67800 - Probabilistic Methods in Artificial Intelligence **(Due: 06/06/24)**

Project 1 - Representation

Instructor: Prof. Gal Elidan *TA:* Ela Fallik *Name:* Hadar Tal

1 Warmup

The solution is above.

2 Sampling

1. Use the sampling procedure described in the recitation to implement in the HMM class the method `sample`.

   Implemented the `sample` method in the `HMM` class.

2. Implement in the HMM class the method `log_joint`, that calculates the joint probability for each sample $p(X_{1:T}, O_{1:T})$.

   Implemented the `log_joint` method in the `HMM` class.

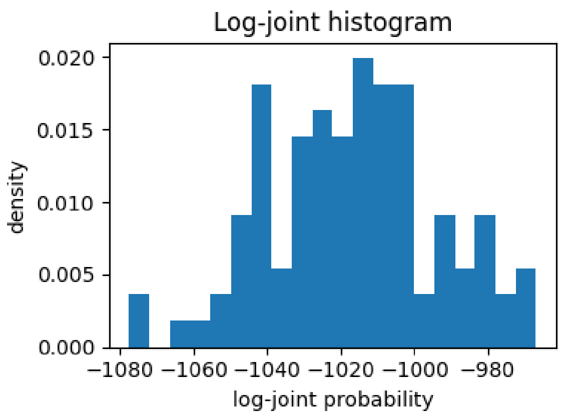
3. Sample $N = 100$ samples $\xi^{[i]} = (X_{1:T}, O_{1:T})$ from the HMM1 model with $T = 1000$. Calculate the joint probability of each sample $p(\xi^{[i]})$.

   Loaded the HMM1 model and sampled $N = 100$ samples from the model with $T = 1000$.

```
Load HMM1. CPDs:
prior
['prior(0)=0.849' 'prior(1)=0.151']
transition_mat
[['tau(0->0)=0.750' 'tau(1->0)=0.250']
['tau(0->1)=0.250' 'tau(1->1)=0.750']]
emission_mat
[['e(0->0)=0.830' 'e(0->1)=0.170']
['e(1->0)=0.170' 'e(1->1)=0.830']]
```

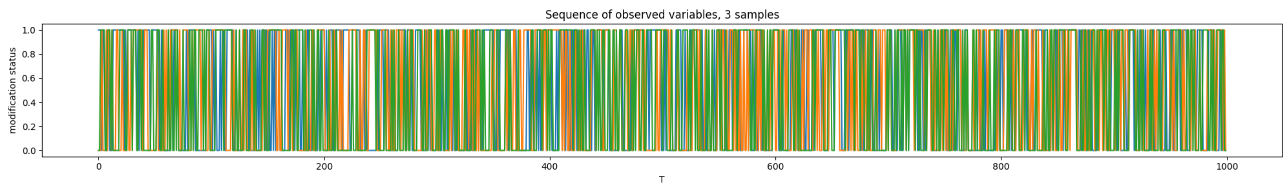
   The log joint probabilities histogram is shown below.

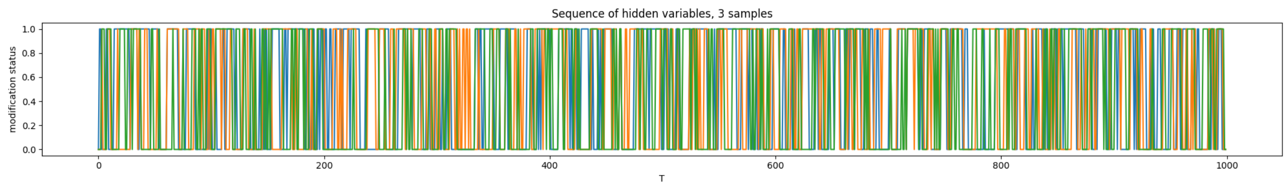
4. Plot the histogram of the log joint probabilities [*]. Why are the joint probabilities so small, even though the samples were sampled from the "correct" model?

