Sequence to Sequence modelling(HMM)

Hidden Markov Models (HMMs) are incredibly versatile tools for modeling sequential data where we deal with hidden states that generate observable events.

Key Concepts of HMM

1. **Hidden States**:
   - Represent the unobservable states of the system (e.g., "Sunny" or "Rainy").
   - Transition probabilities govern the likelihood of moving between these states.
2. **Observations**:
   - Represent observable outcomes generated by the hidden states (e.g., "Dry" or "Wet").
   - Emission probabilities define the likelihood of observing a specific event given the current hidden state.
3. **Core Probabilities**:
   - **Initial State Distribution**: Probability of starting in each hidden state.
   - **Transition Probabilities**: Probability of moving from one hidden state to another.
   - **Emission Probabilities**: Probability of observing a specific event given a hidden state.

HMM Algorithm Workflow

1. **Define Model Parameters**:
   - State space, observation space, initial probabilities, transition matrix, and emission matrix.
2. **Training the Model**:
   - Use algorithms like Baum-Welch (or Forward-Backward) to estimate model parameters from data.
3. **Decoding**:
   - Use the Viterbi algorithm to find the most likely sequence of hidden states given observations.
4. **Evaluation**:
   - Validate the model using metrics such as accuracy or likelihood scores.

Python Implementation Examples

Example 1: Weather Prediction

Code:

Step 1: Import Required Libraries

```
import numpy as np
from hmmlearn import hmm
import matplotlib.pyplot as plt
import seaborn as sns
```

Explanation:

- **`numpy`**: Used for numerical computations like creating probability matrices and observation sequences.
- **`hmmlearn`**: Provides tools to implement Hidden Markov Models (HMMs). In this example, we use `hmm.CategoricalHMM` to define and work with discrete observation spaces.
- **`matplotlib` and `seaborn`**: Used to visualize results like predicted hidden states.

Step 2: Define States and Observations

```
# Define states and observations
states = ["Sunny", "Rainy"]
observations = ["Dry", "Wet"]
n_states = len(states)
n_observations = len(observations)
```

Explanation:

- **`states`**: The hidden states of the model (`Sunny` and `Rainy`), which are not directly observable.
- **`observations`**: The measurable events (`Dry` and `Wet`) that depend on the hidden states.
- **`n_states` and `n_observations`**: Used to specify the number of states and observations in the model.

Step 3: Define Model Parameters

```
# Define model parameters
state_probability = np.array([0.6, 0.4])  # Initial probabilities
transition_probability = np.array([[0.7, 0.3], [0.3, 0.7]])  # Transition
matrix
emission_probability = np.array([[0.9, 0.1], [0.2, 0.8]])  # Emission
matrix
```

Explanation:

- **`state_probability`**:
  - Initial probabilities for each state:
    - 60% chance of starting in `Sunny`.
    - 40% chance of starting in `Rainy`.
- **`transition_probability`**:
  - A **2x2 matrix** describing the likelihood of transitioning between states:

        - If it's `Sunny`, there's a 70% chance it remains `Sunny` and a 30% chance it becomes `Rainy`.
        - If it's `Rainy`, there's a 70% chance it remains `Rainy` and a 30% chance it becomes `Sunny`.
- **`emission_probability`**:
    - A **2x2 matrix** describing the likelihood of observations given the states:
        - If it's `Sunny`, there's a 90% chance of observing `Dry` and a 10% chance of observing `Wet`.
        - If it's `Rainy`, there's a 20% chance of observing `Dry` and an 80% chance of observing `Wet`.

Step 4: Initialize the HMM Model

```
# Create HMM model
model = hmm.CategoricalHMM(n_components=n_states)
model.startprob_ = state_probability
model.transmat_ = transition_probability
model.emissionprob_ = emission_probability
```

Explanation:

- **`hmm.CategoricalHMM(n_components=n_states)`**:
    - Creates an HMM with discrete observation spaces (categories).
    - `n_components` specifies the number of hidden states (`Sunny`, `Rainy`).
- **`model.startprob_`**:
    - Sets the initial state probabilities.
- **`model.transmat_`**:
    - Sets the state transition probabilities.
- **`model.emissionprob_`**:
    - Sets the emission probabilities.

