

```

from nltk.probability import ConditionalFreqDist, FreqDist

# Initialize frequency distributions
initial_tag_fd = FreqDist() # Frequency of the first tag in a tweet
transition_fd = ConditionalFreqDist() # Frequency of a tag given the previous tag
emission_fd = ConditionalFreqDist() # Frequency of a word given its tag

# Process each tweet's POS tags
for tags_list in df['pos_tags']:
    if not tags_list: # Skip empty tag lists
        continue

    # Initial tag frequency
    initial_tag_fd[tags_list[0][1]] += 1

    for i, (word, tag) in enumerate(tags_list):
        # Emission frequency
        emission_fd[tag][word] += 1

        # Transition frequency
        if i > 0:
            prev_tag = tags_list[i-1][1]
            transition_fd[prev_tag][tag] += 1

# Convert frequencies to probabilities (using Maximum Likelihood Estimation)
# This is a simplification; for a robust HMM, smoothing (e.g., Laplace) is often used

# Initial probabilities: P(Tag_i at start)
initial_prob = {tag: initial_tag_fd[tag] / initial_tag_fd.N() for tag in initial_tag_fd.keys()}

# Transition probabilities: P(Tag_j | Tag_i)
transition_prob = {}
for prev_tag in transition_fd:
    total_transitions = transition_fd[prev_tag].N()
    if total_transitions > 0:
        transition_prob[prev_tag] = {tag: transition_fd[prev_tag][tag] / total_transitions for tag in transition_fd[prev_tag].keys()}
    else:
        transition_prob[prev_tag] = {}

# Emission probabilities: P(Word_k | Tag_j)
emission_prob = {}
for tag in emission_fd:
    total_emissions = emission_fd[tag].N()
    if total_emissions > 0:
        emission_prob[tag] = {word: emission_fd[tag][word] / total_emissions for word in emission_fd[tag].keys()}

```

```

else:
    emission_prob[tag] = {}

print("Calculated HMM Parameters (first 5 for each type):")
print("\nInitial Probabilities:", {k: v for k, v in list(initial_prob.items())})
print("\nTransition Probabilities (first 5 previous tags):")
for i, (prev_tag, next_tags) in enumerate(transition_prob.items()):
    if i >= 5: break
    print(f" {prev_tag}:", {k: v for k, v in list(next_tags.items())[:5]})
print("\nEmission Probabilities (first 5 tags):")
for i, (tag, words) in enumerate(emission_prob.items()):
    if i >= 5: break
    print(f" {tag}:", {k: v for k, v in list(words.items())[:5]})

```

Calculated HMM Parameters (first 5 for each type):

Initial Probabilities: {'NN': 0.3353867676203683, 'JJ': 0.13077144908239713, 'RB

Transition Probabilities (first 5 previous tags):

WRB: {'NN': 0.2631164484693375, 'JJ': 0.18554975883453095, 'PRP': 0.0805853594
 NN: {'NN': 0.3595183710397132, 'IN': 0.09323072283437303, 'VBD': 0.06080342233
 VBD: {'JJ': 0.1855463656346634, 'NN': 0.15572258857717672, 'DT': 0.11564397749
 NNP: {'NN': 0.4392312711311466, 'NNP': 0.20932439646479625, 'JJ': 0.0843466397
 JJ: {'NN': 0.5665254923118425, 'NNS': 0.15297814944699217, 'JJ': 0.09950633935

Emission Probabilities (first 5 tags):

WRB: {'why': 0.3527957583840224, 'when': 0.262271086100901, 'how': 0.251471879
 NN: {'modi': 0.10320215229946092, 'india': 0.016277776092079577, 'congress': 0
 VBD: {'was': 0.14797588285960378, 'did': 0.06361122444353778, 'had': 0.0479441
 NNP: {'': 0.4527071102413568, '': 0.06582458637253158, '': 0.05538753483958
 JJ: {'modi': 0.04614891259984841, 'indian': 0.017589750928914825, 'india': 0.0

```

import nltk
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag

# Download necessary NLTK data (if not already downloaded)
try:
    nltk.data.find('tokenizers/punkt')
except LookupError:
    nltk.download('punkt')
try:
    nltk.data.find('taggers/averaged_perceptron_tagger_eng')
except LookupError:
    nltk.download('averaged_perceptron_tagger_eng')
try:
    nltk.data.find('tokenizers/punkt_tab')
except LookupError:
    nltk.download('punkt_tab')

# Function to perform POS tagging
def get_pos_tags(text):

