

Ex1b: Hidden Markov Model (HMM) based Predictive Text System

AIM :

To implement a predictive text system using Hidden Markov Model (HMM), enabling to understand contextual probability, POS transitions, and next-word prediction using the Brown Corpus.

ALGORITHM :

1. Initialize the environment by installing and importing the Natural Language Toolkit (nltk). Download the Brown Corpus (for text data) and the Universal Tagset (for simplified Part-of-Speech tags).
2. Extract raw sentences from the Brown Corpus using `brown.sents()`.
3. Tokenization: Convert all words to lowercase and flatten the nested list into a single list of tokens to verify the total volume of training data.
4. Load the tagged version of the corpus (`brown.tagged_sents`) which provides the And Hidden States; (POS tags) linked to the and Observation; (words).
5. Train the HMM Model. Use the `HiddenMarkovModelTrainer` to perform Supervised Learning. Calculate the internal parameters of the HMM: Initial State Probabilities, Transition Probabilities, Emission Probabilities
6. Build POS Transition Probabilities
7. Build Emission Probabilities
8. Define Prediction Function
9. Test the Model
10. Verify POS prediction on certain words.

CODE :

Python

```
import nltk
from nltk.corpus import brown, stopwords
from nltk.util import ngrams
from collections import defaultdict, Counter
import string

nltk.download('brown')
nltk.download('universal_tagset')
nltk.download('stopwords')

STOPWORDS = set(stopwords.words('english'))

tagged_sentences = brown.tagged_sents(tagset='universal')
transition = defaultdict(Counter)
emission = defaultdict(Counter)
for sent in tagged_sentences:
    prev_tag = '<s>'
    for word, tag in sent:
        word = word.lower()
        if not word.isalpha():
            continue
        if word in STOPWORDS:
            continue
        transition[prev_tag][tag] += 1
        emission[tag][word] += 1
        prev_tag = tag

def predict_hmm_candidates(previous_word, top_k=5):
    previous_word = previous_word.lower()
    possible_tags = []
    for tag in emission:
        if previous_word in emission[tag]:
            possible_tags.append(tag)
```

```

if not possible_tags:
    return "No prediction found"

word_scores = Counter()
for tag in possible_tags:
    total_trans = sum(transition[tag].values())
    if total_trans == 0:
        continue

    for next_tag, t_count in transition[tag].items():
        t_prob = t_count / total_trans

        total_emis = sum(emission[next_tag].values())
        if total_emis == 0:
            continue

        for word, e_count in emission[next_tag].items():
            e_prob = e_count / total_emis
            word_scores[word] += t_prob * e_prob

if not word_scores:
    return "No prediction found"

return word_scores.most_common(top_k)

previous_word = input("Enter previous word (HMM):")
previous_word = previous_word.strip().lower()
results = predict_hmm_candidates(previous_word)
print("\nHMM Suitable Next Words (Ranked):")
if isinstance(results, str):
    print(results)
else:
    for word, score in results:
        print(f"{word} → score: {score:.6f}")

```

OUTPUT :

Enter previous word (HMM): government

HMM Suitable Next Words (Ranked):

would → score: 0.006515

one → score: 0.005276

said → score: 0.004708

could → score: 0.003843

may → score: 0.003310

RESULT :

Thus a HMM based model was built to predict the next word with POS-based suggestions, and evaluated the effectiveness of context-driven predictive text generation using the Brown Corpus.