Notebook Gemini

```
import pandas as pd

file_path = '/content/Twitter_Data.csv'
df = pd.read_csv(file_path, on_bad_lines='skip')
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

```
import re
import numpy as np

# Function to remove URLs
def remove_urls(text):
    if pd.isna(text):
        return ""
    url_pattern = re.compile(r'https?://\S+|www\.\S+')
    return url_pattern.sub(r'', str(text))

# Function to remove mentions
def remove_mentions(text):
    if pd.isna(text):
        return ""
    mention_pattern = re.compile(r'@\w+')
    return mention_pattern.sub(r'', str(text))

# Apply the preprocessing steps to the 'clean_text' column
df['processed_text'] = df['clean_text'].apply(remove_urls)
df['processed_text'] = df['processed_text'].apply(remove_mentions)

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

	clean_text	processed_text
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 nltk

# Download necessary NLTK data
nltk.download('punkt', quiet=True, raise_on_error=False)
nltk.download('averaged_perceptron_tagger_eng', quiet=True, raise_on_error=False)
nltk.download('punkt_tab', quiet=True, raise_on_error=False)
```

```
True
```

```
# Function to perform POS tagging
def pos_tag_text(text):
    if pd.isna(text) or text == "":
        return []
    tokens = nltk.word_tokenize(text)
    return nltk.pos_tag(tokens)

# Apply POS tagging to the 'processed_text' column
df['pos_tagged_text'] = df['processed_text'].apply(pos_tag_text)
```

```
# Display the DataFrame with the new pos_tagged_text column
display(df[['processed_text', 'pos_tagged_text']].head())
```

	processed_text	pos_tagged_text
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,...

```
from collections import defaultdict

# Initialize dictionaries for HMM parameters
initial_probabilities = defaultdict(lambda: 0)
transition_probabilities = defaultdict(lambda: defaultdict(lambda: 0))
emission_probabilities = defaultdict(lambda: defaultdict(lambda: 0))
tag_counts = defaultdict(lambda: 0)
vocabulary = set()

# Iterate through each POS-tagged tweet
for tagged_tweet in df['pos_tagged_text']:
    if not tagged_tweet:
        continue

    # Increment initial probability for the first tag in the tweet
    first_word, first_tag = tagged_tweet[0]
    initial_probabilities[first_tag] += 1
    tag_counts[first_tag] += 1
    vocabulary.add(first_word.lower())

    # Iterate through the rest of the words for transition and emission probabilities
    for i in range(len(tagged_tweet)):
        word, tag = tagged_tweet[i]
        vocabulary.add(word.lower())

        emission_probabilities[tag][word.lower()] += 1

        if i > 0:
            prev_word, prev_tag = tagged_tweet[i-1]
            transition_probabilities[prev_tag][tag] += 1
            tag_counts[tag] += 1 # Count all tags for normalization

# Normalize probabilities

# Normalize initial probabilities
total_initial_tags = sum(initial_probabilities.values())
if total_initial_tags > 0:
    for tag in initial_probabilities:
        initial_probabilities[tag] /= total_initial_tags

# Normalize transition probabilities
for prev_tag, next_tag_counts in transition_probabilities.items():
    total_transitions = sum(next_tag_counts.values())
    if total_transitions > 0:
        for next_tag in next_tag_counts:
            transition_probabilities[prev_tag][next_tag] /= total_transitions

# Normalize emission probabilities
for tag, word_counts in emission_probabilities.items():
    total_emissions = sum(word_counts.values())
    if total_emissions > 0:
        for word in word_counts:
            emission_probabilities[tag][word] /= total_emissions

print("HMM Parameters Built Successfully!")
print("\nTop 5 Initial Probabilities:")
for tag, prob in sorted(initial_probabilities.items(), key=lambda item: item[1], reverse=True)[:5]:
    print(f"  {tag}: {prob:.4f}")

print("\nTop 5 Transition Probabilities (e.g., from NN):")
if 'NN' in transition_probabilities:
        for next_tag, prob in sorted(transition_probabilities['NN'].items(), key=lambda item: item[1], reverse=True)[:5]:
            print(f"  NN -> {next_tag}: {prob:.4f}")
    else:
        print("  'NN' tag not found in transitions.")

    print("\nTop 5 Emission Probabilities (e.g., for NN):")
    if 'NN' in emission_probabilities:
        for word, prob in sorted(emission_probabilities['NN'].items(), key=lambda item: item[1], reverse=True)[:5]:
            print(f"  NN -> {word}: {prob:.4f}")
    else:
        print("  'NN' tag not found in emissions.")
```

```
HMM Parameters Built Successfully!

