A Short Technical Report towards A7707 NLP course end project

Text Prediction With N-Grams: An Interactive Implementation in Python

submitted in the Partial Fulfilment of the

Requirements

for the Award of the Degree of

BACHELOR OF TECHNOLOGY

IN

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

Submitted By

KANAGANTI HIMAVAMSHI	21881A7223
MEDAPATI JAYANTH	21881A7233
MOHAMMED NAWAZ	21881A7236

Department of Artificial Intelligence and Data Science

VARDHAMAN COLLEGE OF ENGINEERING

(AUTONOMOUS)

Affiliated to JNTUH, Approved by AICTE, Accredited by NAAC with A++ Grade, ISO 9001:2015 Certified Kacharam, Shamshabad, Hyderabad – 501218, Telangana, India

2023- 24

ABSTRACT

Text prediction is an important capability for applications like auto-complete and conversational agents. This project implements an n-gram statistical language model for next-word prediction in Python. The model is trained on arbitrary text file uploaded by the user. An interactive web interface is built using Streamlit for generating predictions on input text. N-gram models predict the next word based on the previous n-1 words of context. The model counts all n-grams and their frequencies in the training corpus. For prediction, it looks up the previous n-1 words and randomly samples the next word based on the learned probability distribution. Users can customize the n-gram size in the interface. The application demonstrates core natural language processing concepts like tokenization, language modeling, and probability estimation.

1. INTRODUCTION

Text prediction is an important capability for many applications today including autocomplete, search engines, and conversational agents. Suggesting likely next words as the user is typing, can enhance productivity by reducing keystrokes and correcting mistakes. The utility of text prediction has led to widespread adoption in mobile keyboards, web search, and voice assistants.

Text prediction is powered by statistical language modeling - estimating a probability distribution over sequences of words. N-gram models are a simple yet effective technique for language modeling and next-word prediction. An n-gram is a contiguous sequence of n-words from a sentence. By analyzing n-grams in a large text corpus, we can estimate probabilities of word sequences.

This project implements an n-gram text predictor in Python that trains on custom text files. It tokenizes the text, generates n-grams, builds probability distributions, and allows interactively predicting the next word for a given text prefix. The goals are:

- Demonstrate core NLP techniques like tokenization, n-gram modeling, and probability estimation.
- Implement customizable n-gram modeling that trains on arbitrary text.
- Develop an interactive web interface for user text prediction using Streamlit.
- Provide a solid baseline that can be extended to more advanced models like backoff and smoothing.

This foundation will allow exploring text prediction for applications like autocomplete, search query suggestions, and conversational agents. The interactive interface facilitates experimentation and understanding.

2. METHODOLOGY:

The core methodology is n-gram language modeling and probability estimation. The key steps are:

Text Preprocessing:

- Tokenize the input text into words/tokens using NLTK. This gives us the vocabulary for the language model.
- Normalize tokens to lowercase.

- Remove punctuation and special characters.

N-Gram Generation:

- Slide a window of size n over the tokenized text to extract all contiguous n-grams.
- Repeat this for n=2 to 10 to extract unigrams up to ten grams.

Probability Estimation:

- Count the frequency of each unique n-gram in the corpus.
- To get probabilities, divide the count by the total counts for that (n-1)-gram prefix.

Prediction:

- Take the input context of previous n-1 words.
- Look up the counts/probabilities for the identified n-gram prefix.
- Randomly sample the next word based on the distribution.

$$\hat{w} = \operatorname{argmax} P(w|h)$$

The model extracts n-grams up to size 10 to take longer word sequence context into account. We use backoff to lower-order n-grams when higher ones are unavailable.

This process trains the n-gram model on the text corpus, which can then be used interactively to predict the next token for a given text sequence.

