. The final sum takes at worst $O(n)$ to compute. Thus the total runtime is $O(n^2) + 3O(n) = O(n^2)$. □

Problem 2. [15]

Consider the problem of learning what is going on inside a system (black box) when you cannot observe the inside, and only see some kind of output that gives you partial information about the inside. We model this as follows.

There is a known directed graph $G = (V, E)$, with a known start node $s \in V$. Associated with each node $v \in V$ is a probability distribution q_v over letters of the alphabet[3] a–z, and a probability distribution p_v over edges $E_v \subseteq E$ leaving v. So $q_{v,x} \geq 0$ is the probability that the letter $x \in \{a, \ldots, z\}$ is produced at node v. Because it is a probability distribution, we know that $\sum_{x \in \{a,\ldots,z\}} q_{v,x} = 1$ for all v. Similarly, $p_{v,e} \geq 0$ is the probability of taking edge e out of v, for each $e \in E_v$. Here, we know that $\sum_{e \in E_v} p_{v,e} = 1$ for all nodes v.

The way such a system produces an output is now as follows: the system starts in $v_1 = s$. Then, for each time step $t = 1, 2, \ldots$, assume that the system is at node v_t. It randomly picks a letter x_t to output according to the distribution q_{v_t}. Then, it randomly picks an edge $e = (v_t, u)$ out of v_t to follow, according to the distribution p_{v_t}. The endpoint u of the edge e it picked becomes the new node $v_{t+1} = u$. This repeats for some number $T \geq 0$ of steps. As a result, it produces some output string x as the sequence of letters x_t that are output.

The computational question we are facing is now the following: we know G and s (and all the probabilities), but we can't see the sequence $\langle v_1, v_2, \ldots \rangle$ of vertices that the system is at. All we can observe is the output, i.e., the sequence of letters $x = x_1 x_2 \cdots x_T$ (which is also part of our input). There may be many different sequences $\langle v_1, v_2, \ldots, v_T \rangle$ of vertices which could have produced this same sequence x of letters. Among all of them, the goal is to output one with largest likelihood. The likelihood of a sequence $\langle v_1, v_2, \ldots, v_T \rangle$ is defined as

$$L(\langle v_1, v_2, \ldots, v_T \rangle) = \prod_{t=1}^{T} q_{v_t, x_t} \cdot \prod_{t=1}^{T-1} p_{v_t, (v_t, v_{t+1})}.$$

So it is the product of the probability of outputting the observed character x_t in each of the assumed states v_t, times the probability of taking the edge from v_t to v_{t+1} for each of the time steps $t = 1, \ldots, T - 1$.

[3] We could make them numbers, or anything else.

Give and analyze (running time and correctness) an algorithm for finding the maximum likelihood of any vertex sequence for the given string. You do not need to output the actual sequence. You should give pseudo-code for an actual implementation, but if you prove correctness of a recurrence, you do not need to do another correctness proof for the pseudo-code.

Remark 1. *Due to how the question was phrased, we have chosen to 1-index everything! Take note while reading the following algorithm and proof.*

Algorithm 2 Path of largest likelihood of being taken.

let $n \leftarrow V$.size
let $dp \leftarrow T \times n$ array of zeros
let prev $\leftarrow$ empty map of previous nodes
let $dp[1][1] \leftarrow q_{v_1,x_1}$
for $i = 1, \ldots, T$ **do**
  for $j = 1, \ldots, n$ **do**
    for $k = 1, \ldots, n$ **do**
      let $t_{kj} \leftarrow p_{v_k,(v_k,v_j)} \cdot q_{v_j,x_i} \cdot dp[i-1][k]$
      if $t_{kj} > dp[i][j]$ **then**
        $dp[i][j] = t_{kj}$
        prev[v_j] = v_k
let $(v^*, p^*) \leftarrow$ highest probability node and value amongst the last row of the dp table
let $v_T \ldots v_2 v_1 \leftarrow$ retrace prev map from $v^* = v_T$ to v_1
return $v_1 v_2 \ldots v_T, p^*$

Proposition 2. *Enumerate the vertices $v_1, \ldots, v_n$, setting $s = v_1$. Let OPT(i, j) be the largest probability among all sequences $\langle v_1, \ldots, v_j \rangle$ that output $x_1 \ldots x_i$. (Note that v_j here is fixed!) The recurrence for this problem is given by*

$$OPT(i,j) = \max_{k=1,\ldots,n} (p_{v_k,(v_k,v_j)} \cdot q_{v_j,x_i} \cdot OPT(i-1,k)).$$

Furthermore, the base cases are $OPT(1,1) = q_{v_1,x_1}$ and OPT$(1, j) = 0$ if $j > 1$.