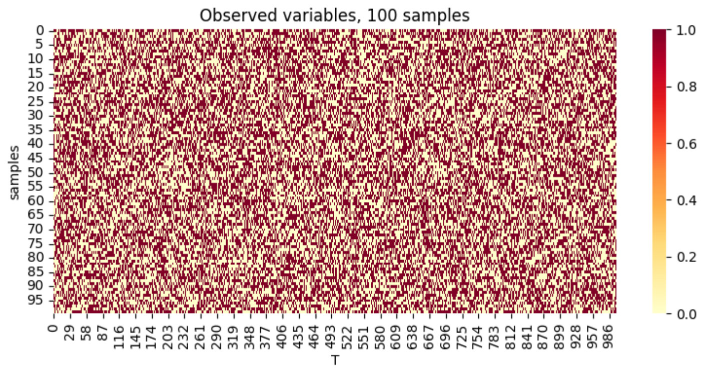


The joint probabilities are so small because they are the product of the prior, transition, and emission probabilities. The transition matrix and the emission matrix are relatively far from the identity matrix. This implies that the randomness (or entropy) in the model is large, meaning that the system transitions between states frequently and the observations can vary significantly. As a result, each individual sequence of observations and hidden states becomes rare, leading to very small joint probabilities. In essence, the higher the entropy, the more unique each sample is, which decreases the probability of any specific sample occurring.

5. Plot 3 examples of the sampled observations on top of the genome [*]. Plot a heatmap of all samples [*].

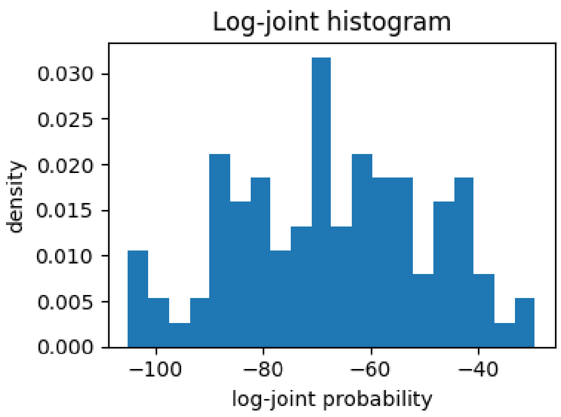


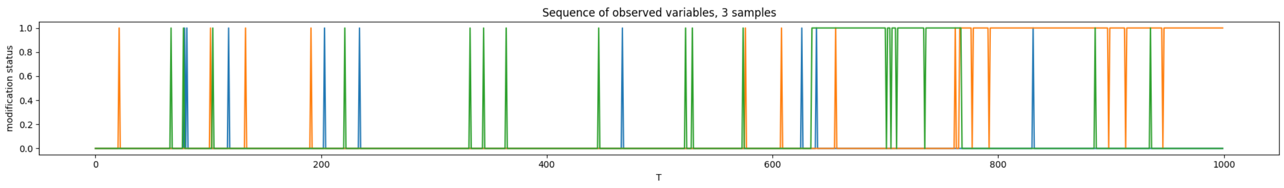


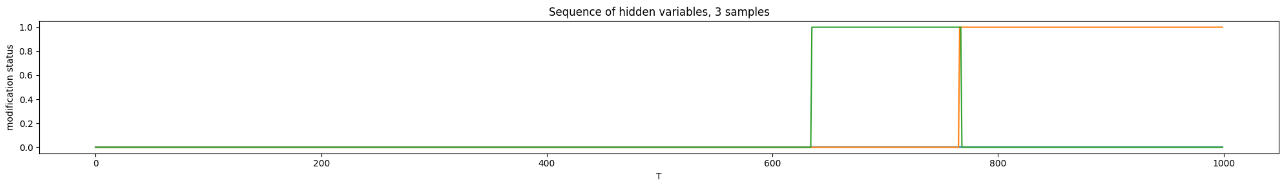


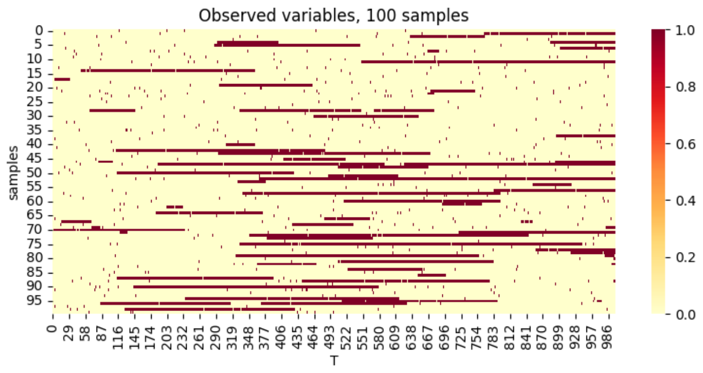
6. Do the same (3-5) for HMM2 with $T = 1000$. What are the differences? How can they be explained?

```
Load HMM2. CPDs:
prior
['prior(0)=1.000' 'prior(1)=0.000']
transition_mat
[['tau(0->0)=0.999' 'tau(0->1)=0.001']
['tau(1->0)=0.005' 'tau(1->1)=0.995']]
emission_mat
[['e(0->0)=0.990' 'e(0->1)=0.010']
['e(1->0)=0.010' 'e(1->1)=0.990']]
```









