```python
import numpy as np
import math
import sys

class NGramModel:
    vocab_size = 0
    ngram_probs = []
    ngram_counts = ''
    words = []
    processed_text = ''
    def __init__(self, N, data):
        self.N = N
        self.words = self.Preprocess_Text(data)
        self.vocab_size = self.get_vocab_size(self.words)

    # Preprossing / cleaning raw data
    def Preprocess_Text(self, text):
        lines = text.split('\n')
        _words = []
        for line in lines:
            phonemes = line.split(' ')
            phonemes = phonemes[1:-1]
            word = ' '.join(phonemes)
            _words.append(word)
        return _words

    # Finding number of unique words in the text
    def get_vocab_size(self, _words):
        v = []
        for word in _words:
            for phoneme in word:
                if phoneme not in v:
                    v.append(phoneme)
        return len(v)

    # Preparing data for using N-gram model
    def prepare_data(self):

        words = self.words
        cleaned_text = []

        for word in words:
            # Adding <S> at start and </S> at end of each sentence of text
            s = ["<s>"] + word.strip().split() + ["</s>"]
            cleaned_text.append(s)

        return cleaned_text

    # Storing counts and probabilities of N-grams in the training text
    def evaluate_Ngrams(self,text):
        ngrams = []
        ngramCounts = []
        for n in range(self.N):
            # dictionary for storing count of ngrams
            currGramCounts = {}
            totalCount = 0
            for line in text:
```

```
                for i in range(len(line) - (n)):
                    # grouping of words according the 'n' value for ngram
model
                    ngram = line[i:i+n+1]
                    ngram = " ".join(ngram)
                    if not ngram in currGramCounts:
                        currGramCounts[ngram] = 0
                    currGramCounts[ngram] += 1
                    # total count of ngrams
                    totalCount += 1
            ngramCounts.append(currGramCounts)

            # dictionaries to store the current and overall Ngram
probabilities
            currProb = {}
            for ngram in currGramCounts.keys():
                ngram_prefix = " ".join(ngram.split()[:-1])
                if n != 0:
                    # this is C(wi-1), count of prefix sentence of the given
word
                    countPrefix = ngramCounts[n - 1][ngram_prefix]
                else:
                    # this is unigram model as n=0
                    countPrefix = totalCount
                currProb[ngram] = ngramCounts[n][ngram] / (countPrefix*1.0)

            self.ngram_probs.append(currProb)

        return ngramCounts


    # Evaluating probability of each word to get test sentence probability
    def find_probability(self,ngrams,test_sentence):
        sentence_prob = 1.0
        sentence = ["<s>"] +test_sentence.split() + ["</s>"]
        for i in range(len(sentence) - (self.N - 1)):
            # Constructing ngrams from the text sentence to find the
probability of given test sentence
            # according the calculated ngram probabilities as given above
            ngram = sentence[i:i+self.N]
            ngram = " ".join(ngram)
            ngram_prefix = " ".join(ngram.split()[:-1])
            try:
                prefix_count = ngrams[self.N-2][ngram_prefix]
            except:
                prefix_count = 0

            # Applying Laplace smoothing (Add-one) to resolve if an Ngram
doesn't appear in the training corpus
            try:
                ngram_prob = ngrams[self.N-1][ngram]
            except:
                ngram_prob = 0

            # calculating the probability of current Ngram
            curr_ngram_prob = (ngram_prob + 1)*1.0 /
(prefix_count+self.vocab_size)
```

```
            # multiplying the probabilities of each ngram to get overall
sentence probability
            sentence_prob += math.log(curr_ngram_prob, 2)

        return sentence_prob

    # Calculating Model Perplexity
    def model_perplexity(self,test_sentence,sentence_prob):

        N = len(test_sentence.split(" "))
        power = -1.0/N
        try:
            Perplexity = math.pow(2, sentence_prob * power)
            return Perplexity
        except:
            return -1

    # Arranging N-gram probilities by prefix for easy access
    def Ngrams_by_prefix(self,ngrams):
        ngramsByPrefix = {}