Proof. Consider the subproblems we've identified through our recurrence: the pairs (x_i, v_j) which represent the maximum sequence for the substring $x_1 \ldots x_i$ ending at v_j. At the subproblem (x_i, v_j), we must have come from a subproblem with one less character for its substring, that is to say $x_1 \ldots x_{i-1}$, ending at some v_k. Depending on k, the transition probability that we will traverse the edge (v_k, v_j) and produce character x_i after arriving at v_j is $p_{v_k,(v_k,v_j)} \cdot q_{v_j,x_i}$. Furthermore, we must factor in the probability of arriving at the subproblem (x_{i-1}, v_k) in the first place. By definition, the maximum probability for the subproblem (x_i, v_j) is OPT(i, j) and the smaller subproblems (x_{i-1}, v_k) are OPT$(i-1, k)$. Hence the maximum probability of the subproblem (x_i, v_j) is

$$\text{OPT}(i,j) = \max_{k=1,\ldots,n} \left(p_{v_k,(v_k,v_j)} \cdot q_{v_j,x_i} \cdot OPT(i-1,k)\right).$$

This is exactly what we claim as our recurrence.

For the base cases, we must start at $v_1 = s$, so all subproblems (x_1, v_j) for $j \neq 1$ have probability zero of occuring. The subproblem (x_1, v_1) occurs only if the character produced at v_1 is x_1, which has probability q_{v_1,x_1}. This completes our claim. □

Proposition 3. *Algorithm 2 terminates and runs in $O(Tn^2)$.*

Proof. **Termination:** Again, algorithm 2 is not recursive and is just a bunch of loops, so after a finite amount of iterations it must terminate.

Runtime: Refer to the code above. Initializing dp and prev takes $O(nT)$. The main triple loop iterates Tn^2 times and the body is again $O(1)$, so this runs in $O(Tn^2)$. The final calculation of (v^*, p^*) takes $O(n)$ to compute. Retracing prev takes at most $O(T)$ steps. The highest factor here is $O(Tn^2)$, so the total runtime is just $O(Tn^2)$. □

If this problem looks a little "weird" or "contrived" to you, here is a bit of context. This type of analysis is actually really useful for a number of problems related to machine learning, artificial intelligence, and data mining. As one example, think about a sports team you like, and their sequence of wins and losses. If you see something like LLLWLLLLWLLLLLWWWWWLWWWW, you would probably explain it as "At the beginning of the season, the team was bad, and just got a couple random wins. But then at the end of the season, they got their act together and won all games except a random loss." On the other hand, if you see something like LWLLWLWWLLWLWLLWWWLLLWLWL, you would probably explain it as "It was a mediocre team which went back and forth between winning and losing." Implicitly, you would be using that good teams mostly win games (output 'W' with high probability), bad teams mostly lose games (output 'L' with high probability), and mediocre teams win about half of their games. Here, good/bad/mediocre would be the unknown state. You'd implicitly use that going from good to bad or vice versa (or going to/from mediocre) is possible, but unlikely. So you would not necessarily explain the second example as "My team kept going back and forth between good and bad." You would consider that as an unlikely explanation. In Figure **??**, you see a potential example system for the sports teams. Here, teams stay in the same state with probability 90%, and transition to another state with probability 5% each, and good teams output 'W' with probability 90%, bad teams output 'W' with probability 10%, and mediocre teams output 'W' with probability 50%. If you solved the optimization problem for the two input sequences we gave you, the algorithm would likely discover the example explanations we gave. You can try playing with some simpler sequences by hand, and see which explanations have the highest likelihood.

Another, even more important, kind of example arises in human speech. Here, unknown states are what the speaker has in mind. The output you observe is sounds (phonemes), which are usually quite ambiguous in terms of what it seems the speaker is saying.[4] What your brain does for you automatically and very well (and you would like to have a computer do equally well) is to use all of the context — the sounds you heard earlier and later — to figure out what meaning a sound actually had.[5]

There are a lot of other examples where this type of reasoning for explaining observed sequences of data/events is extremely useful.

Chocolate Problem: 1 chocolate bar

Chocolate Problem: 3 chocolate bars

Problem 6.29 from the textbook. As a warmup, we recommend Problem 4.30 from the textbook (which isn't exactly easy, either).

[4]For example, "science" and "signs" sound quite similar from the average English speaker.

[5]In our earlier example, if the sound you hear was preceded by something that sounded like "computer", it is more likely that you heard "science", but if it sounded like "traffic", it is more likely that you heard "signs".