```python
import pandas as pd
import numpy as np
import re
import nltk
from collections import defaultdict

# Download necessary NLTK data
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('universal_tagset')
nltk.download('punkt_tab') # Added to resolve LookupError
nltk.download('averaged_perceptron_tagger_eng') # Added to resolve new LookupError

# 1. Load and Preprocess
df = pd.read_csv('twitter_dataset.csv')

def preprocess_text(text):
    # Remove URLs
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
    # Remove mentions (@user)
    text = re.sub(r'@\w+', '', text)
    return text.strip()

df['Cleaned_Text'] = df['Text'].apply(preprocess_text)

# 2. POS Tagging
tagged_tweets = []
for text in df['Cleaned_Text']:
    tokens = nltk.word_tokenize(text)
    if tokens:
        # Using universal tagset (NOUN, VERB, ADJ, etc.) for simpler HMM
        tagged = nltk.pos_tag(tokens, tagset='universal')
        tagged_tweets.append(tagged)

print(f"Processed {len(tagged_tweets)} tweets.")
print("Sample tagged tweet:", tagged_tweets[0][:5])
```

```
[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!
[nltk_data] Downloading package universal_tagset to /root/nltk_data...
[nltk_data]   Package universal_tagset is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger_eng.zip.
Processed 10000 tweets.
Sample tagged tweet: [('Party', 'NOUN'), ('least', 'ADJ'), ('receive', 'ADJ'), ('say', 'NOUN'), ('
```

```python
# Initialize counts
tag_counts = defaultdict(int)
tag_pair_counts = defaultdict(lambda: defaultdict(int))
tag_word_counts = defaultdict(lambda: defaultdict(int))
start_tag_counts = defaultdict(int)
```

```
vocab = set()
tags = set()

for tweet in tagged_tweets:
    if not tweet: continue

    # Record start tag
    start_tag = tweet[0][1]
    start_tag_counts[start_tag] += 1

    for i, (word, tag) in enumerate(tweet):
        word = word.lower() # Normalize to lowercase
        vocab.add(word)
        tags.add(tag)

        tag_counts[tag] += 1
        tag_word_counts[tag][word] += 1

        # Record transitions (Previous Tag -> Current Tag)
        if i > 0:
            prev_tag = tweet[i-1][1]
            tag_pair_counts[prev_tag][tag] += 1

tags = sorted(list(tags))
total_tweets = len(tagged_tweets)

# 1. Start Probabilities
start_prob = {tag: count / total_tweets for tag, count in start_tag_counts.items()}

# 2. Transition Probabilities
trans_prob = defaultdict(dict)
for t1 in tags:
    count_out = sum(tag_pair_counts[t1].values())
    for t2 in tags:
        if count_out > 0:
            trans_prob[t1][t2] = tag_pair_counts[t1][t2] / count_out
        else:
            trans_prob[t1][t2] = 0.0

# 3. Emission Probability Helper
def get_emission_prob(tag, word):
    # P(word | tag)
    count = tag_word_counts[tag].get(word, 0)
    total = tag_counts[tag]
    if total == 0: return 0
    return count / total

print("HMM Parameters Built.")
print(f"Tags: {tags}")
```

```
HMM Parameters Built.