```

```

tokens = word_tokenize(text)
return pos_tag(tokens)

# Apply POS tagging to the 'clean_text_cleaned' column
df['pos_tags'] = df['clean_text_cleaned'].apply(get_pos_tags)

# Display the DataFrame with the new POS tags column
display(df[['clean_text_cleaned', 'pos_tags']].head())

```

```

[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data] /root/nltk_data...
[nltk_data] Unzipping taggers/averaged_perceptron_tagger_eng.zip.

```

	clean_text_cleaned	pos_tags
0	when modi promised “minimum government maximum...	[(when, WRB), (modi, NN), (promised, VBD), (“,...
1	talk all the nonsense and continue all the dra...	[(talk, NN), (all, PDT), (the, DT), (nonsense,...
2	what did just say vote for modi welcome bjp t...	[(what, WP), (did, VBD), (just, RB), (say, VB)...
3	asking his supporters prefix chowkidar their n...	[(asking, VBG), (his, PRP\$), (supporters, NNS)...
4	answer who among these the most powerful world...	[(answer, NN), (who, WP), (among, IN), (these,...

```

import re

def remove_urls(text):
    # Remove URLs (http/https) from the text
    return re.sub(r'http\S+|www\S+', '', text)

def remove_mentions(text):
    # Remove mentions (@username) from the text
    return re.sub(r'@\w+', '', text)

```

```

# Apply the cleaning functions to the 'clean_text' column
df['clean_text_cleaned'] = df['clean_text'].astype(str).apply(remove_urls)
df['clean_text_cleaned'] = df['clean_text_cleaned'].apply(remove_mentions)

# Display the DataFrame with the new cleaned column
display(df[['clean_text', 'clean_text_cleaned']].head())

```

	clean_text	clean_text_cleaned
0	when modi promised "minimum government maximum...	when modi promised "minimum government maximum...
1	talk all the nonsense and continue all the dra...	talk all the nonsense and continue all the dra...
2	what did just say vote for modi welcome bjp t...	what did just say vote for modi welcome bjp t...
3	asking his supporters prefix chowkidar their n...	asking his supporters prefix chowkidar their n...
4	answer who among these the most powerful world...	answer who among these the most powerful world...

```
import pandas as pd
```

```
df = pd.read_csv('/content/Twitter_Data.csv')
display(df.head())
```

	clean_text	category
0	when modi promised "minimum government maximum...	-1.0
1	talk all the nonsense and continue all the dra...	0.0
2	what did just say vote for modi welcome bjp t...	1.0
3	asking his supporters prefix chowkidar their n...	1.0
4	answer who among these the most powerful world...	1.0

```
print(f"Total unique previous tags in transition_prob: {len(transition_prob)}")
print("\nExample transition probabilities for the first 3 previous tags:")
for i, (prev_tag, next_tags) in enumerate(transition_prob.items()):
    if i >= 3: break
    print(f" {prev_tag}: {next_tags}")
```

