```
"""
STEP 1: Import required libraries for preprocessing,
POS tagging, and HMM parameter building.
"""

import pandas as pd
import re
import nltk
from nltk import word_tokenize, pos_tag
from collections import defaultdict, Counter

# Download required NLTK models (run once)
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
True
```

```
import pandas as pd
import re
import nltk
from nltk import word_tokenize, pos_tag
from collections import defaultdict, Counter

# Download required NLTK models (run once)
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')

"""
STEP 2: Load the Twitter sentiment dataset from CSV file.
Change filename and column name if needed.
"""

# Load dataset
df = pd.read_csv("Twitter_Data.csv")   # change filename if required

# Extract tweet text column and remove missing values
tweets = df["clean_text"].dropna().tolist() # Corrected column name to 'clean_text'

# Show dataset size and sample
print("Total tweets loaded:", len(tweets))
print(tweets[:3])
```

```
Total tweets loaded: 16089
['when modi promised “minimum government maximum governance” expected him begin the difficul
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

```
"""
STEP 3: Define a function to clean noisy tweet text.
Removes URLs, mentions, hashtags, and special characters.
"""
```

```
def clean_tweet(text):
    """
    Clean tweet by removing noise.

    Args:
        text (str): Raw tweet text

    Returns:
        str: Cleaned tweet
    """
    text = re.sub(r"http\S+", "", text)       # remove URLs
    text = re.sub(r"@\w+", "", text)          # remove mentions
    text = re.sub(r"#", "", text)             # remove hashtag symbol
    text = re.sub(r"[^a-zA-Z\s]", "", text)   # remove punctuation/emojis
    return text.lower().strip()
```

```
"""
STEP 4: Apply cleaning function to all tweets.
"""

cleaned_tweets = [clean_tweet(t) for t in tweets]

# Show sample cleaned tweets
print("Cleaned tweet samples:")
print(cleaned_tweets[:5])
```

```
Cleaned tweet samples:
['when modi promised minimum government maximum governance expected him begin the difficult
```

```
"""
STEP 5: Define POS tagging function using NLTK.
This provides weak supervision labels.
"""

def pos_tag_tweets(tweet_list, limit=500):
    """
    Tokenize and POS-tag tweets.

    Args:
        tweet_list (list): Cleaned tweets
        limit (int): Number of tweets to tag

    Returns:
        list: POS tagged sentences
    """
    tagged_sentences = []

    for tweet in tweet_list[:limit]:
        tokens = word_tokenize(tweet)   # split tweet into words

        if tokens:                      # skip empty tweets
            tagged = pos_tag(tokens)    # assign POS tags
            tagged_sentences.append(tagged)

    return tagged_sentences
```

```
import nltk
nltk.download('punkt_tab')
nltk.download('averaged_perceptron_tagger_eng') # Download the specific tagger for English

"""
STEP 6: Generate POS tagged tweets.
"""

tagged_tweets = pos_tag_tweets(cleaned_tweets, limit=500)

# Show example tagged tweet
print("Sample POS tagged tweet:")
print(tagged_tweets[0])
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger_eng.zip.
Sample POS tagged tweet:
[('when', 'WRB'), ('modi', 'NN'), ('promised', 'VBD'), ('minimum', 'JJ'), ('government', 'NN
```

```
"""
STEP 7: Define function to compute HMM parameters:
- Transition probabilities
- Emission probabilities
"""

def build_hmm_parameters(tagged_sentences):
    """
    Build HMM transition and emission probabilities.

    Returns:
        transition_probs, emission_probs, tag_counts
    """

    transition_counts = defaultdict(Counter)
    emission_counts = defaultdict(Counter)
    tag_counts = Counter()

    # Count tag transitions and emissions
    for sentence in tagged_sentences:
        prev_tag = "<START>"

        for word, tag in sentence:
            transition_counts[prev_tag][tag] += 1
            emission_counts[tag][word] += 1
            tag_counts[tag] += 1
            prev_tag = tag