3. IMPLEMENTATION SOURCE CODE:

```
from nltk.tokenize import word_tokenize
import streamlit as st
import random

def generate_ngrams(words, n):
    print(f"Generating {n}-grams")
    ngrams_list = []
    for j in range(2, n):
        for i in range(len(words) - j + 1):
            ngrams_list.append(tuple(words[i:i+j]))
    return ngrams_list

def create_ngram_model(text, n):
    print(f"Creating {n}-gram model")
    ngrams_freq = {}
    ngrams = generate_ngrams(text, n)
```

```python
    for gram in ngrams:
        prefix = gram[:-1]
        suffix = gram[-1]
        if prefix not in ngrams_freq:
            ngrams_freq[prefix] = {}
        if suffix not in ngrams_freq[prefix]:
            ngrams_freq[prefix][suffix] = 0
        ngrams_freq[prefix][suffix] += 1
    for prefix in ngrams_freq:
        total_count = float(sum(ngrams_freq[prefix].values()))
        for suffix in ngrams_freq[prefix]:
            ngrams_freq[prefix][suffix] /= total_count
    return ngrams_freq

def predict_next_word(model, prefix):
    if prefix in model:
        choices = model[prefix]
        return random.choices(list(choices.keys()), weights=list(choices.values()))[0]
    else:
        return None

def main():
    st.title("N-gram Model Prediction")
    uploaded_file = st.file_uploader("Upload Text File", type=["txt"])
    if uploaded_file is not None:
        text =  uploaded_file.read().decode("utf-8").lower()
        tokens = word_tokenize(text)
        n = st.slider("Select the n-gram:", min_value=2, max_value=10, value=3)
        ngram_model = create_ngram_model(tokens, n)
        n_more=st.slider("Select the number of words to predict:", min_value=10, max_value=150, value=50)
        input_sentence = st.text_input("Enter a sentence for prediction:")
        if st.button("Predict"):
            input_sentence = input_sentence.lower()
            output = input_sentence
            input_tokens = word_tokenize(input_sentence)
            for i in range(n_more):
                prefix = tuple(input_tokens[-(n - 2):])
                predicted_word = predict_next_word(ngram_model, prefix)
                if predicted_word is None:
                    break
                input_tokens.append(predicted_word)
            predicted_sentence = ' '.join(input_tokens)
            st.write(f"Predicted sentence: {predicted_sentence}")

if __name__ == "__main__":
    main()
```