Total unique previous tags in transition_prob: 38

Example transition probabilities for the first 3 previous tags:

```
WRB: {'NN': 0.2631164484693375, 'JJ': 0.18554975883453095, 'PRP': 0.0805853594
NN: {'NN': 0.3595183710397132, 'IN': 0.09323072283437303, 'VBD': 0.06080342233
VBD: {'JJ': 0.1855463656346634, 'NN': 0.15572258857717672, 'DT': 0.11564397749
```

```
print("\n--- Analysis of Transition Probabilities ---\n")
```

```
high_prob_transitions = []
narrow_distribution_tags = []
```

```

broad_distribution_tags = []

for prev_tag, next_tags in transition_prob.items():
    if not next_tags: # Skip if no transitions from this tag (shouldn't happen)
        continue

    # Identify highest probability transition for the current prev_tag
    max_prob_tag = max(next_tags, key=next_tags.get)
    max_prob_value = next_tags[max_prob_tag]

    if max_prob_value > 0.9: # Threshold for unusually high probability
        high_prob_transitions.append(f" '{prev_tag}' -> '{max_prob_tag}' with

    # Check for narrow/broad distributions
    num_next_tags = len(next_tags)
    if num_next_tags <= 2: # Very narrow distribution
        narrow_distribution_tags.append(f" '{prev_tag}' has only {num_next_tag
    elif num_next_tags >= 20: # Very broad distribution (arbitrary threshold, a
        broad_distribution_tags.append(f" '{prev_tag}' has {num_next_tags} uni

print("\n1. Tags with unusually high transition probabilities (e.g., > 0.9):")
if high_prob_transitions:
    for transition in high_prob_transitions:
        print(transition)
else:
    print(" No transitions with probability > 0.9 found.")

print("\n2. Tags exhibiting very narrow distributions (2 or fewer unique next t
if narrow_distribution_tags:
    for tag_info in narrow_distribution_tags:
        print(tag_info)
else:
    print(" No tags with very narrow distributions found.")

print("\n3. Tags exhibiting very broad distributions (20 or more unique next ta
if broad_distribution_tags:
    for tag_info in broad_distribution_tags:
        print(tag_info)
else:
    print(" No tags with very broad distributions found.")

# Also explicitly look for any tags that have 100% certainty to be followed by
perfect_transitions = []
for prev_tag, next_tags in transition_prob.items():
    if len(next_tags) == 1:
        next_tag, prob = list(next_tags.items())[0]
        perfect_transitions.append(f" '{prev_tag}' *always* followed by '{next

if perfect_transitions:
    print("\n4. Tags with 100% certainty transitions (only one possible next ta
    for transition in perfect_transitions:

```

```

        print(transition)
    else:
        print("\n4. No tags found with 100% certainty transitions.")

```

--- Analysis of Transition Probabilities ---

- Tags with unusually high transition probabilities (e.g., > 0.9):
 - 'PDT' -> 'DT' with $P=0.9427$
 - '\$' -> 'CD' with $P=0.9956$
 - 'SYM' -> 'NN' with $P=1.0000$
 - `` -> 'RB' with $P=1.0000$
- Tags exhibiting very narrow distributions (2 or fewer unique next tags):
 - '\$' has only 2 unique next tags.
 - 'SYM' has only 1 unique next tags.
 - `` has only 1 unique next tags.
- Tags exhibiting very broad distributions (20 or more unique next tags):
 - 'WRB' has 31 unique next tags.
 - 'NN' has 37 unique next tags.
 - 'VBD' has 35 unique next tags.
 - 'NNP' has 31 unique next tags.
 - 'JJ' has 36 unique next tags.
 - 'PRP' has 32 unique next tags.
 - 'VB' has 35 unique next tags.
 - 'DT' has 32 unique next tags.
 - 'VBG' has 33 unique next tags.
 - 'VBZ' has 34 unique next tags.
 - 'NNS' has 35 unique next tags.
 - 'MD' has 27 unique next tags.
 - 'CC' has 32 unique next tags.
 - 'RB' has 33 unique next tags.
 - 'IN' has 31 unique next tags.
 - 'WP' has 30 unique next tags.
 - 'VBP' has 34 unique next tags.
 - 'PRP\$' has 30 unique next tags.
 - 'EX' has 26 unique next tags.
 - 'JJS' has 27 unique next tags.
 - 'RBS' has 25 unique next tags.
 - 'WDT' has 29 unique next tags.
 - 'VBN' has 35 unique next tags.
 - 'JJR' has 28 unique next tags.
 - 'RBR' has 29 unique next tags.
 - 'CD' has 35 unique next tags.
 - 'FW' has 29 unique next tags.
 - 'RP' has 31 unique next tags.
- Tags with 100% certainty transitions (only one possible next tag):
 - 'SYM' **always** followed by 'NN' ($P=1.0000$)
 - `` **always** followed by 'RB' ($P=1.0000$)

```

print(f"Total unique tags with emission probabilities: {len(emission_prob)}")
print("\nTop 5 words with emission probabilities for the first 5 unique tags:")