Tags: ['.', 'ADJ', 'ADP', 'ADV', 'CONJ', 'DET', 'NOUN', 'NUM', 'PRON', 'PRT', 'VERB', 'X']
```

```
erbi(obs, states, start_p, trans_p, emit_func):
 [{}]
h = {}

ilon = 1e-10 # Small value to handle zero probabilities for unseen events

nitialize base cases (t=0)
 y in states:
```

```
 V[0][y] = (start_p.get(y, 0) + epsilon) * (emit_func(y, obs[0]) + epsilon)
 path[y] = [y]

un Viterbi for t > 0
 t in range(1, len(obs)):
 V.append({})
 newpath = {}

 for y in states:
     # Calculate probability for transitioning to state y from any state y0
     (prob, state) = max(
         (V[t-1][y0] * (trans_p[y0].get(y, 0) + epsilon) * (emit_func(y, obs[t]) + epsilon), y0)
         for y0 in states
     )
     V[t][y] = prob
     newpath[y] = path[state] + [y]

 path = newpath

erminate: Find best final state
ob, state) = max((V[len(obs) - 1][y], y) for y in states)
urn prob, path[state]

n a sample tweet (first 10 words)
idx = 0
tokens = [w.lower() for w, t in tagged_tweets[sample_idx][:10]]
tags = [t for w, t in tagged_tweets[sample_idx][:10]]

ob, predicted_tags = viterbi(sample_tokens, tags, start_prob, trans_prob, get_emission_prob)

Viterbi Decoding Results:")
"Words:     {sample_tokens}")
"Predicted: {predicted_tags}")
"Actual:    {actual_tags}")
```

```
Viterbi Decoding Results:
Words:     ['party', 'least', 'receive', 'say', 'or', 'single', '.', 'prevent', 'prevent', 'husban
Predicted: ['NOUN', 'ADJ', 'NOUN', 'VERB', 'CONJ', 'ADJ', '.', 'NOUN', 'NOUN', 'NOUN']
Actual:    ['NOUN', 'ADJ', 'ADJ', 'NOUN', 'CONJ', 'ADJ', '.', 'NOUN', 'ADJ', 'NOUN']
```

```
import pandas as pd
import numpy as np
import re
from collections import defaultdict

# 1. Load Data
df = pd.read_csv('twitter_dataset.csv')

# 2. Preprocessing
def preprocess_text(text):
    text = re.sub(r'http\S+|www\S+|https\S+', '', text)
    text = re.sub(r'@\w+', '', text)
    text = re.sub(r'[^\w\s]', '', text)
    return text.strip()

df['Cleaned_Text'] = df['Text'].apply(preprocess_text)

# 3. Custom POS Tagger (Fallback for NLTK)
def heuristic_tag(text):
    tokens = text.split()
```

```
        tagged = []
        for token in tokens:
            word = token.lower()
            if word in ['is', 'am', 'are', 'was', 'were']: tag = 'VERB'
            elif word in ['the', 'a', 'an', 'this']: tag = 'DET'
            elif word in ['and', 'or', 'but', 'if']: tag = 'CONJ'
            elif word.endswith('ing') or word.endswith('ed'): tag = 'VERB'
            elif word.endswith('ly'): tag = 'ADV'
            else: tag = 'NOUN' # Default
            tagged.append((token, tag))
        return tagged

    tagged_tweets = [heuristic_tag(t) for t in df['Cleaned_Text']]

    # 4. Train HMM
    tag_counts = defaultdict(int)
    trans_counts = defaultdict(lambda: defaultdict(int))
    emit_counts = defaultdict(lambda: defaultdict(int))
    start_counts = defaultdict(int)

    vocab = set()
    all_tags = set()

    for tweet in tagged_tweets:
        if not tweet: continue
        start_counts[tweet[0][1]] += 1
        for i, (word, tag) in enumerate(tweet):
            w = word.lower()
            vocab.add(w)
            all_tags.add(tag)
            tag_counts[tag] += 1
            emit_counts[tag][w] += 1
            if i > 0:
                trans_counts[tweet[i-1][1]][tag] += 1

    # Normalize Probabilities
    start_prob = {k: v/len(tagged_tweets) for k, v in start_counts.items()}

    trans_prob = defaultdict(dict)
    for t1 in all_tags:
        total = sum(trans_counts[t1].values())
        for t2 in all_tags:
            trans_prob[t1][t2] = trans_counts[t1][t2]/total if total > 0 else 0

    def get_emission(tag, word):
        return emit_counts[tag].get(word, 0) / tag_counts[tag] if tag_counts[tag] > 0 else 0

    # 5. Viterbi Algorithm
    def viterbi(obs, states, start_p, trans_p, emit_func):
        V = [{}]
        path = {}

        # Initialize
        for y in states:
            V[0][y] = start_p.get(y, 1e-6) * (emit_func(y, obs[0]) + 1e-6)
            path[y] = [y]

        # Run
        for t in range(1, len(obs)):
            V.append({})
            newpath = {}
        for y in states:
            (prob, state) = max(
                (V[t-1][y0] * (trans_p[y0].get(y, 0) + 1e-6) * (emit_func(y, obs[t]) + 1e-6), y0)
                for y0 in states
            )
            V[t][y] = prob
            newpath[y] = path[state] + [y]
        path = newpath

    (prob, state) = max((V[len(obs)-1][y], y) for y in states)
    return path[state]

# Run Test
test_tweet = tagged_tweets[0][:5]
tokens = [t[0].lower() for t in test_tweet]
print(f"Test Tokens: {tokens}")
print(f"Predicted:   {viterbi(tokens, list(all_tags), start_prob, trans_prob, get_emission)}")
```

```
Test Tokens: ['party', 'least', 'receive', 'say', 'or']
Predicted:   ['NOUN', 'NOUN', 'NOUN', 'NOUN', 'CONJ']
```