4. Results:

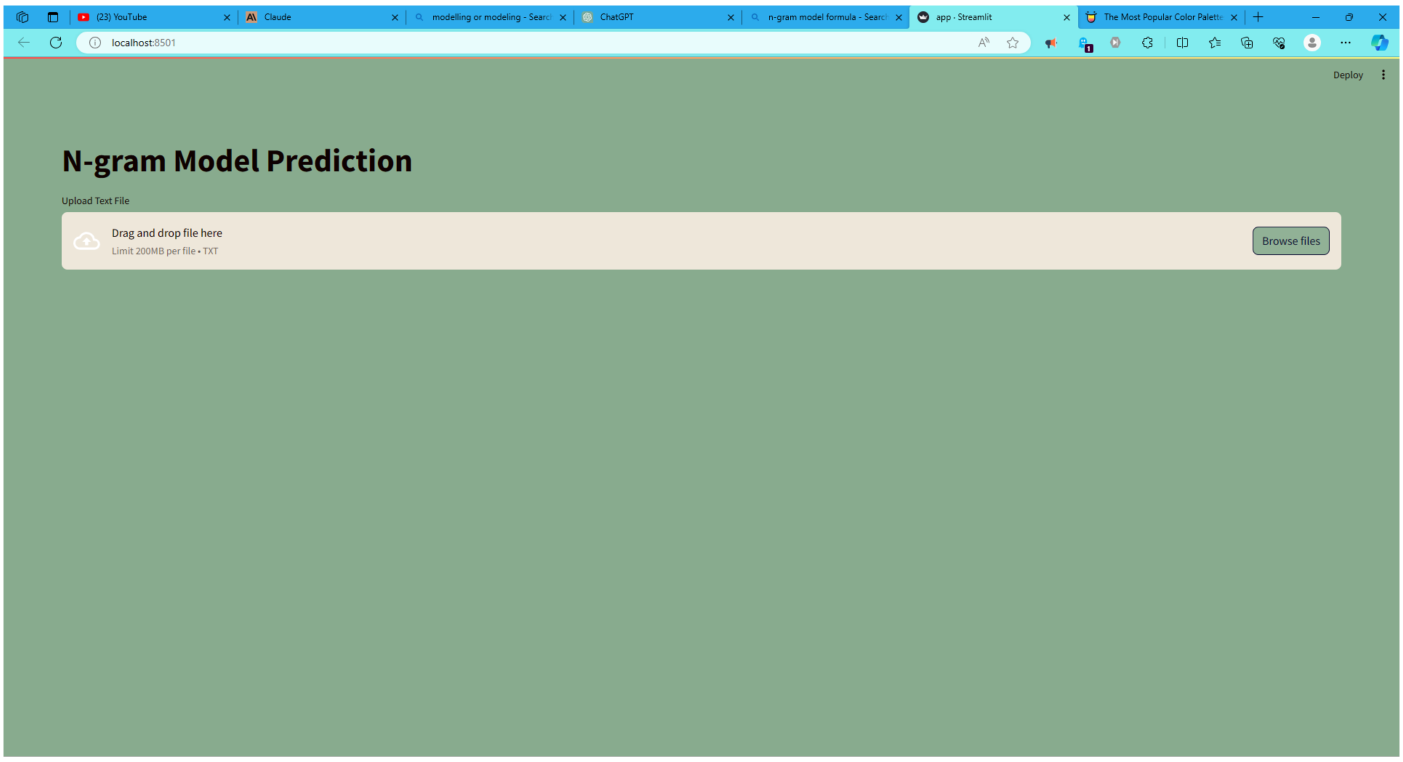
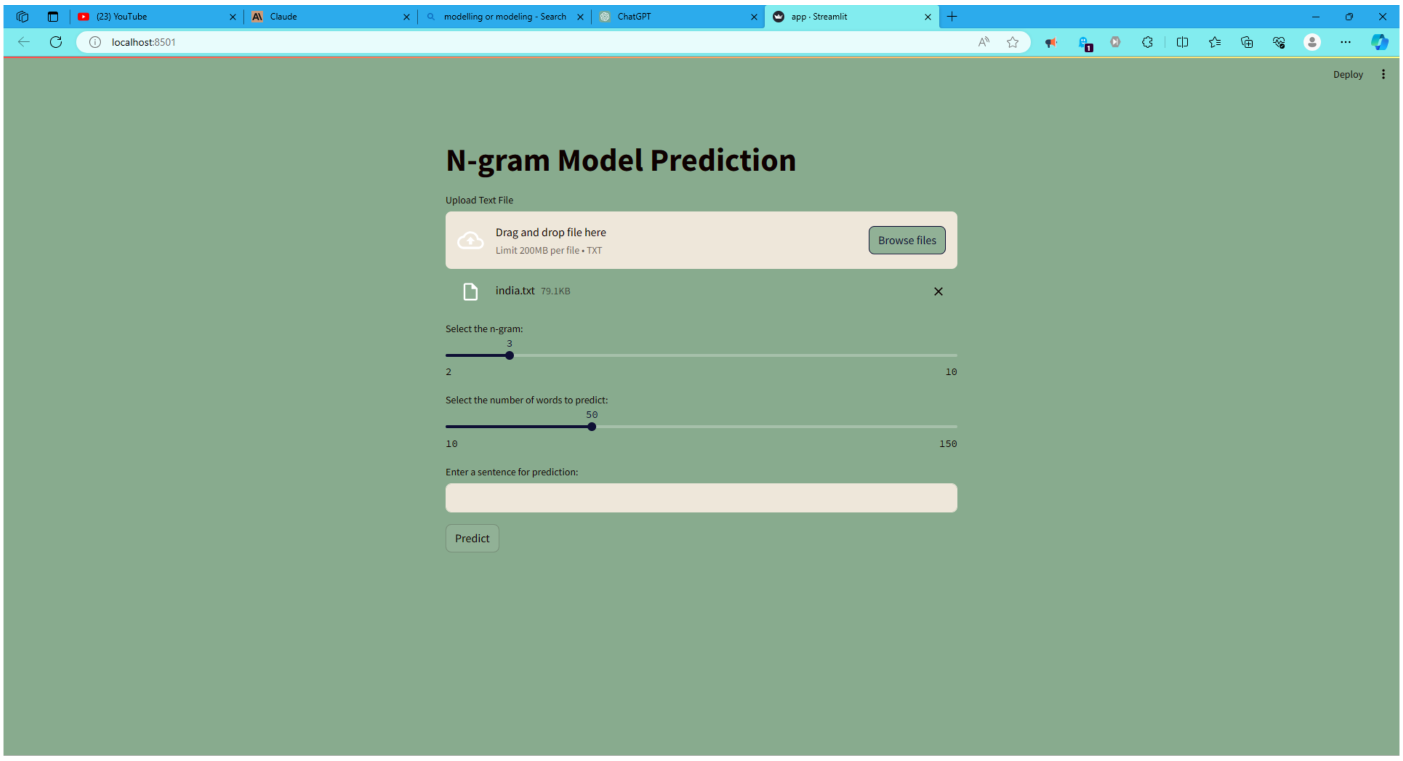