Step 5: Define Observation Sequence

```
# Define sequence of observations
observations_sequence = np.array([0, 1, 0, 1, 0, 0]).reshape(-1, 1)
```

Explanation:

- **`observations_sequence`**:
    - Represents observed events over time:
        - `0`: "Dry"
        - `1`: "Wet"
    - The sequence `[0, 1, 0, 1, 0, 0]` means:
        - Time step 1: Dry
        - Time step 2: Wet
        - Time step 3: Dry, and so on.
- **`.reshape(-1, 1)`**:

- Reshapes the sequence into a column vector, as required by `hmmlearn`.

Step 6: Predict the Most Likely Sequence of Hidden States

```
# Predict hidden states
hidden_states = model.predict(observations_sequence)
```

Explanation:

- **`model.predict()`**:
  - Uses the HMM model to predict the **hidden state sequence** for the given observations.
  - Returns the most likely sequence of hidden states based on the input observations.

Step 7: Decode Observations with Viterbi Algorithm

```
# Decode observations using Viterbi algorithm
log_prob, viterbi_hidden_states = model.decode(observations_sequence,
algorithm="viterbi")
```

Explanation:

- **`model.decode()`**:
  - Uses the Viterbi algorithm to compute:
    - **Log Probability (`log_prob`)**: Likelihood of the decoded hidden state sequence.
    - **Viterbi Hidden States (`viterbi_hidden_states`)**: The most probable sequence of hidden states.

Step 8: Display Results

```
print("Most likely hidden states:", hidden_states)
print("Viterbi Log Probability:", log_prob)
```

Explanation:

- Displays the predicted hidden states (`hidden_states`) and the log probability of the sequence (`log_prob`).

Step 9: Visualize Results

```
# Plotting
```

```
sns.set_style("whitegrid")
plt.plot(hidden_states, '-o', label="Predicted Hidden State")
plt.xlabel("Time Step")
plt.ylabel("Hidden State (Sunny=0, Rainy=1)")
plt.title("Weather Prediction (Sunny or Rainy)")
plt.legend()
plt.show()
```

Explanation:

- **Plot**:
  - X-axis: Time steps.
  - Y-axis: Predicted hidden states (0 = `Sunny`, 1 = `Rainy`).
  - Visualization provides an intuitive view of how the model predicts state transitions over time.

The **Viterbi algorithm** is a dynamic programming algorithm used to find the most probable sequence of hidden states (also called the **optimal path**) in a **Hidden Markov Model (HMM)**, given a sequence of observations. It is a core method in applications like speech recognition, natural language processing, and bioinformatics.

When working with HMMs, we often want to infer the **hidden state sequence** that most likely explains the observed sequence. The Viterbi algorithm efficiently computes this sequence by:

1. Maximizing the probability of the entire path, rather than individual states.
2. Avoiding a brute-force computation of all possible paths (which would be computationally expensive).

Example 2: Speech Recognition step-by-step.

Step 1: Import Required Libraries

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from hmmlearn import hmm
```

Explanation:

- The libraries here are identical to the weather prediction example:
  - **`numpy`**: Numerical computations.
  - **`hmmlearn`**: Provides the `CategoricalHMM` class to work with discrete observations.
  - **`matplotlib` and `seaborn`**: For visualizing the predicted hidden states.

Step 2: Define States and Observations

```
# Define the state space
states = ["Silence", "Word1", "Word2", "Word3"]
n_states = len(states)

# Define the observation space
observations = ["Loud", "Soft"]
n_observations = len(observations)
```

Explanation:

- **`states`**:
  - Hidden states in this model represent:
    - "Silence" (no speech).
    - Three distinct words: `Word1`, `Word2`, and `Word3`.
  - These states are not directly observable.
- **`observations`**:
  - The measurable events ("Loud" and "Soft") are volume levels of the speech.

Step 3: Define Model Parameters

```
# Define the initial state distribution
start_probability = np.array([0.8, 0.1, 0.1, 0.0])

# Define the state transition probabilities
transition_probability = np.array([[0.7, 0.2, 0.1, 0.0],
                                    [0.0, 0.6, 0.4, 0.0],
                                    [0.0, 0.0, 0.6, 0.4],
                                    [0.0, 0.0, 0.0, 1.0]])