Top 5 Initial Probabilities:
  NN: 0.3298
  JJ: 0.1157
  RB: 0.1023
  DT: 0.0759
  NNS: 0.0688

Top 5 Transition Probabilities (e.g., from NN):
  NN -> NN: 0.3551
  NN -> IN: 0.0930
  NN -> NNS: 0.0617
  NN -> VBD: 0.0568
  NN -> RB: 0.0551

Top 5 Emission Probabilities (e.g., for NN):
  NN -> modi: 0.1005
  NN -> india: 0.0140
  NN -> congress: 0.0119
  NN -> bjp: 0.0112
  NN -> money: 0.0096
```

Inspect Transition Probabilities for a specific previous tag

```
#@title Inspect Transition Probabilities for a specific prev

# You can change the 'prev_tag_to_inspect' to any valid POS
prev_tag_to_inspect = 'NN' # @param {type:"string"}

print(f"\nTop 10 Transition Probabilities FROM '{prev_tag_to
if prev_tag_to_inspect in transition_probabilities:
    sorted_transitions = sorted(
        transition_probabilities[prev_tag_to_inspect].items(
        key=lambda item: item[1],
        reverse=True
    )[:10]

    if sorted_transitions:
        for next_tag, prob in sorted_transitions:
            print(f"  {prev_tag_to_inspect} -> {next_tag}: {
    else:
        print(f"  No observed transitions from '{prev_tag_to
else:
    print(f"  '{prev_tag_to_inspect}' tag not found in trans
```

prev_tag_to_inspect

NN

Hide code

```
Top 10 Transition Probabilities FROM 'NN':
  NN -> NN: 0.3551
  NN -> IN: 0.0930
  NN -> NNS: 0.0617
  NN -> VBD: 0.0568
  NN -> RB: 0.0551
  NN -> JJ: 0.0537
  NN -> CC: 0.0520
  NN -> VBZ: 0.0461
  NN -> MD: 0.0314
  NN -> DT: 0.0292
```

```
from collections import Counter
import itertools

# Flatten the list of tokenized words from 'pos_tagged_text'
# We only need the word, not the tag, for overall word frequency
```

```
all_words = list(itertools.chain.from_iterable(
    [word for word, tag in tweet_tags] for tweet_tags in df['pos_tagged_text'] if tweet_tags
))

# Convert all words to lowercase for consistent counting
all_words_lower = [word.lower() for word in all_words]

# Count word frequencies
word_counts = Counter(all_words_lower)

print("Total unique words in vocabulary:", len(word_counts))
print("Total words (including duplicates):", len(all_words_lower))

print("\nTop 20 most common words:")
for word, count in word_counts.most_common(20):
    print(f"  '{word}': {count}")

print("\nTop 20 rarest words (appearing only once):")
rare_words = [(word, count) for word, count in word_counts.items() if count == 1]
if rare_words:
    for word, count in sorted(rare_words, key=lambda item: item[0])[:20]: # Sort alphabetically for consistent output
        print(f"  '{word}': {count}")
else:
    print("  No words appear only once.")

# We can also inspect the words with the lowest emission probabilities for a specific tag
# For example, words rarely emitted by 'NN'
print("\nTop 10 words with lowest emission probability for 'NN' (if NN is present and has emissions):")
if 'NN' in emission_probabilities and emission_probabilities['NN']:
    sorted_nn_emissions = sorted(
        emission_probabilities['NN'].items(),
        key=lambda item: item[1]
    )[:10]
    for word, prob in sorted_nn_emissions:
        print(f"  NN -> '{word}': {prob:.6f}")
else:
    print("  'NN' tag not found or has no emission probabilities.")
```

```
Total unique words in vocabulary: 33517
Total words (including duplicates): 499980

Top 20 most common words:
  'modi': 22863
  'the': 15405
  'and': 10159
  'for': 7966
  'you': 6336
  'will': 4897
  'not': 4681
  'this': 4540
  'that': 4368
  'are': 4270
  'india': 3800
  'has': 3170
  'with': 3130
  'have': 3084
  'but': 2815
  'from': 2700
  'all': 2680
  '’': 2627
  'bjp': 2492
  'his': 2452

Top 20 rarest words (appearing only once):
  '000waiting': 1
  '025': 1
  '0420462818': 1
  '04280': 1
  '05852234246': 1
  '05yrs': 1
  '062': 1
  '081': 1
  '08th': 1
  '0deposit': 1
  '1000000': 1
  '1000000₹': 1
  '1000rs': 1
  '100100': 1
  '100agree': 1
  '100because': 1
  '100cr': 1
  '100dayscan': 1
  '100it': 1
  '100paise': 1