Figure 1 Initial web page



N-gram Model Prediction

Upload Text File

Drag and drop file here
Limit 200MB per file • TXT
Browse files

india.txt 79.1KB

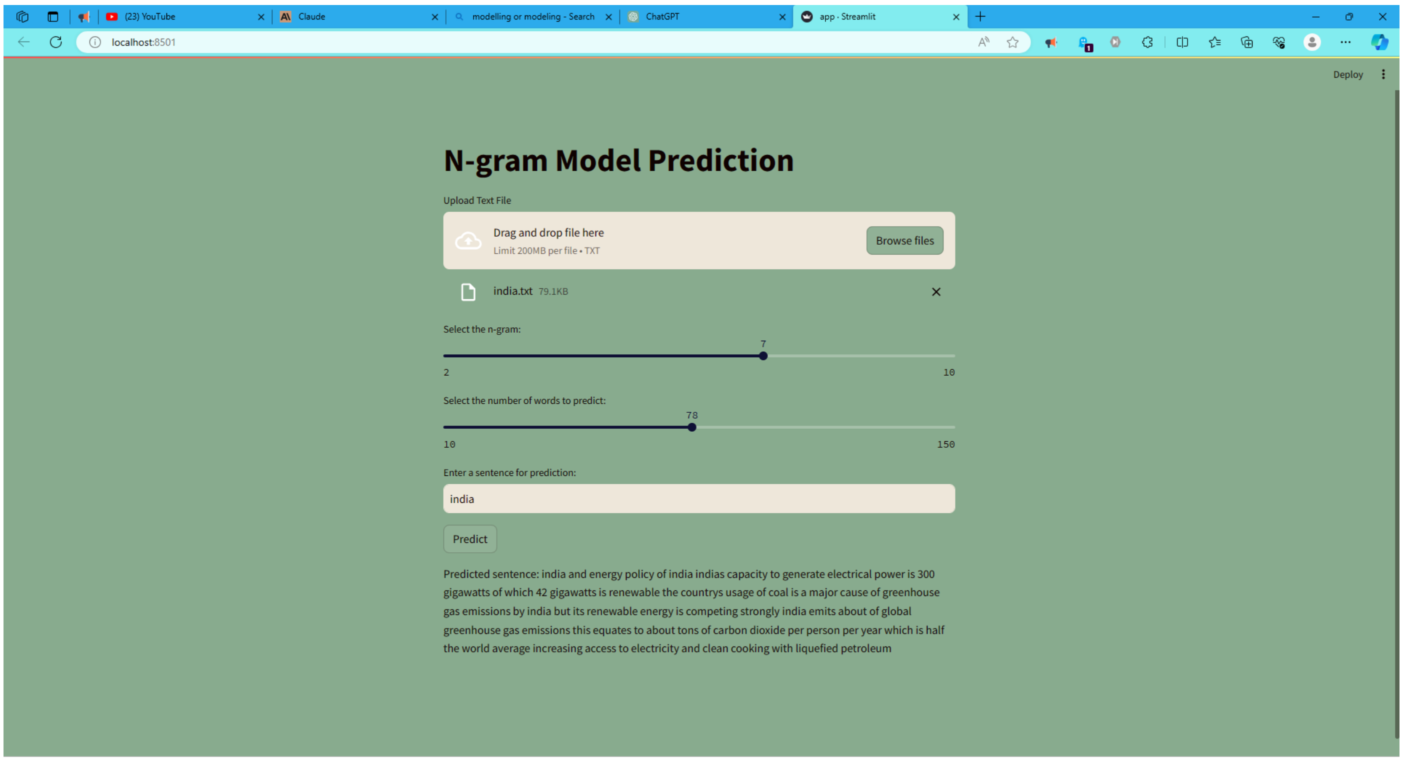
Select the n-gram: 3 (range 2–10)