```

```

for i, (tag, words) in enumerate(emission_prob.items()):
    if i >= 5: # Limit to first 5 tags
        break
    print(f" Tag: '{tag}'")
    # Sort words by probability in descending order and take the top 5
    sorted_words = sorted(words.items(), key=lambda item: item[1], reverse=True)
    for j, (word, prob) in enumerate(sorted_words):
        if j >= 5: # Limit to top 5 words per tag
            break
        print(f"      Word: '{word}', Probability: {prob:.4f}")

```

Total unique tags with emission probabilities: 38

Top 5 words with emission probabilities for the first 5 unique tags:

```

Tag: 'WRB'
  Word: 'why', Probability: 0.3528
  Word: 'when', Probability: 0.2623
  Word: 'how', Probability: 0.2515
  Word: 'where', Probability: 0.1225
  Word: 'whenever', Probability: 0.0053
Tag: 'NN'
  Word: 'modi', Probability: 0.1032
  Word: 'india', Probability: 0.0163
  Word: 'congress', Probability: 0.0108
  Word: 'bjp', Probability: 0.0102
  Word: 'country', Probability: 0.0079
Tag: 'VBD'
  Word: 'was', Probability: 0.1480
  Word: 'did', Probability: 0.0636
  Word: 'had', Probability: 0.0479
  Word: 'said', Probability: 0.0451
  Word: 'were', Probability: 0.0403
Tag: 'NNP'
  Word: "'", Probability: 0.4527
  Word: '"', Probability: 0.0658
  Word: '"', Probability: 0.0554
  Word: '"', Probability: 0.0507
  Word: 'o', Probability: 0.0192
Tag: 'JJ'
  Word: 'modi', Probability: 0.0461
  Word: 'indian', Probability: 0.0176
  Word: 'india', Probability: 0.0149
  Word: 'narendra', Probability: 0.0144
  Word: 'good', Probability: 0.0134

```

```

print("\n--- Words with very low emission probabilities (P < 0.0001) ---")
low_prob_emissions = []
threshold = 0.0001

for tag, words_probs in emission_prob.items():
    for word, prob in words_probs.items():
        if prob < threshold:
            low_prob_emissions.append((word, tag, prob))