The differences between the results of HMM1 and HMM2 are evident in the log joint probabilities and the behavior of hidden and observed sequences:

(a) **Log Joint Probabilities:**
- **HMM1:** Log joint probabilities are much smaller (more negative).
- **HMM2:** Log joint probabilities are higher (less negative).

(b) **Hidden and Observed Sequences:**
- **HMM1:** Frequent transitions between states, indicating high randomness.
- **HMM2:** Fewer transitions, indicating more stability and persistence in states.

Explanation

(a) **Transition and Emission Matrices:**

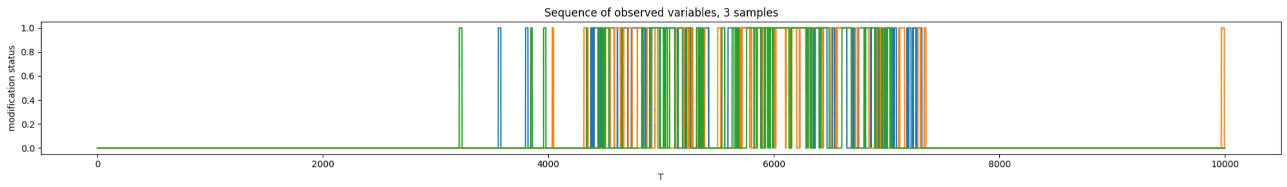
- **HMM1:** Matrices far from the identity matrix, implying high entropy (more randomness). This results in lower log joint probabilities due to the rarity of each sequence.
- **HMM2:** Matrices closer to the identity matrix, implying lower entropy (less randomness). This results in higher log joint probabilities because sequences are more predictable.

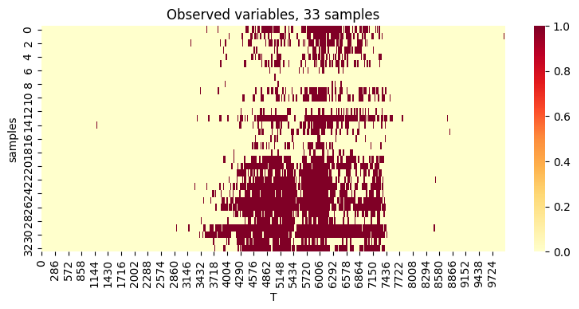
(b) **Entropy and Probability Distribution:**

- **High Entropy (HMM1):** Greater randomness means lower probability for specific sequences, resulting in very negative log joint probabilities.
- **Low Entropy (HMM2):** Less randomness increases the probability for specific sequences, resulting in higher log joint probabilities.

In summary, HMM1's higher entropy leads to more random and rare sequences with lower log joint probabilities, while HMM2's lower entropy leads to more stable and common sequences with higher log joint probabilities.

7. Load real observations from the `small_binary_data.csv` file [*]. Plot these observations same as the sampled data (3) [*]. What is the pattern in the real data? Explain the differences. Does this data fit the HMM model assumptions?





The real data from `small_binary_data.csv` shows a distinct pattern: many 1's cluster in the middle of the sequence, while 0's appear predominantly at the beginning and end. This indicates the sequence does not follow the typical Markov properties assumed in HMMs.

Differences and Explanation

(a) **Pattern Differences:**

- **Real Data:** Clear structure with 1's clustering in the middle and 0's at the edges.
- **HMM Sampled Data:** More randomness and frequent state transitions, as expected from a Markov process.

(b) **Explanation:**

- Real data exhibits non-Markovian behavior where state occurrences are not solely dependent on the previous state, indicating a structured or periodic phenomenon.
- This clustering pattern suggests that transitions between states are not memoryless, which is a fundamental assumption of HMMs.

The real data shows a clear deviation from HMM assumptions, evidenced by the structured clustering of states. This suggests the real-world process generating this data has a different underlying mechanism than what an HMM can capture.

