CS 189/289A Introduction to Machine Learning
Fall 2024 Jennifer Listgarten, Saeed Saremi

DIS9

1 Concerns about Randomness

One may be concerned that the randomness introduced in random forests may cause trouble. For example, some features or sample points may never be considered at all. In this problem we will be exploring this phenomenon.

(a) Consider n training points in a feature space of d dimensions. Consider building a random forest with T binary trees, each having exactly h internal nodes. Let m be the number of features randomly selected (from among d input features) at each tree node. For this setting, compute the probability that a certain feature (say, the first feature) is never considered for splitting in any tree node in the forest.

Solution: The probability that it is not considered for splitting in a particular node of a particular tree is $1 - \frac{m}{d}$. The subsampling of m features at each treenode is independent of all others. There is a total of ht treenodes and hence the final answer is $(1 - \frac{m}{d})^{hT}$.

(b) Now let us investigate the possibility that some sample point might never be selected. Suppose each tree employs $n' = n$ bootstrapped (sampled with replacement) training sample points. Compute the probability that a particular sample point (say, the first sample point) is never considered in any of the trees.

Solution: The probability that it is not considered in one of the trees is $(1 - \frac{1}{n})^n$, which approaches $1/e$ as $n \to \infty$. Since the choice for every tree is independent, the probability that it is not considered in any of the trees is $(1 - \frac{1}{n})^{nT}$, which approaches e^{-T} as $n \to \infty$.

(c) Compute the values of the two probabilities you obtained in parts (b) and (c) for the case where there are $n = 50$ training points with $d = 5$ features each, $T = 25$ trees with $h = 8$ internal nodes each, and we randomly select $m = 1$ potential splitting features in each treenode. You may leave your answer in a fraction and exponentiated form, e.g., $\left(\frac{51}{100}\right)^2$. What conclusions can you draw about the concerns of not considering a feature or sample mentioned at the beginning of the problem?

Solution: $(\frac{4}{5})^{200} \approx 4.15 * 10^{-20}$ and $(\frac{49}{50})^{1250} \approx 1.07 * 10^{-11}$. It is quite unlikely that a feature will be missed, and extremely unlikely a sample will be missed.

2 Probabilistic Graphical Models

Recall that we can represent joint probability distributions with directed acyclic graphs (DAGs). Let G be a DAG with vertices $X_1, ..., X_k$. If P is a (joint) distribution for $X_1, ..., X_k$ with (joint) probability mass function p, we say that G represents P if

$$p(x_1, \cdots, x_k) = \prod_{i=1}^{k} P(X_i = x_i | \text{pa}(X_i)), \tag{1}$$

where $\text{pa}(X_i)$ denotes the parent nodes of X_i. (Recall that in a DAG, node Z is a parent of node X iff there is a directed edge going out of Z into X.)

Consider the following DAG

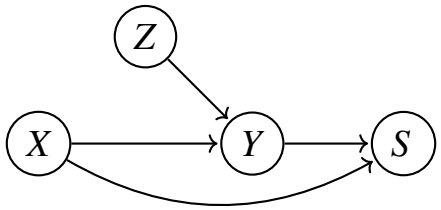

Figure 1: G, a DAG

(a) Write down the joint factorization of $P_{S,X,Y,Z}(s, x, y, z)$ implied by the DAG G shown in Figure 1.

Solution:

$$P_{S,X,Y,Z}(s, x, y, z) = P(X = x)P(Z = z)P(Y = y|X = x, Z = z)P(S = s|X = x, Y = y)\,.$$

(b) Is $S \perp Z \mid Y$?

Solution: No. As a counterexample, consider the case where all nodes represent binary random variables, $P(X = 1) = P(Z = 1) = 0.5$, $Y = X \otimes Z$, and $S = X \otimes Y$, where $\otimes$ is the XOR operator. Then we can see that $S = Z$, whereas knowing Y does not fully determine S or Z.