        for N in range(1,self.N+1):
            for ngram in ngrams[N - 1].keys():
                # get the first part of the ngram
                prefix = " ".join(ngram.split()[:-1])
                if prefix not in ngramsByPrefix:
                    ngramsByPrefix[prefix] = {}
                # creating a dictionary of 'ngram:probability' pairs for
given prefix
                ngramsByPrefix[prefix][ngram] = ngrams[N - 1][ngram]
        return ngramsByPrefix

    # To generate sentences using the given model
    def Generate_sentences(self,num_generate):

        if self.processed_text == '':
            self.processed_text = self.prepare_data()
        processed_text = self.processed_text

        if self.ngram_probs == []:
            self.evaluate_Ngrams(processed_text)

        generated_sentences = []
        start = "<s>"
        # getting ngrams by prefix for easy generation of probable sentences
from this model
        ngramsByPrefix = self.Ngrams_by_prefix(self.ngram_probs)
        x = num_generate
        while(x>0):
            sentence = start
            # repeat till we encounter end of sentence symbol i.e. '</s>'
            while(sentence.split()[-1] != "</s>"):
                prefix = " ".join(sentence.split()[-self.N+1:])
                # this is a dictionary of all probable next ngrams starting
with the given prefix
                try:
                    next_ngrams = ngramsByPrefix[prefix]
```

```
                except:
                    break
                # getting only the keys of the dictionary as they contain
probable ngrams
                next_ngrams_list = list(next_ngrams.keys())
                next_probs = np.array(list(next_ngrams.values()))
                next_probs = next_probs/np.sum(next_probs)
                # using numpy.random.choice(a, size=None, replace=True,
p=None) to choose an ngram acc to probabilities of sample
                ngram_i = np.random.choice(next_probs.shape[0],1,p =
next_probs)[0]
                # now we have the index of next probable ngram, we can
concatenate that to generate the complete sentence
                suffix = " ".join(next_ngrams_list[ngram_i].split()[-1:])
                sentence += " "+suffix

            # removing the <s> at beginning and </s> at the end
            sentence = " ".join(sentence.split()[1:-1])
            if(sentence!=""):
                x-=1
                generated_sentences.append('#' + sentence + '#')

        if (generated_sentences!=[]):
            return generated_sentences
        else:
            return -1


    def Language_Modelling(self,test_sentence):

        if self.processed_text == '':
            self.processed_text = self.prepare_data()
        processed_text = self.processed_text

        if self.ngram_counts == '':
            self.ngram_counts = self.evaluate_Ngrams(processed_text)
        ngram_counts = self.ngram_counts

        sentence_prob = self.find_probability(ngram_counts, test_sentence)
        Perplexity = self.model_perplexity(test_sentence, sentence_prob)
        return sentence_prob, Perplexity

    def Sentence_Generation(self,num_generate):
        generated_sentences = self.Generate_sentences(num_generate)
        print("\nThe following words can be generated using given model: ")
        no=1
        for sentence in generated_sentences:
            print(no, sentence)
            no+=1
        print("\n")

if __name__ == '__main__':

    args = (sys.argv)

    assert len(args) > 2
```

```
    if len(args) == 3:
        training_file = args[1]
        data3 = open(training_file, "r").read()
        N = int(args[2])
        model_obj = NGramModel(N, data3)
        model_obj.Sentence_Generation(5)


    if len(args) == 4:
        training_file = args[1]
        data3 = open(training_file, "r").read()
        N = int(args[2])
        testFile = args[3]
        model_obj = NGramModel(N, data3)

        data_points = []
        sum_perp_for_X = 0
        testFile = open(testFile)
        for line in testFile:
            line = line.replace('\n', '')
            prob, perp = model_obj.Language_Modelling(line)
            sum_perp_for_X += prob
            data_points.append([prob, 1])
            print('Word #' + line + '#' + '| ' + 'Prob.=' + str(prob) + '
Perp.=' + str(perp))

        avg_prep_x = sum_perp_for_X / len(data_points)

        print('----------------------------------------')

        print('AVG of perplexity is ' + str(avg_prep_x))
```