Select the number of words to predict: 50 (range 10–150)

Enter a sentence for prediction:

Predict

Figure 2 Preview after document upload



N-gram Model Prediction

Upload Text File

Drag and drop file here
Limit 200MB per file • TXT
Browse files

india.txt 79.1KB

Select the n-gram: 7 (range 2–10)

Select the number of words to predict: 78 (range 10–150)

Enter a sentence for prediction: india

Predict

Predicted sentence: india and energy policy of india indias capacity to generate electrical power is 300 gigawatts of which 42 gigawatts is renewable the countrys usage of coal is a major cause of greenhouse gas emissions by india but its renewable energy is competing strongly india emits about of global greenhouse gas emissions this equates to about tons of carbon dioxide per person per year which is half the world average increasing access to electricity and clean cooking with liquefied petroleum

Figure 3 prediction

1. The initial webpage

This (i.e. Fig 1) shows the title and initial layout of the web interface built with Streamlit. It contains all the key components like file upload, parameters, text input, and prediction output that allow the use of the n-gram model interactively.

2. Text file upload

The (i.e. Fig 2) interface allows uploading a .txt file containing the text corpus to train the n-gram model. The model-building phase tokenizes this text, extracts n-grams, and estimates probabilities. Uploading arbitrary training text enables customizing the prediction model.

3. N-gram size selection, number of predicted words, and prediction

These (i.e. Fig 3) interfaces allow configuring the key parameters and generating predictions. Users can specify:

- **N-gram size:** This controls the length of context used for predictions. A higher n considers more context but may have sparsity issues.
- **Number of predicted words:** How many additional words to predict after the input text.
- **Input text:** The seed text for predictions.
- **Prediction output:** Shows the original text and predicted continuation based on the model.

The interactive controls enable exploring how n-gram size and context length affect the generated text predictions.

5. Conclusion:

This project implemented a customizable n-gram text predictor that trains on arbitrary text corpora provided by the user. The core techniques included tokenization, n-gram modeling, probability estimation, and sampling for prediction. The interactive web interface allows exploring the impact of parameters like n-gram size on predictions.

While simple, n-gram models are quite effective for text prediction in practice. This project provided a solid foundation in core NLP concepts. Some potential extensions could further improve performance:

- Smoothing techniques like Laplace and backoff to address data sparsity for rare n-grams.
- Neural network language models like Word2Vec to learn distributed representations.
- Beam search for constrained predictions highly likely in context
- Additional preprocessing like stemming to reduce vocabulary size.

Overall, the system demonstrates how statistical language modeling can enable predictive text generation. The interactive interface facilitates experimentation and building intuition. This can serve as a baseline for more advanced techniques to improve prediction accuracy and user experience. The project also highlights practical applications of NLP like autocomplete and search queries.

REFERENCES:

[1] Chen, Stanley F., and Joshua Goodman. "An empirical study of smoothing techniques for language modeling." Computer Science Technical Report TR-10-98 (1998). [Classic paper on smoothing techniques like Laplace and backoff to address data sparsity]

[2] Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." arXiv preprint arXiv:1301.3781 (2013). [Introduces Word2Vec model for learning distributed word representations]

Online Resources:

- N-gram language models tutorial https://web.stanford.edu/~jurafsky/slp3/3.pdf
- NLTK toolkit for Python https://www.nltk.org/api/nltk.html
- Streamlit library https://docs.streamlit.io/en/stable/