A version of these solutions from a previous semester erroneously said that this conditional independence did hold. As a result, you may have wrongly heard in section that this statement is true, via faulty algebraic manipulation and/or other algorithms such as the Bayes ball (d-separation). Running these algorithms correctly should show that S and Z are indeed not conditionally independent given Y.

If X is fully removed from G, then we do indeed have $S \perp Z \mid Y$. This is left as an exercise in algebraic manipulation of probability distributions.

(c) Is $S \perp X \mid Y$?

Solution: No. Consider the same example from above with binary random variables. Knowing Y does not determine S, but knowing both X and Y does.

3 PGMs: Sleeping in Class

In this question, you'll be reasoning about a Dynamic Bayesian Network (DBN), a form of a Probabilistic Graphical Model.

Your favorite discussion section TA wants to know if their students are getting enough sleep. Each day, the TA observes the students in their section, noting if they fall asleep in class or have red eyes. The TA makes the following conclusions:

1. The prior probability of getting enough sleep, S, with no observations, is 0.7.
2. The probability of getting enough sleep on night t is 0.8 given that the student got enough sleep the previous night, and 0.3 if not.
3. The probability of having red eyes R is 0.2 if the student got enough sleep, and 0.7 if not.
4. The probability of sleeping in class C is 0.1 if the student got enough sleep, and 0.3 if not.

(a) Formulate this information as a dynamic Bayesian network that the professor could use to filter or predict from a sequence of observations. If you were to reformulate this network as a hidden Markov model instead (that has only a single observation variable), how would you do so? Give a high-level description (probability tables for the HMM formulation are not necessary.)

Solution: Our Bayesian Network has three variables: S_t, whether the student gets enough sleep, R_t, whether they have red eyes in class, and C_t, whether the student sleeps in class. Moreover, S_t is a parent of S_{t+1}, R_t, and C_t.

Let $S_t = 1$ be the event that the student gets enough sleep on day t, and $S_t = 0$ be otherwise. Similarly, let $R_t = 1$ be the event that the student has red eyes on day t, and let $R_t = 0$ be otherwise. Finally, let $C_t = 1$ be the event that the student sleeps in class on day t, and let $C_t = 0$ be otherwise.

The network can be provided pictorally, or fully through conditional probability tables (CPTs.) The CPTs for this problem are given by:

$$\begin{aligned} P(S_1 = 1) &= 0.7 \\ P(S_{t+1} = 1 \mid S_t = 1) &= 0.8 \\ P(S_{t+1} = 1 \mid S_t = 0) &= 0.3 \\ P(R_t = 1 \mid S_t = 1) &= 0.2 \\ P(R_t = 1 \mid S_t = 0) &= 0.7 \\ P(C_t = 1 \mid S_t = 1) &= 0.1 \\ P(C_t = 1 \mid S_t = 0) &= 0.3 \end{aligned}$$

To reformulate this problem as an HMM with a single observation node, we can combine the 2-valued variables R_t and C_t into a single 4-valued variable e_t (e for evidence), multiplying together the emission probabilities.

(b) Consider the following evidence values at timesteps 1, 2, and 3:

(a) $e_1 = (R_1 = 0, C_1 = 0)$ = not red eyes, not sleeping in class

(b) $e_2 = (R_2 = 1, C_2 = 0)$ = red eyes, not sleeping in class

(c) $e_3 = (R_3 = 1, C_3 = 1)$ = red eyes, sleeping in class

Find the likelihood of this sequence of observations. Assume a prior on $P(S_1)$ that is consistent with the prior in the previous part; that is, $P(S_1 = 1) = 0.7$.