Top 10 words with lowest emission probability for 'NN' (if NN is present and has emissions):
  NN -> 'constituency2': 0.000007
  NN -> 'tuthukudi': 0.000007
  NN -> 'thuthukudi': 0.000007
  NN -> 'leadershipwho': 0.000007
  NN -> 'khammam': 0.000007
  NN -> 'condidate': 0.000007
  NN -> 'sonbhadra': 0.000007
  NN -> 'modiganga': 0.000007
  NN -> 'rejuvenation': 0.000007
  NN -> 'repressive': 0.000007
```

```
def viterbi(words, initial_probs, transition_probs, emission_probs, all_tags):
    # Initialize Viterbi path and probabilities
    # Viterbi_path[tag][t] stores the tag sequence ending with 'tag' at time 't'
    # Viterbi_prob[tag][t] stores the probability of the most likely path ending with 'tag' at time 't'
    T = len(words)
    Viterbi_prob = [{tag: 0.0 for tag in all_tags} for _ in range(T)]
    Viterbi_path = [{tag: [] for tag in all_tags} for _ in range(T)]

    # Step 1: Initialization for t = 0
    for tag in all_tags:
        # Initial probability of starting with 'tag'
        init_p = initial_probs.get(tag, 1e-10) # Use a small non-zero value for unseen tags

        # Emission probability of the first word given the tag
        word_lower = words[0].lower()
        emit_p = emission_probs.get(tag, {}).get(word_lower, 1e-10) # Use a small non-zero value for unseen words/tags

        Viterbi_prob[0][tag] = init_p * emit_p
        Viterbi_path[0][tag] = [tag]

    # Step 2: Recursion for t = 1 to T-1
    for t in range(1, T):
        word_lower = words[t].lower()
        for current_tag in all_tags:
            max_prob = 0.0
            best_prev_tag = None

            # Emission probability of the current word given the current tag
            emit_p = emission_probs.get(current_tag, {}).get(word_lower, 1e-10)

            for prev_tag in all_tags:
                # Probability of path ending with 'prev_tag' at t-1
                prob_prev_path = Viterbi_prob[t-1][prev_tag]

                # Transition probability from 'prev_tag' to 'current_tag'
                trans_p = transition_probs.get(prev_tag, {}).get(current_tag, 1e-10)

                # Calculate total probability for this path
                current_path_prob = prob_prev_path * trans_p * emit_p

                if current_path_prob > max_prob:
                    max_prob = current_path_prob
                    best_prev_tag = prev_tag

            Viterbi_prob[t][current_tag] = max_prob
            if best_prev_tag:
                Viterbi_path[t][current_tag] = Viterbi_path[t-1][best_prev_tag] + [current_tag]
            else:
                Viterbi_path[t][current_tag] = [current_tag] # Fallback if no prev tag found (shouldn't happen with 1e-10)

    # Step 3: Termination
    max_prob = 0.0
    best_last_tag = None

    for tag in all_tags:
        if Viterbi_prob[T-1][tag] > max_prob:
            max_prob = Viterbi_prob[T-1][tag]
            best_last_tag = tag

    if best_last_tag:
        return Viterbi_path[T-1][best_last_tag]
    else:
        return [] # Should not happen if words is not empty


# Get all unique tags from the training data
all_tags = set()
for tagged_tweet in df['pos_tagged_text']:
    for word, tag in tagged_tweet:
        all_tags.add(tag)

print("Viterbi algorithm defined and ready for use!")
```

```
Viterbi algorithm defined and ready for use!
```

Snapshot of HMM Parameters

Below are ways to inspect the `initial_probabilities`, `transition_probabilities`, and `emission_probabilities` that have been computed.

