

1:SETUP & IMPORTS

```
import re
import nltk
import math
from collections import defaultdict, Counter

from nltk.tokenize import TweetTokenizer
from nltk import pos_tag
```

```
nltk.download('averaged_perceptron_tagger')
```

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

2:LOAD DATASET (TWITTER SENTIMENT CSV)

```
import pandas as pd

# Example: Kaggle Twitter Sentiment dataset
df = pd.read_csv("Twitter_Data.csv") # column: 'text'
tweets = df['clean_text'].dropna().tolist()
```

3:PREPROCESSING(REMOVING URLS AND MENTIONS)

```
tokenizer = TweetTokenizer(preserve_case=False)

def preprocess_tweet(tweet):
    tweet = re.sub(r"http\S+|www\S+", "", tweet) # remove URLs
    tweet = re.sub(r"@w+", "", tweet) # remove mentions
    return tokenizer.tokenize(tweet)

tokenized_tweets = [preprocess_tweet(t) for t in tweets]
```

4:BUILD HMM PARAMETERS,COUNT TRANSITION,EMISSION

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

```
START = "<START>"
END = "<END>"
```

```
for sent in tagged_tweets:
    prev_tag = START
    for word, tag in sent:
```

```

        transition_counts[prev_tag][tag] += 1
        emission_counts[tag][word] += 1
        tag_counts[tag] += 1
        prev_tag = tag
    transition_counts[prev_tag][END] += 1

```

5: CONVERT TO PROBABILITIES

```

transition_probs = defaultdict(dict)
emission_probs = defaultdict(dict)

for prev_tag, next_tags in transition_counts.items():
    total = sum(next_tags.values())
    for tag in next_tags:
        transition_probs[prev_tag][tag] = next_tags[tag] / total

for tag, words in emission_counts.items():
    total = sum(words.values())
    for word in words:
        emission_probs[tag][word] = words[word] / total

```

6: HMM PARAMETER SNAPSHOTS (TRANSITION)

```

print("Transition snapshot (PRP):")
print(transition_probs['PRP'])

```

```

Transition snapshot (PRP):
{'VB': 0.05321105808962704, 'VBP': 0.41347871291087945, 'VBD': 0.07568446589000

```

EMISSION SNAPSHOT

```

print("Emission snapshot (NN):")
print(list(emission_probs['NN'].items())[:10])

```

```

Emission snapshot (NN):
[('modi', 0.10166664002813029), ('government', 0.00570597449093757), ('governan

```

7: RARE AND UNKNOWN TOKEN ANALYSIS

```

vocab = Counter()
for sent in tokenized_tweets:
    vocab.update(sent)

rare_words = [w for w, c in vocab.items() if c == 1]

print("Number of rare tokens:", len(rare_words))

```

```

Number of rare tokens: 66217

```

8:MANUAL VITERBI DECODING (ONE EXAMPLE)

```
sentence = ["i", "love", "nlp"]
tags = list(tag_counts.keys())
```

VITERBI ALGORITHM

```
def viterbi(obs, tags, trans_p, emit_p):
    V = [{}]
    backpointer = [{}]

    # Initialization
    for tag in tags:
        trans = trans_p.get(START, {}).get(tag, 1e-6)
        emit = emit_p.get(tag, {}).get(obs[0], 1e-6)
        V[0][tag] = math.log(trans) + math.log(emit)
        backpointer[0][tag] = None

    # Recursion
    for t in range(1, len(obs)):
        V.append({})
        backpointer.append({})
        for tag in tags:
            max_prob, best_prev = max(
                (V[t-1][pt] + math.log(trans_p.get(pt, {}).get(tag, 1e-6)), pt)
                for pt in tags
            )
            emit = math.log(emit_p.get(tag, {}).get(obs[t], 1e-6))
            V[t][tag] = max_prob + emit
            backpointer[t][tag] = best_prev

    # Termination
    best_last_tag = max(V[-1], key=V[-1].get)

    # Backtrace
    best_path = [best_last_tag]
    for t in reversed(range(1, len(obs))):
        best_path.insert(0, backpointer[t][best_path[0]])

    return best_path
```

RUN VITERBI

```
predicted_tags = viterbi(sentence, tags, transition_probs, emission_probs)
print(list(zip(sentence, predicted_tags)))
```

```
[('i', 'NN'), ('love', 'NN'), ('nlp', 'NN')]
```