Solution: We can apply the α-update algorithm. Define

$$\alpha_t(j) = P(S_t = j, e_{1:t})$$

for $t = 1, 2, 3$. The α-update algorithm tells us that these are related via the following recursion

$$\begin{aligned}
\alpha_t(j) &= P(S_t = j, e_1, \ldots, e_t) \\
&= \sum_i P(S_t = j, S_{t-1} = i, e_1, \ldots, e_{t-1}, e_t) \\
&= \sum_i P(S_{t-1} = i, e_1, \ldots, e_{t-1}) P(S_t = j \mid S_{t-1} = i) P(e_t \mid S_t = j) \\
&= \sum_i \alpha_{t-1}(i) P(S_t = j \mid S_{t-1} = i) P(e_t \mid S_t = j)
\end{aligned}$$

and we use the following base case

$$\begin{aligned}
\alpha_1(1) &= P(S_1 = 1, e_1) \\
&= P(S_1 = 1) P(e_1 \mid S_1 = 1) \\
&= P(S_1 = 1) P(R_1 = 0, C_1 = 0 \mid S_1 = 1) \\
&= P(S_1 = 1) P(R_1 = 0 \mid S_1 = 1) P(C_1 = 0 \mid S_1 = 1) \\
\alpha_1(0) &= P(S_1 = 0, e_1) \\
&= P(S_1 = 0) P(e_1 \mid S_1 = 0) \\
&= P(S_1 = 0) P(R_1 = 0, C_1 = 0 \mid S_1 = 0) \\
&= P(S_1 = 0) P(R_1 = 0 \mid S_1 = 0) P(C_1 = 0 \mid S_1 = 0)
\end{aligned}$$

Once we have computed each $\alpha_3(j)$, we can compute the overall likelihood of the observations by summing over them:

$$P(e_1, e_2, e_3) = \sum_j P(e_1, e_2, e_3, S_3 = j) = \sum_j \alpha_3(j)$$

The computations can get very tedious so we will use Python to step through the α-update algorithm. Running it will give the final likelihood as $P(e_1, e_2, e_3) \approx 0.01527$.

```
# Let pi be the prior probabilities of the two states
# pi[0] = P(S_1 = 0) and pi[1] = P(S_1 = 1)
pi = [0.3, 0.7]

# Let S store the transition probabilities
```

```
# S[i][j] = P(S_{t+1} = j | S_t = i)
S = [[None, None] for _ in range(2)]
S[1][1] = 0.8
S[1][0] = 0.2
S[0][1] = 0.3
S[0][0] = 0.7

# Let R store the emission probabilities for a student having red eyes
# R[i][j] = P(R_t = j | S_t = i)
R = [[None, None] for _ in range(2)]
R[1][1] = 0.2
R[1][0] = 0.8
R[0][1] = 0.7
R[0][0] = 0.3

# Let C store the emission probabilities for a student sleeping in class
# C[i][j] = P(C_t = j | S_t = i)
C = [[None, None] for _ in range(2)]
C[1][1] = 0.1
C[1][0] = 0.9
C[0][1] = 0.3
C[0][0] = 0.7

# Let the evidence e_t at time t be a tuple (R_t, C_t)
# We store the evidence in the list below
evidence = [
    (None, None), # dummy value for t = 0 so we can 1-index the list
    (0, 0), # not red eyes, not sleeping in class
    (1, 0), # red eyes, not sleeping in class
    (1, 1)  # red eyes, sleeping in class
]

# Now we will run the forward/alpha-update algorithm for T iterations
T = 3

# We will store the forward probabilities in the matrix alpha
# alpha[t][i] = alpha_t(i) = P(S_t = i, R_1, ..., R_t, C_1, ..., C_t)
alpha = [[None, None] for _ in range(T + 1)]

# Base case
# alpha_1(i) = pi[i] * P(R_1 | S_1 = i) * P(C_1 | S_1 = i)
# alpha_1(i) = pi[i] * R[i][evidence[1][0]] * C[i][evidence[1][1]]
alpha[1] = [pi[i] * R[i][evidence[1][0]] * C[i][evidence[1][1]] for i in range(2)]
print("alpha_1(0) =", alpha[1][0])
print("alpha_1(1) =", alpha[1][1])

# Forward/alpha-update step
for t in range(2, T + 1):
    # alpha_t(j) = sum_{i} alpha_{t-1}(i) * P(S_t = j | S_{t-1} = i) * P(R_t | S_t = j) * P(C_t | S_t = j)
    for j in [0, 1]:
        alpha[t][j] = sum(alpha[t - 1][i] * S[i][j] * R[j][evidence[t][0]] * C[j][evidence[t][1]] for i in
    ↪ [0, 1])
    print("alpha_" + str(t) + "(0) =", alpha[t][0])
    print("alpha_" + str(t) + "(1) =", alpha[t][1])

# The final result is the sum of alpha_T(i) for all i
likelihood = sum(alpha[T])
print("P(R_1, ..., R_T, C_1, ..., C_T) =", likelihood)
```