3 Calculating Prior, Likelihood, and Posterior

1. We define a subproblem for each $t = 1, \dots, T$ and $k \in \mathrm{Val}(X)$:

$$P[t, k] = p(X_t = k)$$

How can we calculate $P[1, k]$ directly from the network CPDs?

$P[1, k] = Pr(X_1 = k)$ and is given directly by the initial state distribution (prior distribution) of the HMM.

2. Using the multiplicative rule, we get a recursive formula for P:

$$P[t, k] = p(X_t = k) = \sum_{l \in \mathrm{Val}(X)} p(X_t = k | X_{t-1} = l) \cdot p(X_{t-1} = l) = \sum_{l} p(X_t = k | X_{t-1} = l) \cdot P[t-1, l]$$

Using this formula, we can iteratively fill a table $P_{|\mathrm{Val}(X)| \times T}$ by first filling the first column with the start conditions $P[1, k]$, and continuing by columns: At each stage t, we use the t-th column to compute the $(t+1)$-th. Implement in the HMM class the method `log_p_Xt`, that calculates the table $\log P[t, k] = \log p(X_t = k)$.

Implemented the `log_prior_Xt` method in the `HMM` class.

```
def log_prior_Xt(self):
    """
    :return point-wise prior. shape = (T, |val(X)|)

    P'[t, k] = p(X_t = k) = sum_{l in Val(x)} p(X_t = k | X_{t-1}=l) * p(X_{t-1} = l)
           = sum_{l in Val(x)} P(X_t = k | X_{t-1}=l) * P'[t-1, l]
           = P(X_t = k | X_{t-1}=l_1) * P'[t-1, l_1] + ...  P(X_t = k | X_{t-1}=l_n) * P'[t-1, l_n]

    P[t, k] = logp(X_t = k) = logp( exp(   logp(X_t = k | X_{t-1}=l_1) + P[t, l_1])   ) + ...
                                + exp(   logp(X_t = k | X_{t-1}=l_n) + P[t, l_n])   ) )
    """
```

3. Implement in the HMM class the method `log_p_Xt_given_Ot`, that calculates $\log p(X_t | o_t)$.

Implemented the `log_naive_posterior_Xt` method in the `HMM` class.

```
def log_naive_posterior_Xt(self, obs):
    """
    :param obs - N observations. shape = (N,T)
    :return point-wise posterior. shape = (N, T, |val(X)|)

    log( p(X_t = x_t | O_t = o_t) )
           = log(     p(X_t = x_t) * p(O_t = o_t | X_t = x_t)    /
                   sum_{x in Val(X)}  p(X_t = x) * p(O_t = o_t | X_t = x)   )
           = log_p(X_t = x) + log_p(O = o_t| X = x)
                   - log(sum_{x in Val(X)}  p(X_t = x) * p(O = o_t | X = x)   )
           = log_p(X_t = x) + log_p(O = o_t| X = x)
                   - log(sum_{x in Val(X)}  exp( log_p(X_t = x) + log_p(O = o_t | X = x)   )

    """
```

4. Next, we supply you with a function `_log_forward` that calculates the log of the table $\log F[t, k] = \log p(X_t = k, o_{1:t})$ in a similar way to the log prior (we'll talk about this algorithm later in the course) [*]. Implement in the HMM class a method `log_likelihood`, that calculates the log-likelihood of the observations $\log p(o_{1:m})$.

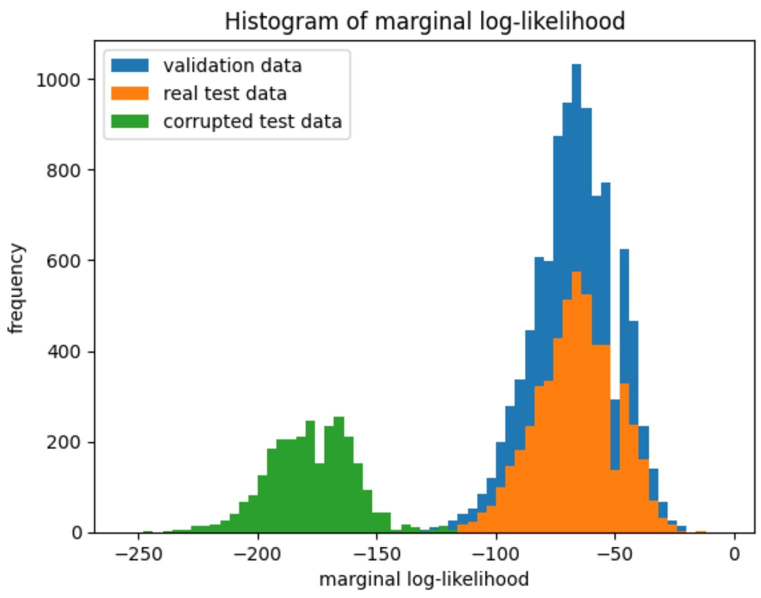
Implemented the `log_likelihood` method in the `HMM` class.

```
def log_likelihood(self, obs):
    """
    :param obs - N observations. shape = (N,T)
    :return log-likelihood. shape = (N)

    log(p(o_[1:T]) = log (     sum_{k in Val(X)} p(o_1, ... , o_T, k) )      )
                   = log( sum_{k in Val(X)} exp ( log_p(o_1, ... , o_T, k) ) )
                   = log( sum_{k in Val(X)} exp ( F[T,k] )     )
    """
```