```
print("\n--- Initial Probabilities (Top 10) ---")
# Sorting to see the most probable initial tags
for tag, prob in sorted(initial_probabilities.items(), key=lambda item: item[1], reverse=True)[:10]:
    print(f"  Tag: {tag}, Probability: {prob:.4f}")

print("\n--- Transition Probabilities (Examples from 'NN', 'VB', 'JJ') ---")
# Inspecting a few example tags for their transitions
for prev_tag_example in ['NN', 'VB', 'JJ']:
    if prev_tag_example in transition_probabilities:
        print(f"  Top 5 transitions from '{prev_tag_example}':")
        sorted_transitions = sorted(
            transition_probabilities[prev_tag_example].items(),
            key=lambda item: item[1],
            reverse=True
        )[:5]
        if sorted_transitions:
            for next_tag, prob in sorted_transitions:
                print(f"    {prev_tag_example} -> {next_tag}: {prob:.4f}")
        else:
            print(f"    No observed transitions from '{prev_tag_example}'.")
    else:
        print(f"  '{prev_tag_example}' tag not found in transition probabilities.")

print("\n--- Emission Probabilities (Examples for 'NN', 'VB', 'JJ') ---")
# Inspecting a few example tags for their emissions
for tag_example in ['NN', 'VB', 'JJ']:
    if tag_example in emission_probabilities:
        print(f"  Top 5 emitted words for '{tag_example}':")
        sorted_emissions = sorted(
            emission_probabilities[tag_example].items(),
            key=lambda item: item[1],
            reverse=True
        )[:5]
        if sorted_emissions:
            for word, prob in sorted_emissions:
                print(f"    {tag_example} -> '{word}': {prob:.4f}")
        else:
            print(f"    No observed emissions for '{tag_example}'.")
    else:
        print(f"  '{tag_example}' tag not found in emission probabilities.")
```

```
--- Initial Probabilities (Top 10) ---
  Tag: NN, Probability: 0.3298
  Tag: JJ, Probability: 0.1157
  Tag: RB, Probability: 0.1023
  Tag: DT, Probability: 0.0759
  Tag: NNS, Probability: 0.0688
  Tag: VB, Probability: 0.0461
  Tag: IN, Probability: 0.0379
  Tag: WRB, Probability: 0.0345
  Tag: PRP, Probability: 0.0302
  Tag: MD, Probability: 0.0202

--- Transition Probabilities (Examples from 'NN', 'VB', 'JJ') ---
```

```
  Top 5 transitions from 'NN':
    NN -> NN: 0.3551
    NN -> IN: 0.0930
    NN -> NNS: 0.0617
    NN -> VBD: 0.0568
    NN -> RB: 0.0551
  Top 5 transitions from 'VB':
    VB -> NN: 0.1897
    VB -> DT: 0.1501
    VB -> JJ: 0.1174
    VB -> IN: 0.0818
    VB -> PRP$: 0.0807
  Top 5 transitions from 'JJ':
    JJ -> NN: 0.5571
    JJ -> NNS: 0.1561
    JJ -> JJ: 0.0989
    JJ -> IN: 0.0334
    JJ -> RB: 0.0214

--- Emission Probabilities (Examples for 'NN', 'VB', 'JJ') ---
  Top 5 emitted words for 'NN':
    NN -> 'modi': 0.1005
    NN -> 'india': 0.0140
    NN -> 'congress': 0.0119
    NN -> 'bjp': 0.0112
    NN -> 'money': 0.0096
  Top 5 emitted words for 'VB':
    VB -> 'have': 0.0401
    VB -> 'modi': 0.0370
    VB -> 'get': 0.0363
    VB -> 'give': 0.0299
    VB -> 'make': 0.0191
  Top 5 emitted words for 'JJ':
    JJ -> 'modi': 0.0494
    JJ -> 'poor': 0.0224
    JJ -> 'indian': 0.0164
    JJ -> 'narendra': 0.0142
    JJ -> 'good': 0.0136
```

Apply Viterbi Decoding to a Sample Tweet

Now, let's take a sample tweet, tokenize it, and then apply our `viterbi` function to find its most likely POS tag sequence.

```
# Choose a sample tweet (you can modify this tweet or select another from df['processed_text'])
sample_tweet = "modi will make india great again"

# Tokenize the sample tweet
sample_words = nltk.word_tokenize(sample_tweet)
print(f"Sample Tweet: '{sample_tweet}'")
print(f"Tokenized Words: {sample_words}")

# Apply Viterbi decoding
predicted_tags = viterbi(sample_words, initial_probabilities, transition_probabilities, emission_probabilities, all_tags)

# Combine words and predicted tags for display
result = list(zip(sample_words, predicted_tags))

print(f"\nPredicted POS Tag Sequence: {result}")
```

```
Sample Tweet: 'modi will make india great again'
Tokenized Words: ['modi', 'will', 'make', 'india', 'great', 'again']

Predicted POS Tag Sequence: [('modi', 'NN'), ('will', 'MD'), ('make', 'VB'), ('india', 'JJ'), ('great', 'JJ'), ('again', 'RB')]
```

```
Start coding or generate with AI.
```