(c) Consider the same evidence values at timesteps 1, 2, and 3 as the previous part:

(a) $e_1 = (R_1 = 0, C_1 = 0)$ = not red eyes, not sleeping in class

(b) $e_2 = (R_2 = 1, C_2 = 0)$ = red eyes, not sleeping in class

(c) $e_3 = (R_3 = 1, C_3 = 1)$ = red eyes, sleeping in class

Find the most likely sequence of hidden states S_t that produced the evidence above.

Solution: We will now apply the viterbi algorithm instead. Note that finding the most likely sequence of hidden states from the sequence of observations is equivalent to the following problem:

$$\underset{S_{1:T}}{\text{argmax}}\ P(S_{1:T} \mid e_{1:T}) = \underset{S_{1:T}}{\text{argmax}}\ P(S_{1:T}, e_{1:T})$$

We can break apart this maximization into 2 different optimization problems:

$$\max_{S_{1:T}} P(S_{1:T}, e_{1:T}) = \max_{S_T} \max_{S_{1:T-1}} P(S_{1:T}, e_{1:T}) = \max_{S_T} v_T(S_T)$$

where $v_T(S_T)$ is defined as

$$v_T(S_T) = \max_{S_{1:T-1}} P(S_{1:T}, e_{1:T})$$

We can interpret $v_t(S_t)$ as the probability of seeing state S_t, given the observations and assuming that the HMM passed through the most probable sequence of states S_1 to S_{t-1}.

Moreover,

$$\begin{aligned} v_T(S_T) = \max_{S_{1:T-1}} P(S_{1:T}, e_{1:T}) &= \max_{S_{1:T-1}} P(S_{1:T-1}, e_{1:T-1}) P(S_T \mid S_{T-1}) P(e_T \mid S_T) \\ &= \max_{S_{T-1}} \max_{S_{1:T-2}} P(S_{1:T-1}, e_{1:T-1}) P(S_T \mid S_{T-1}) P(e_T \mid S_T) \\ &= \max_{S_{T-1}} P(S_T \mid S_{T-1}) P(e_T \mid S_T) \max_{S_{1:T-2}} P(S_{1:T-1}, e_{1:T-1}) \\ &= \max_{S_{T-1}} P(S_T \mid S_{T-1}) P(e_T \mid S_T) v_{T-1}(S_{T-1}) \end{aligned}$$

This yields a recursive relation that can be turned into a dynamic programming algorithm. We can also define the base case:

$$v_1(S_1) = P(S_1, e_1) = P(S_1) P(e_1 \mid S_1)$$

Note that this is the same base case as the α-update algorithm! In fact, the only difference between the α-update algorithm and the viterbi algorithm is that you can replace the sum with a max. You can also see this similarity by comparing the code for the α-update algorithm above with the code for the viterbi algorithm below.