```

```
# Sort by probability for better readability and take a few examples
low_prob_emissions.sort(key=lambda x: x[2])

if low_prob_emissions:
    print(f"Found {len(low_prob_emissions)} word-tag pairs with emission probab
    for i, (word, tag, prob) in enumerate(low_prob_emissions):
        if i >= 10: break
        print(f" Word: '{word}', Tag: '{tag}', Probability: {prob:.6f}")
else:
    print(" No word-tag pairs found with emission probability below the thresh
```

```
--- Words with very low emission probabilities (P < 0.0001) ---
Found 152332 word-tag pairs with emission probability below 0.0001. Here are the
Word: 'constituency2', Tag: 'NN', Probability: 0.000001
Word: 'tuthukudi', Tag: 'NN', Probability: 0.000001
Word: 'thuthukudi', Tag: 'NN', Probability: 0.000001
Word: 'leadershipwho', Tag: 'NN', Probability: 0.000001
Word: 'modiganga', Tag: 'NN', Probability: 0.000001
Word: 'repressive', Tag: 'NN', Probability: 0.000001
Word: 'ministerdisgrace', Tag: 'NN', Probability: 0.000001
Word: 'archery', Tag: 'NN', Probability: 0.000001
Word: 'idu', Tag: 'NN', Probability: 0.000001
Word: 'bekagittu', Tag: 'NN', Probability: 0.000001
```

```
print("\n--- Infrequently occurring words (total count < 5) ---")
word_counts = {}
for tag in emission_fd:
    for word in emission_fd[tag]:
        word_counts[word] = word_counts.get(word, 0) + emission_fd[tag][word]

infrequent_words = []
count_threshold = 5

for word, count in word_counts.items():
    if count < count_threshold:
        infrequent_words.append((word, count))

# Sort by count for better readability and take a few examples
infrequent_words.sort(key=lambda x: x[1])

if infrequent_words:
    print(f"Found {len(infrequent_words)} words with total occurrences below {c
    for i, (word, count) in enumerate(infrequent_words):
        if i >= 10: break
        print(f" Word: '{word}', Total Count: {count}")
else:
    print(" No words found with total occurrences below the threshold.")
```



```

--- Infrequently occurring words (total count < 5) ---
Found 88234 words with total occurrences below 5. Here are the first 10 examples
Word: 'wiselythe', Total Count: 1
Word: 'withot', Total Count: 1
Word: 'jaiwere', Total Count: 1
Word: 'kongujratwhere', Total Count: 1
Word: 'workersthere', Total Count: 1
Word: 'whatbieber', Total Count: 1
Word: 'wrongthen', Total Count: 1
Word: 'withoue', Total Count: 1
Word: 'wrz✔promised✔jobs', Total Count: 1
Word: 'wadhai', Total Count: 1

```

```

selected_phrase = "modi power"
tokens = selected_phrase.split()
print(f"Selected phrase: '{selected_phrase}'")
print(f"Tokenized phrase: {tokens}")

```

```

Selected phrase: 'modi power'
Tokenized phrase: ['modi', 'power']

```

```

all_tags = list(initial_prob.keys()) # All possible tags

viterbi_table = [{}] # Stores scores for each step
backpointer_table = [{}] # Stores backpointers for path reconstruction

first_word = tokens[0]
print(f"\n--- Viterbi Initialization for '{first_word}' ---\n")

for tag in all_tags:
    initial_p = initial_prob.get(tag, 0) # P(Tag_i at start)
    emission_p = emission_prob.get(tag, {}).get(first_word, 0) # P(Word_k | Tag

    if initial_p > 0 and emission_p > 0:
        score = initial_p * emission_p
        viterbi_table[0][tag] = score
        # No backpointer for the first word
        backpointer_table[0][tag] = None
        print(f" Tag: {tag}, Initial Prob: {initial_p:.6f}, Emission Prob('{fi

print(f"\nScores for '{first_word}': {viterbi_table[0]}")

```

```

--- Viterbi Initialization for 'modi' ---

```

```

Tag: NN, Initial Prob: 0.335387, Emission Prob('modi'|'NN'): 0.103202, Score:
Tag: JJ, Initial Prob: 0.130771, Emission Prob('modi'|'JJ'): 0.046149, Score:
Tag: RB, Initial Prob: 0.098399, Emission Prob('modi'|'RB'): 0.013175, Score:
Tag: NNS, Initial Prob: 0.069665, Emission Prob('modi'|'NNS'): 0.047367, Score:
Tag: VB, Initial Prob: 0.042711, Emission Prob('modi'|'VB'): 0.042159, Score:

```

```

Tag: IN, Initial Prob: 0.033661, Emission Prob('modi'|'IN'): 0.002181, Score:
Tag: WRB, Initial Prob: 0.030844, Emission Prob('modi'|'WRB'): 0.000033, Score:
Tag: PRP, Initial Prob: 0.028642, Emission Prob('modi'|'PRP'): 0.007099, Score:
Tag: VBG, Initial Prob: 0.019739, Emission Prob('modi'|'VBG'): 0.000044, Score:
Tag: MD, Initial Prob: 0.018591, Emission Prob('modi'|'MD'): 0.006102, Score:
Tag: WP, Initial Prob: 0.017518, Emission Prob('modi'|'WP'): 0.001031, Score:
Tag: CC, Initial Prob: 0.016450, Emission Prob('modi'|'CC'): 0.001110, Score:
Tag: CD, Initial Prob: 0.014137, Emission Prob('modi'|'CD'): 0.007211, Score:
Tag: VBD, Initial Prob: 0.012652, Emission Prob('modi'|'VBD'): 0.017553, Score:
Tag: VBN, Initial Prob: 0.012204, Emission Prob('modi'|'VBN'): 0.009745, Score:
Tag: VBZ, Initial Prob: 0.008388, Emission Prob('modi'|'VBZ'): 0.033028, Score:
Tag: VBP, Initial Prob: 0.006651, Emission Prob('modi'|'VBP'): 0.046309, Score:
Tag: JJS, Initial Prob: 0.003442, Emission Prob('modi'|'JJS'): 0.004163, Score:
Tag: WDT, Initial Prob: 0.002878, Emission Prob('modi'|'WDT'): 0.001665, Score:
Tag: EX, Initial Prob: 0.002534, Emission Prob('modi'|'EX'): 0.000491, Score:
Tag: PDT, Initial Prob: 0.001902, Emission Prob('modi'|'PDT'): 0.066915, Score:
Tag: RBR, Initial Prob: 0.001608, Emission Prob('modi'|'RBR'): 0.059779, Score:
Tag: JJR, Initial Prob: 0.000742, Emission Prob('modi'|'JJR'): 0.006814, Score:
Tag: RBS, Initial Prob: 0.000675, Emission Prob('modi'|'RBS'): 0.207652, Score:
Tag: UH, Initial Prob: 0.000577, Emission Prob('modi'|'UH'): 0.004237, Score:
Tag: FW, Initial Prob: 0.000092, Emission Prob('modi'|'FW'): 0.276727, Score:
Tag: NNP, Initial Prob: 0.000031, Emission Prob('modi'|'NNP'): 0.007946, Score:

```

Scores for 'modi': {'NN': 0.034612636271181156, 'JJ': 0.006034960174259071, 'RB':

```

viterbi_table.append({}) # Scores for the current word
backpointer_table.append({}) # Backpointers for the current word

current_word_idx = 1
current_word = tokens[current_word_idx]

print(f"\n--- Viterbi Recursion for '{current_word}' ---\n")

for current_tag in all_tags:
    max_score_for_current_tag = 0
    best_prev_tag_for_current_tag = None

    # Emission probability for the current word given the current tag
    emission_p_current = emission_prob.get(current_tag, {}).get(current_word, 0)

    if emission_p_current == 0:
        continue # If word cannot be emitted by this tag, no need to calculate

    print(f"  Considering current_tag: {current_tag}")

    for prev_tag, prev_score in viterbi_table[current_word_idx - 1].items():
        # Transition probability from previous tag to current tag
        transition_p = transition_prob.get(prev_tag, {}).get(current_tag, 0)

        # Calculate path score
        path_score = prev_score * transition_p * emission_p_current

```

```

    if path_score > max_score_for_current_tag:
        max_score_for_current_tag = path_score
        best_prev_tag_for_current_tag = prev_tag

    if path_score > 0: # Only print non-zero paths
        print(f"    Path: {prev_tag} -> {current_tag}, Prev Score: {prev_score}")

    if max_score_for_current_tag > 0:
        viterbi_table[current_word_idx][current_tag] = max_score_for_current_tag
        backpointer_table[current_word_idx][current_tag] = best_prev_tag_for_current_tag
        print(f"    Max Score for '{current_tag}': {max_score_for_current_tag:.5f}")

    print(f"Scores for '{current_word}': {viterbi_table[current_word_idx]}")
    print(f"Backpointers for '{current_word}': {backpointer_table[current_word_idx]}")