# Define the observation likelihoods
emission_probability = np.array([[0.7, 0.3],
                                  [0.4, 0.6],
                                  [0.6, 0.4],
                                  [0.3, 0.7]])
```

Explanation:

- **Initial State Probabilities (`start_probability`)**:
  - High likelihood (80%) of starting in the "Silence" state.
  - Lower probabilities (10%) of starting in `Word1` or `Word2`. No chance of starting in `Word3`.
- **Transition Probabilities (`transition_probability`)**:
  - Defines the likelihood of moving between states:
    - "Silence" tends to remain silent (70%) but can transition to `Word1` (20%) or `Word2` (10%).
    - `Word1` can transition to `Word2` (40%) or remain in `Word1` (60%).
    - `Word2` can transition to `Word3` (40%) or remain in `Word2` (60%).
    - `Word3` always remains in `Word3` (100%).
- **Emission Probabilities (`emission_probability`)**:
  - Defines the likelihood of observing "Loud" or "Soft" given a hidden state:

    - "Silence" is more likely to produce "Loud" (70%).
    - `Word1` and `Word2` have mixed likelihoods of emitting "Loud" or "Soft".
    - `Word3` is more likely to produce "Soft" (70%).

Step 4: Initialize the HMM Model

```
# Fit the model
model = hmm.CategoricalHMM(n_components=n_states)
model.startprob_ = start_probability
model.transmat_ = transition_probability
model.emissionprob_ = emission_probability
```

Explanation:

- The HMM model is created using the `hmm.CategoricalHMM` class with:
  - **`n_components=n_states`**: Specifies 4 hidden states (Silence, Word1, Word2, Word3).
- The model parameters are set:
  - **Initial probabilities**: `start_probability`.
  - **Transition probabilities**: `transition_probability`.
  - **Emission probabilities**: `emission_probability`.

Step 5: Define Observation Sequence

```
# Define the sequence of observations
observations_sequence = np.array([0, 1, 0, 0, 1, 1, 0, 1]).reshape(-1, 1)
```

Explanation:

- **`observations_sequence`**:
  - Represents observed events over time:
    - `0`: "Loud"
    - `1`: "Soft"
  - The sequence `[0, 1, 0, 0, 1, 1, 0, 1]` corresponds to:
    - Time step 1: Loud
    - Time step 2: Soft
    - Time step 3: Loud, and so on.

Step 6: Predict the Most Likely Sequence of Hidden States

```
# Predict the most likely hidden states
hidden_states = model.predict(observations_sequence)
print("Most likely hidden states:", hidden_states)
```

Explanation:

- **`model.predict()`**:
  - Computes the **most probable sequence of hidden states** that explains the given observations.
- **Output (`hidden_states`)**:
  - A sequence of integers corresponding to the hidden states:
    - `0`: "Silence"
    - `1`: "Word1"
    - `2`: "Word2"
    - `3`: "Word3"

Step 7: Visualize Results

```
# Plot the results
sns.set_style("darkgrid")
plt.plot(hidden_states, '-o', label="Predicted Hidden State")
plt.xlabel("Time Step")
plt.ylabel("Hidden State")
plt.title("Speech Recognition Hidden States")
plt.legend()
plt.show()
```

Explanation:

- The plot shows how the model predicts hidden states over time.
- **X-axis**: Time steps.
- **Y-axis**: Hidden states (Silence, Word1, Word2, Word3).

Output Example

Predicted Hidden States:

```
Most likely hidden states: [0 1 2 2 3 3 3 3]
```

Interpretation:

- **Time Step 1**: "Silence".
- **Time Step 2**: Transitions to "Word1".
- **Time Steps 3 & 4**: "Word2".
- **Time Steps 5-8**: "Word3".

Visualization:

- The graph reflects these transitions, illustrating the most likely hidden states over time.

Applications in Speech Recognition

1. **State Space**: Represents phonemes, words, or speech segments (e.g., Silence, Word1, etc.).
2. **Observation Space**: Represents features of speech (e.g., volume, frequency).
3. **Use Cases**:
   - Identify words or phrases spoken in audio data.
   - Segment audio into meaningful parts (e.g., silence vs. spoken words).