4 Identifying Corrupt Data

Implemented the function `Q4_identify_corrupt_data` with comments that divide the function into the required steps.



The histogram of marginal log-likelihoods shows distinct distributions for validation data, real test data, and corrupted test data. Here's an explanation of the results:
Observations

1. **Validation Data (Blue):**
   - Peak around -50 to -100.
   - Roughly normal distribution centered around these values.
2. **Real Test Data (Orange):**
   - Overlaps with validation data but slightly shifted to lower values.
   - Indicates similarity to validation data with some deviations due to noise.
3. **Corrupted Test Data (Green):**
   - Distinct peak around -150 to -200.
   - Significantly lower values indicate substantial deviations and anomalies.

Key Points

- **Overlap Between Validation and Real Test Data:** Indicates the model captures real data patterns well, with slight variations.
- **Separation of Corrupted Data:** Clear separation with lower log-likelihoods shows the model effectively identifies anomalies.

Conclusion
The histogram demonstrates the model's ability to distinguish between real and corrupted test data based on marginal log-likelihoods.

5 Predict Active Promoters

Say we want to predict the most likely segmentation to active promoters given a sequence of modifications. We'll try to use the following rule:

$$\forall t, \hat{X}_t = \arg\max_x p(X_t|O_t)$$

1. Implement this rule in the `naive_predict` method of the HMM class (use the point-wise posterior from Section 3).

   Implemented the `naive_predict_by_naive_posterior` method in the `HMM` class.

2. Sample N = 100 samples $\{\xi^{[i]}\}i = 1^N$ from HMM3 and predict their active promoter sequence $\{\hat{X}^{[i]}\}_{i=1}^N$.
   Loaded the HMM3 model and sampled N = 100 samples from the model.

```
Load HMM3. CPDs:
prior
['prior(0)=1.000' 'prior(1)=0.000']
transition_mat
[['tau(0->0)=0.999' 'tau(0->1)=0.001']
['tau(1->0)=0.000' 'tau(1->1)=1.000']]
emission_mat
[['e(0->0)=0.990' 'e(0->1)=0.010']
['e(1->0)=0.010' 'e(1->1)=0.990']]
```

3. Calculate the single-location accuracy of the prediction:

$$\text{single-location-accuracy} = \frac{1}{N \cdot T} \sum_{i=1}^{N} \sum_{t=1}^{T} \mathbf{1}\{\xi[i]_t^k = \hat{X}[i]_t^k\}$$

   What is the accuracy of the naive-predictor? Is this a good prediction rule? If not, suggest a better rule (no need to implement).

   Accuracy of the Naive Predictor
   The naive prediction accuracy is 0.98989.
   Confusion Matrix

$$\begin{bmatrix} 62048 & 641 \\ 370 & 36941 \end{bmatrix}$$

   Invalid Transitions
   There are 994 invalid transitions from state 1 to state 0.
   Evaluation of the Prediction Rule
   Is this a good prediction rule? No, the naive predictor is not ideal as it ignores temporal dependencies and only considers the point-wise posterior $p(X_t|O_t)$.
   Reasons for Its Performance:
   The high accuracy is due to the high diagonal values in the transition and emission matrices, indicating a tendency to remain in the same state and strong state-indicative observations.
   Suggested Improvement
   A better predictor would use the Viterbi algorithm to account for temporal dependencies, optimizing the state sequence globally rather than point-wise, thus improving overall prediction accuracy.