        # Mark end of sentence
        transition_counts[prev_tag]["<END>"] += 1

    # Convert transition counts to probabilities
    transition_probs = {}
    for prev_tag in transition_counts:
        total = sum(transition_counts[prev_tag].values())
        transition_probs[prev_tag] = {
            tag: count / total
            for tag, count in transition_counts[prev_tag].items()
        }

    # Convert emission counts to probabilities
    emission_probs = {}
    for tag in emission_counts:
        total = sum(emission_counts[tag].values())
        emission_probs[tag] = {
            word: count / total
            for word, count in emission_counts[tag].items()
        }

    return transition_probs, emission_probs, tag_counts
```

```
"""
STEP 8: Build HMM probabilities from tagged tweets.
"""

transition_probs, emission_probs, tag_counts = build_hmm_parameters(tagged_tweets)

print("HMM parameters computed.")
print("Total tags:", len(tag_counts))
```

```
HMM parameters computed.
Total tags: 31
```

```
"""
STEP 9: Display sample transition probabilities.
"""

print("Transition probability snapshot:")

for prev_tag in list(transition_probs.keys())[:3]:
    print(prev_tag, list(transition_probs[prev_tag].items())[:5])
```

```
Transition probability snapshot:
<START> [('WRB', 0.034), ('NN', 0.378), ('WP', 0.008), ('VBG', 0.02), ('JJ', 0.114)]
WRB [('NN', 0.23711340206185566), ('VBZ', 0.010309278350515464), ('JJS', 0.01030927835051546
NN [('VBD', 0.05525040387722133), ('JJ', 0.04975767366720517), ('VBG', 0.02294022617124394),
```

```
"""
STEP 10: Display sample emission probabilities.
"""

print("Emission probability snapshot:")

for tag in list(emission_probs.keys())[:3]:
    print(tag, list(emission_probs[tag].items())[:5])
```

```
Emission probability snapshot:
WRB [('when', 0.2268041237113402), ('why', 0.38144329896907214), ('how', 0.24742268041237114
NN [('modi', 0.09402261712439418), ('government', 0.006138933764135703), ('governance', 0.00
VBD [('promised', 0.003236245954692557), ('expected', 0.006472491909385114), ('did', 0.05177
```

```
"""
STEP 11: Identify rare words (appear only once).
Rare words create sparsity problems in HMM.
"""

rare_words = []
```

```
    for tag in emission_probs:
        for word, prob in emission_probs[tag].items():
            # reverse lookup count using probability * tag count
            # approximate rare detection by low probability
            if prob < 0.001:
                rare_words.append(word)

    print("Rare word count:", len(rare_words))
    print("Sample rare words:", rare_words[:20])
```

```
Rare word count: 1728
Sample rare words: ['justice', 'business', 'psus', 'talk', 'nonsense', 'drama', 'campaigner'
```

```
"""
STEP 12: Define simplified Viterbi demonstration.
Shows scoring process step-by-step.
"""

def viterbi_demo(tokens, transition_probs, emission_probs, tag_counts):

    print("\n--- Viterbi Demo ---")

    possible_tags = list(tag_counts.keys())
    prev_scores = {"<START>": 1.0}

    for word in tokens:
        new_scores = {}

        for tag in possible_tags:

            # Handle unknown words with small probability
            emit_prob = emission_probs.get(tag, {}).get(word, 1e-6)

            best_score = 0

            for prev_tag in prev_scores:
                trans_prob = transition_probs.get(prev_tag, {}).get(tag, 1e-6)
                score = prev_scores[prev_tag] * trans_prob * emit_prob
                best_score = max(best_score, score)

            new_scores[tag] = best_score

        prev_scores = new_scores

        # Show top tag candidates
        top_tags = sorted(new_scores.items(), key=lambda x: -x[1])[:3]
        print("Word:", word, "Top tags:", top_tags)
```

```
"""
STEP 13: Run Viterbi decoding demo on one tweet.
"""

example = cleaned_tweets[0]
tokens = word_tokenize(example)

print("Example tweet:", example)
viterbi_demo(tokens, transition_probs, emission_probs, tag_counts)
```

```
Example tweet: when modi promised minimum government maximum governance expected him begin t

