

```
import pandas as pd
df=pd.read_csv('/content/Twitter_Data (1).csv')
display(df.head())
```

```
import re

def remove_urls(text):
    url_pattern = re.compile(r'https?://\S+|www\.\S+')
    return url_pattern.sub(r'', str(text))

df['text_no_urls'] = df['clean_text'].apply(remove_urls)
display(df[['clean_text', 'text_no_urls']].head())
```

```
def remove_mentions(text):
    mention_pattern = re.compile(r'@\w+')
    return mention_pattern.sub(r'', str(text))

df['text_cleaned'] = df['text_no_urls'].apply(remove_mentions)
display(df[['clean_text', 'text_no_urls', 'text_cleaned']].head())
```

```
!pip install nltk

Requirement already satisfied: nltk in /usr/local/lib/python3.12/dist-packages (3.9.1)
Requirement already satisfied: click in /usr/local/lib/python3.12/dist-packages (from nltk) (8.3.1)
Requirement already satisfied: joblib in /usr/local/lib/python3.12/dist-packages (from nltk) (1.5.3)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.12/dist-packages (from nltk) (2025.11.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from nltk) (4.67.1)
```

```
import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('punkt_tab')
nltk.download('averaged_perceptron_tagger_eng')
```

```
[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 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.
True
```

```
from nltk.tokenize import word_tokenize

def get_tokens(text):
    if isinstance(text, str):
        return word_tokenize(text)
    return []

df['text_tokens'] = df['text_cleaned'].apply(get_tokens)
display(df[['text_cleaned', 'text_tokens']].head())
```

```
import nltk
from collections import defaultdict, Counter

# Download necessary NLTK data
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('punkt_tab')
nltk.download('averaged_perceptron_tagger_eng')

# Drop rows with NaN in clean_text
df = df.dropna(subset=['clean_text'])

# Use a subset for efficiency
subset_df = df.head(5000).copy()

# Tokenize and POS tag
def pos_tag_tweet(text):
    tokens = nltk.word_tokenize(str(text))
    return nltk.pos_tag(tokens)

subset_df['tagged_tweets'] = subset_df['clean_text'].apply(pos_tag_tweet)

# Initialize counters for HMM parameters
transitions = defaultdict(Counter)
emissions = defaultdict(Counter)
initial_states = Counter()

# Count occurrences
for tags in subset_df['tagged_tweets']:
    if not tags:
        continue

    # Initial state
    initial_states[tags[0][1]] += 1

    for i in range(len(tags)):
        word, tag = tags[i]
        emissions[tag][word] += 1

    if i > 0:
        prev_tag = tags[i-1][1]
        transitions[prev_tag][tag] += 1
```

```
import os
import nltk

print("NLTK Data Path:", nltk.data.path)
for path in nltk.data.path:
    if os.path.exists(path):
        print(f"Contents of {path}:", os.listdir(path))
    else:
        print(f"{path} does not exist.")

NLTK Data Path: ['/root/nltk_data', '/usr/nltk_data', '/usr/share/nltk_data', '/usr/lib/nltk_data', '/usr/share/nltk_data',
Contents of /root/nltk_data: ['taggers', 'tokenizers']
/usr/nltk_data does not exist.
/usr/share/nltk_data does not exist.
/usr/lib/nltk_data does not exist.
/usr/share/nltk_data does not exist.
/usr/local/share/nltk_data does not exist.
/usr/lib/nltk_data does not exist.
/usr/local/lib/nltk_data does not exist.
```

```
try:  
    import spacy  
    nlp = spacy.load("en_core_web_sm")  
    print("Spacy is available and model is loaded.")  
except Exception as e:  
    print("Spacy error:", e)
```