```

iterbi Recursion for 'power' ---

```

sidering current_tag: NN
ath: NN -> NN, Prev Score: 0.034613, Transition P: 0.359518, Emission P('power'|
ath: JJ -> NN, Prev Score: 0.006035, Transition P: 0.566525, Emission P('power'|
ath: RB -> NN, Prev Score: 0.001296, Transition P: 0.067306, Emission P('power'|
ath: NNS -> NN, Prev Score: 0.003300, Transition P: 0.076584, Emission P('power'|
ath: VB -> NN, Prev Score: 0.001801, Transition P: 0.190560, Emission P('power'|
ath: IN -> NN, Prev Score: 0.000073, Transition P: 0.321770, Emission P('power'|
ath: WRB -> NN, Prev Score: 0.000001, Transition P: 0.263116, Emission P('power'|
ath: PRP -> NN, Prev Score: 0.000203, Transition P: 0.024407, Emission P('power'|
ath: VBG -> NN, Prev Score: 0.000001, Transition P: 0.257260, Emission P('power'|
ath: MD -> NN, Prev Score: 0.000113, Transition P: 0.018673, Emission P('power'|
ath: WP -> NN, Prev Score: 0.000018, Transition P: 0.109971, Emission P('power'|
ath: CC -> NN, Prev Score: 0.000018, Transition P: 0.237532, Emission P('power'|
ath: CD -> NN, Prev Score: 0.000102, Transition P: 0.366125, Emission P('power'|
ath: VBD -> NN, Prev Score: 0.000222, Transition P: 0.155723, Emission P('power'|
ath: VBN -> NN, Prev Score: 0.000119, Transition P: 0.209336, Emission P('power'|
ath: VBZ -> NN, Prev Score: 0.000277, Transition P: 0.140420, Emission P('power'|
ath: VBP -> NN, Prev Score: 0.000308, Transition P: 0.145603, Emission P('power'|
ath: JJS -> NN, Prev Score: 0.000014, Transition P: 0.502695, Emission P('power'|
ath: WDT -> NN, Prev Score: 0.000005, Transition P: 0.111150, Emission P('power'|
ath: EX -> NN, Prev Score: 0.000001, Transition P: 0.178723, Emission P('power'|
ath: PDT -> NN, Prev Score: 0.000127, Transition P: 0.000981, Emission P('power'|
ath: RBR -> NN, Prev Score: 0.000096, Transition P: 0.136520, Emission P('power'|
ath: JJR -> NN, Prev Score: 0.000005, Transition P: 0.330635, Emission P('power'|
ath: RBS -> NN, Prev Score: 0.000140, Transition P: 0.054429, Emission P('power'|
ath: UH -> NN, Prev Score: 0.000002, Transition P: 0.411017, Emission P('power'|
ath: FW -> NN, Prev Score: 0.000025, Transition P: 0.495081, Emission P('power'|
ath: NNP -> NN, Prev Score: 0.000000, Transition P: 0.439231, Emission P('power'|
ax Score for 'NN': 0.000071451 (from NN)

s for 'power': {'NN': 7.145057586828129e-05}
ointers for 'power': {'NN': 'NN'}

```

```

print(
    "\n--- Viterbi Termination and Backtracking ---\n")

```

```
# Find the overall best score for the last word
last_word_idx = len(tokens) - 1
final_scores = viterbi_table[last_word_idx]

if not final_scores:
    print("No possible tag sequences found for the given phrase.")
    predicted_tags = []
else:
    best_last_tag = max(final_scores, key=final_scores.get)
    max_final_score = final_scores[best_last_tag]

    print(f"Highest probability for the last word ('{tokens[last_word_idx]}'):

    # Backtrack to find the full sequence of tags
    predicted_tags = [best_last_tag]
    for i in range(last_word_idx, 0, -1):
        prev_tag = backpointer_table[i][predicted_tags[0]]
        predicted_tags.insert(0, prev_tag)

    print(f"Most likely POS tag sequence for '{' '.join(tokens)}': {predicted_tags}
```

--- Viterbi Termination and Backtracking ---

Highest probability for the last word ('power'): 0.000071451 with tag 'NN'

Most likely POS tag sequence for 'modi power': ['NN', 'NN']