--- Viterbi Demo ---
Word: when Top tags: [('WRB', 0.0077113402061855674), ('NN', 3.7799999999999997e-07), ('JJ',
Word: modi Top tags: [('NN', 0.0001719167929650417), ('JJ', 7.102945596798269e-05), ('VBP',
Word: promised Top tags: [('VBD', 3.073939237733073e-08), ('VBN', 2.7029954950465275e-08), (
Word: minimum Top tags: [('JJ', 1.0939407200642616e-11), ('NN', 6.727455454338023e-15), ('IN
Word: government Top tags: [('NN', 3.8482173179705617e-14), ('NNS', 1.5694853973430555e-18),
Word: maximum Top tags: [('JJ', 1.6450029339180988e-18), ('NN', 1.3092642442077548e-20), ('I
Word: governance Top tags: [('NN', 1.2182568680483118e-21), ('RB', 5.2901761110621176e-23),
Word: expected Top tags: [('VBD', 4.356581487758444e-25), ('NN', 4.144828698077131e-28), ('I
Word: him Top tags: [('PRP', 1.4177734407197844e-27), ('JJ', 9.023340298269917e-32), ('NN',
Word: begin Top tags: [('VB', 6.432911281376102e-31), ('VBP', 4.989928869572425e-34), ('MD',
Word: the Top tags: [('DT', 5.663492641439937e-32), ('NN', 1.1755573915205058e-37), ('PRP$',
Word: difficult Top tags: [('JJ', 9.883135247358137e-36), ('NN', 2.7653772663280945e-38), ('
Word: job Top tags: [('NN', 2.1957767633155827e-38), ('NNS', 1.4179412253512103e-42), ('JJ',
Word: reforming Top tags: [('VBG', 2.1996338677738617e-42), ('NN', 7.470607210892758e-45), (
Word: the Top tags: [('DT', 1.6659344931966337e-43), ('NN', 5.090855676507191e-49), ('JJ', 4
Word: state Top tags: [('NN', 2.891079271185091e-46), ('JJ', 3.383929439305662e-50), ('NNS',
Word: why Top tags: [('WRB', 1.1401954789344795e-48), ('NN', 9.836208311980292e-53), ('IN',
Word: does Top tags: [('VBZ', 4.088553936116466e-52), ('NN', 2.7035562902570132e-55), ('JJ',
Word: take Top tags: [('VB', 5.955518865232548e-55), ('VBP', 8.706769695367807e-56), ('JJ',
Word: years Top tags: [('NNS', 8.469909116053642e-58), ('NN', 1.0883181682658461e-61), ('DT'
Word: get Top tags: [('VBP', 5.314803397945929e-61), ('VB', 2.5018329482479467e-61), ('NN',
Word: justice Top tags: [('NN', 9.251973868700175e-65), ('JJ', 9.328022290272446e-68), ('DT'
Word: state Top tags: [('NN', 1.11875283049323e-67), ('IN', 8.459801631153956e-72), ('NNS',
Word: should Top tags: [('MD', 3.9908603303556164e-70), ('NN', 3.8062899208703425e-74), ('IN
Word: and Top tags: [('CC', 1.648921161166882e-72), ('VB', 2.93743526853586e-76), ('RB', 5.6
Word: not Top tags: [('RB', 1.6632984394586409e-74), ('NN', 3.834700374806702e-79), ('JJ', 2
Word: business Top tags: [('NN', 8.04685084703497e-79), ('JJ', 3.3799647432849387e-81), ('VB
Word: and Top tags: [('CC', 2.8145989925947914e-80), ('NN', 2.7377492542577784e-85), ('IN',
Word: should Top tags: [('MD', 8.306572401708154e-83), ('NN', 6.5455790525460265e-87), ('JJ'
Word: exit Top tags: [('VB', 3.103540411367678e-85), ('RB', 1.1806295799382147e-89), ('PRP',
Word: psus Top tags: [('NN', 3.664907532510646e-89), ('DT', 4.72620367213352e-92), ('PRP$',
Word: and Top tags: [('CC', 1.2818983780168356e-90), ('NN', 1.2468974577491794e-95), ('IN',
Word: temples Top tags: [('NNS', 1.2849823356223291e-94), ('NN', 2.9811590186438036e-97), ('
```