Spacy is available and model is loaded.

```
import pandas as pd
from collections import defaultdict, Counter
import json

# Load the dataset
df = pd.read_csv('Twitter Data (1).csv').dropna(subset=['clean text'])
```

```

subset_df = df.head(1000).copy()

# Simple Heuristic POS Tagger since NLTK data is unavailable
def heuristic_tagger(word):
    word = str(word).lower()
    if word in ['the', 'a', 'an', 'this', 'that', 'these', 'those']: return 'DT'
    if word in ['i', 'you', 'he', 'she', 'it', 'we', 'they', 'me', 'him', 'her', 'us', 'them', 'my', 'your', 'his', 'their']:
        if word in ['is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'do', 'does', 'did', 'go', 'get']:
            if word in ['in', 'on', 'at', 'to', 'for', 'with', 'by', 'from', 'of', 'about', 'up', 'down', 'out', 'over', 'under']:
                if word in ['and', 'or', 'but', 'so', 'yet']: return 'CC'
                if word in ['not', 'very', 'now', 'then', 'here', 'there']: return 'RB'
                if word.endswith('ly'): return 'RB'
                if word.endswith('ing'): return 'VBD'
                if word.endswith('ed'): return 'VBD'
                if word.endswith('s') and len(word) > 3: return 'NNS'
                if word in ['good', 'bad', 'great', 'best', 'worst', 'big', 'small', 'new', 'old']: return 'JJ'
            return 'NN' # Default to Noun

    def process_tweet(text):
        tokens = str(text).split()
        return [(token, heuristic_tagger(token)) for token in tokens]

    # Tag the tweets
    subset_df['tagged'] = subset_df['clean_text'].apply(process_tweet)

    # Build HMM Parameters
    transitions = defaultdict(Counter)
    emissions = defaultdict(Counter)
    starts = Counter()

    for sequence in subset_df['tagged']:
        if not sequence:
            continue

        # Start probabilities
        starts[sequence[0][1]] += 1

        for i in range(len(sequence)):
            word, tag = sequence[i]
            emissions[tag][word] += 1

            if i > 0:
                prev_tag = sequence[i-1][1]
                transitions[prev_tag][tag] += 1

    # Normalize probabilities
    def normalize(counts_dict):
        result = {}
        for state, counts in counts_dict.items():
            total = sum(counts.values())
            result[state] = {k: v / total for k, v in counts.items()}
        return result

    transition_probs = normalize(transitions)
    emission_probs = normalize(emissions)
    total_starts = sum(starts.values())
    start_probs = {k: v / total_starts for k, v in starts.items()}

    # Display some parameters
    print("HMM Parameters Summary:")
    print(f"Total States: {len(emission_probs)}")
    print(f"Total Words in subset: {sum(len(s) for s in subset_df['tagged'])}")

    # Save a sample of the parameters to a file for the user
    hmm_params = {
        "start_probabilities": start_probs,
        "transition_probabilities": {k: dict(Counter(v).most_common(5)) for k, v in transition_probs.items()},
        "emission_probabilities": {k: dict(Counter(v).most_common(5)) for k, v in emission_probs.items()}
    }

    with open('hmm_parameters.json', 'w') as f:
        json.dump(hmm_params, f, indent=4)

    print("\nSample Transition Probabilities (top 5 for first 3 states):")
    for state in list(transition_probs.keys())[:3]:
        print(f"{state}: {dict(Counter(transition_probs[state]).most_common(5))}")

    print("\nSample Emission Probabilities (top 5 for first 3 states):")
    for state in list(emission_probs.keys())[:3]:
        print(f"{state}: {dict(Counter(emission_probs[state]).most_common(5))}")

```

```

HMM Parameters Summary:
Total States: 11
Total Words in subset: 20411

Sample Transition Probabilities (top 5 for first 3 states):
NN: {'NN': 0.6178895994614608, 'NNS': 0.0994614607876136, 'DT': 0.0487209693705823, 'VB': 0.04594412655671491, 'IN': 0.03876
VBD: {'NN': 0.5661016949152542, 'NNS': 0.09661016949152543, 'DT': 0.08305084745762711, 'IN': 0.08135593220338982, 'CC': 0.04
PRP: {'NN': 0.6260296540362438, 'VB': 0.1696869851729819, 'NNS': 0.09555189456342669, 'RB': 0.02471169686985173, 'VBD': 0.02

Sample Emission Probabilities (top 5 for first 3 states):
NN: {'modi': 0.07103306438854612, 'india': 0.013921847808890999, 'bjp': 0.009729473184622686, 'all': 0.008622053472551811, 'VBD: {'need': 0.031198686371100164, 'called': 0.019704433497536946, 'used': 0.0180623973727422, 'created': 0.014778325123152
PRP: {'you': 0.31290322580645163, 'his': 0.18225806451612903, 'they': 0.12741935483870967, 'your': 0.11774193548387096, 'him