Once the viterbi probabilities have been computed using the recursive relation above, we can backtrack through them to find the most likely sequence of hidden states. For the sequence of observations given in this problem, we get

- $S_1 = 1$: the student got enough sleep on day 1
- $S_2 = 0$: the student did not get enough sleep on day 2
- $S_3 = 0$: the student did not get enough sleep on day 3

Intuitively, this sequence of hidden states should make sense if you think about what the observations on each day are describing.

```python
# Let pi be the prior probabilities of the two states
# pi[0] = P(S_1 = 0) and pi[1] = P(S_1 = 1)
pi = [0.3, 0.7]

# Let S store the transition probabilities
# S[i][j] = P(S_{t+1} = j | S_t = i)
S = [[None, None] for _ in range(2)]
S[1][1] = 0.8
S[1][0] = 0.2
S[0][1] = 0.3
S[0][0] = 0.7

# Let R store the emission probabilities for a student having red eyes
# R[i][j] = P(R_t = j | S_t = i)
R = [[None, None] for _ in range(2)]
R[1][1] = 0.2
R[1][0] = 0.8
R[0][1] = 0.7
R[0][0] = 0.3

# Let C store the emission probabilities for a student sleeping in class
# C[i][j] = P(C_t = j | S_t = i)
C = [[None, None] for _ in range(2)]
C[1][1] = 0.1
C[1][0] = 0.9
C[0][1] = 0.3
C[0][0] = 0.7

# Let the evidence e_t at time t be a tuple (R_t, C_t)
# We store the evidence in the list below
evidence = [
    (None, None), # dummy value for t = 0 so we can 1-index the list
    (0, 0), # not red eyes, not sleeping in class
    (1, 0), # red eyes, not sleeping in class
    (1, 1)  # red eyes, sleeping in class
]

# Now we will run the viterbi algorithm for T iterations
T = 3

# We will store the viterbi probabilities in the matrix v
# v[t][i] = v_t(i) = P(S_t = i, R_1, ..., R_t, C_1, ..., C_t)
v = [[None, None] for _ in range(T + 1)]
# We will also store a sequence of backpointers in the matrix bp
bp = [[None, None] for _ in range(T + 1)]

# Base case
# v_1(i) = pi[i] * P(R_1 | S_1 = i) * P(C_1 | S_1 = i)
# v_1(i) = pi[i] * R[i][evidence[1][0]] * C[i][evidence[1][1]]
v[1] = [pi[i] * R[i][evidence[1][0]] * C[i][evidence[1][1]] for i in range(2)]
print("v_1(0) =", v[1][0])
print("v_1(1) =", v[1][1])
# The base case for the backpointer matrix is trivial
bp[1] = [0, 0]

# Viterbi step
for t in range(2, T + 1):
    # v_t(j) = max_{i} v_{t-1}(i) * P(S_t = j | S_{t-1} = i) * P(R_t | S_t = j) * P(C_t | S_t = j)
    # bp_t(i) = argmax_{i} v_{t-1}(i) * P(S_t = j | S_{t-1} = i) * P(R_t | S_t = j) * P(C_t | S_t = j)
    for j in [0, 1]:
        v[t][j] = max(v[t - 1][i] * S[i][j] * R[j][evidence[t][0]] * C[j][evidence[t][1]] for i in [0, 1])
        bp[t][j] = max([0, 1], key=lambda i: v[t - 1][i] * S[i][j] * R[j][evidence[t][0]] * C[j][evidence[t
    ↪ ][1]])
    print("v_" + str(t) + "(0) =", v[t][0])
    print("v_" + str(t) + "(1) =", v[t][1])

# The final result is the max of v_T(i) for all i
```

```
likelihood = max(v[T])
print("P(S_1, ..., S_T, R_1, ..., R_T, C_1, ..., CT) =", likelihood)

# Now we will backtrack to find the most likely sequence of states
# Let the most likely sequence of states be stored in the list path
path = [None for _ in range(T + 1)]
path[T] = max([0, 1], key=lambda i: v[T][i])
print("S_" + str(T) + " =", path[T])
for t in range(T - 1, 0, -1):
    path[t] = bp[t + 1][path[t + 1]]
    print("S_" + str(t) + " =", path[t])
# The most likely sequence of states is the list path
print("Most likely sequence of states:", path[1:])
```