```

```

import pandas as pd
import numpy as np
from collections import defaultdict, Counter
import json

# 1. Load and Prepare Data
def load_and_clean_data(file_path, num_rows=1000):
    df = pd.read_csv(file_path).dropna(subset=['clean_text'])
    return df.head(num_rows).copy()

# 2. Heuristic POS Tagger
# Note: Using a heuristic tagger because NLTK pre-trained data is often
# unavailable in restricted environments.
def heuristic_tagger(word):
    word = str(word).lower()
    if word in ['the', 'a', 'an', 'this', 'that']: return 'DT'
    if word in ['i', 'you', 'he', 'she', 'it', 'we', 'they', 'me', 'him', 'her']: return 'PRP'
    if word in ['is', 'are', 'was', 'were', 'be', 'have', 'has', 'do', 'can', 'will']: return 'VB'
    if word in ['in', 'on', 'at', 'to', 'for', 'with', 'by', 'of', 'from']: return 'IN'
    if word in ['and', 'on', 'but', 'so']: return 'CC'
    if word.endswith('ly'): return 'RB'
    if word.endswith('ing'): return 'VBG'
    if word.endswith('ed'): return 'VBD'
    if word.endswith('s') and len(word) > 3: return 'NNS'
    return 'NN' # Default to Noun

# 3. Build HMM Parameters with Smoothing
def build_hmm_parameters(df, alpha=1.0):
    tags = ['DT', 'PRP', 'VB', 'IN', 'CC', 'RB', 'VBG', 'VBD', 'NNS', 'NN']
    tag_to_idx = {tag: i for i, tag in enumerate(tags)}

    # Tag the corpus
    df['tagged'] = df['clean_text'].apply(lambda x: [(w, heuristic_tagger(w)) for w in str(x).split()])

    # Initialize counts
    trans_counts = np.zeros((len(tags), len(tags)))
    emiss_counts = defaultdict(Counter)
    start_counts = np.zeros(len(tags))
    vocab = set()

    for sequence in df['tagged']:
        if not sequence: continue

        # Initial state counts
        start_counts[tag_to_idx[sequence[0][1]]] += 1

        for i in range(len(sequence)):
            word, tag = sequence[i]
            vocab.add(word)
            emiss_counts[tag][word] += 1

            # Transition counts
            if i > 0:
                prev_tag = sequence[i-1][1]
                trans_counts[tag_to_idx[prev_tag]][tag_to_idx[tag]] += 1

    # Apply Laplace Smoothing to handle irregularities (zero probabilities)
    # Transition probabilities: P(tag_j | tag_i)
    trans_probs = (trans_counts + alpha) / (trans_counts.sum(axis=1)[:, None] + alpha * len(tags))

    # Initial state probabilities: P(tag_i | start)
    start_probs = (start_counts + alpha) / (start_counts.sum() + alpha * len(tags))

    return trans_probs, emiss_counts, start_probs, tags, tag_to_idx, vocab

# 4. Viterbi Decoding Algorithm
def viterbi(sentence, trans_probs, emiss_counts, start_probs, tags, tag_to_idx, vocab, alpha=1.0):
    words = sentence.split()

```

```

n = len(words)
m = len(tags)
vocab_size = len(vocab)

# Function for Emission Prob with smoothing (handles rare/unknown tokens)
def get_emission_prob(tag, word):
    count = emiss_counts[tag][word]
    total = sum(emiss_counts[tag].values())
    # +1 in denominator for the 'unknown' token possibility
    return (count + alpha) / (total + alpha * (vocab_size + 1))

# dp[i][j] = max probability of being in tag j at word i
dp = np.zeros((n, m))
backpointer = np.zeros((n, m), dtype=int)

# Initialization Step
for j in range(m):
    dp[0][j] = start_probs[j] * get_emission_prob(tags[j], words[0])

# Recursion Step
for i in range(1, n):
    for j in range(m):
        # Calculate P(word | tag) * P(tag | prev_tag) * prev_dp_value
        probs = dp[i-1] * trans_probs[:, j] * get_emission_prob(tags[j], words[i])
        dp[i][j] = np.max(probs)
        backpointer[i][j] = np.argmax(probs)

# Termination / Path Backtracking
best_path = [0] * n
best_path[n-1] = np.argmax(dp[n-1])
for i in range(n-2, -1, -1):
    best_path[i] = backpointer[i+1][best_path[i+1]]

return [(words[i], tags[best_path[i]]) for i in range(n)]

# --- Main Execution ---
# Load data
df_subset = load_and_clean_data('Twitter_Data (1).csv', num_rows=1000)

# Build Model
trans_p, emiss_c, start_p, tag_list, t2idx, full_vocab = build_hmm_parameters(df_subset)

# Apply Viterbi to a sample tweet
sample_tweet = "talk all the nonsense and continue all the drama will vote for modi"
tagged_output = viterbi(sample_tweet, trans_p, emiss_c, start_p, tag_list, t2idx, full_vocab)

print(f"Original Tweet: {sample_tweet}")
print("Viterbi POS Tags:")
print(tagged_output)

# Display a snippet of the Transition Matrix (first 3 tags)
print("\nTransition Probabilities (Snippet):")
for i in range(3):
    for j in range(3):
        print(f"P({tag_list[j]} | {tag_list[i]}): {trans_p[i, j]:.4f}")

Original Tweet: talk all the nonsense and continue all the drama will vote for modi
Viterbi POS Tags:
[('talk', 'NN'), ('all', 'NN'), ('the', 'DT'), ('nonsense', 'NN'), ('and', 'CC'), ('continue', 'NN'), ('all', 'NN'), ('the', 'DT'), ('will', 'VBZ'), ('vote', 'VBD'), ('for', 'IN'), ('modi', 'NN')]

Transition Probabilities (Snippet):
P(DT | DT): 0.0254
P(PRP | DT): 0.0091
P(VB | DT): 0.0152
P(DT | PRP): 0.0140
P(PRP | PRP): 0.0056
P(VB | PRP): 0.2765
P(DT | VB): 0.0223
P(PRP | VB): 0.0211
P(VB | VB): 0.0149

```

