Activity 1 — N-Gram Language Modeling and Sentence Generation

Objective

To build and evaluate Bigram, Trigram, and 4-gram models that generate syntactically coherent sentences (10–12 words) given two starting words, and compare their performance using perplexity.

Importing Required Libraries

We import the necessary Python libraries for natural language processing, text tokenization, and statistical modeling.

Downloading NLTK Data

This function ensures that the required NLTK datasets — 'punkt', 'gutenberg', and 'brown' — are available for tokenization and training.

Loading and Combining the Text Corpus

We load multiple texts from the Gutenberg and Brown corpora.
The combined dataset exceeds 50,000 words to provide sufficient training data for the N-gram models.

Preprocessing the Text

Each text is:

- Sentence tokenized
- Word tokenized
- Converted to lowercase
- Surrounded by `<s>` and `</s>` tokens to mark sentence boundaries This prepares the data for N-gram model training.

Defining the N-Gram Model Class

The `NGramModel` class:

- Counts n-grams and (n-1)-gram contexts
- Calculates probabilities using MLE and Laplace smoothing
- Generates new sentences
- Computes perplexity on test data

Training and Evaluating the N-Gram Models

We train and test Bigram, Trigram, and 4-gram models. Each model is trained on 90% of the sentences and evaluated on 10% using perplexity.

Running the Main Program

This section:

- Downloads NLTK data
- Loads and preprocesses the text
- Trains all models
- Calculates perplexities
- Generates 5 example sentences for each model

```
import math
import random
from collections import defaultdict, Counter
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.corpus import gutenberg, brown
from tqdm import tqdm
```

```
# 1. NLTK data download (run once)
def download_nltk_data():
    nltk.download('punkt')
    nltk.download('gutenberg')
    nltk.download('brown')
    nltk.download('punkt_tab')
```

```
# 2. Load and assemble corpus
def load_corpus(use_gutenberg=True, use_brown=True, min_words=50000):
    texts = []
    if use_gutenberg:
        # list of gutenberg fileids to include (adjust as needed)
        guten_ids = ['austen-emma.txt', 'austen-persuasion.txt', 'austen-
        for fid in guten_ids:
            texts.append(' '.join(gutenberg.words(fid)))
    if use_brown:
        texts.append(' '.join(brown.words()))
```

```
    combined = '\n'.join(texts)
    words = word_tokenize(combined)
    if len(words) < min_words:
        print(f"WARNING: Combined tokens {len(words)} < required {min_wor
    return combined
```

```
# 3. Preprocessing utilities
def preprocess_text(raw_text):
    sents = sent_tokenize(raw_text)
    tokenized_sents = []
    for sent in sents:
        tokens = word_tokenize(sent)

        # Filter for alphabetic tokens AND lowercase them in one step
        tokens = [t.lower() for t in tokens if t.isalpha()]

        # IMPORTANT: Skip sentences that are now empty (e.g., if a "sente
        if not tokens:
            continue

        # add boundary tokens
        tokenized_sents.append(['<s>'] + tokens + ['</s>'])
    return tokenized_sents
```

```
# 4. N-gram model builder
class NGramModel:
    def __init__(self, n):
        assert n >= 2
        self.n = n
        self.counts = Counter()    # counts of n-grams (tuples)
        self.context_counts = Counter()  # counts of (n-1)-gram contexts
        self.vocab = set()
        self.total_contexts = 0

    def train(self, tokenized_sentences):
        for sent in tokenized_sentences:
            # update vocabulary
            for w in sent:
                self.vocab.add(w)
            # pad and extract ngrams
            for i in range(len(sent) - self.n + 1):
                ngram = tuple(sent[i:i+self.n])
                context = tuple(sent[i:i+self.n-1])
                self.counts[ngram] += 1
                self.context_counts[context] += 1
        self.total_contexts = sum(self.context_counts.values())

    def mle_prob(self, ngram):
        context = ngram[:-1]
        num = self.counts[ngram]
        denom = self.context_counts.get(context, 0)
        if denom == 0:
```

```
            return 0.0
        return num / denom

    def laplace_prob(self, ngram, alpha=1.0):
        context = ngram[:-1]
        num = self.counts[ngram] + alpha
        denom = self.context_counts.get(context, 0) + alpha * len(self.vo
        return num / denom

    def generate(self, start_two, max_words=12, smoothing='laplace', alph
        if self.n < 2:
            raise ValueError("n must be >= 2")
        # prepare initial history - for n>2, we need to create (n-1)-leng
        # We'll build the sentence incrementally; assume start_two are th
        sentence = ['<s>'] + [w.lower() for w in start_two]
        # continue generating until </s> or max length reached
        while len([w for w in sentence if w not in ('<s>')]) < max_words:
            if len(sentence) < self.n - 1:
                # pad with <s>
                context = tuple((['<s>'] * (self.n - 1 - len(sentence)) +
            else:
                context = tuple(sentence[-(self.n - 1):])
            # produce distribution over next tokens
            candidates = []
            probs = []
            for w in self.vocab:
                ngram = context + (w,)
                if smoothing == 'mle':
                    p = self.mle_prob(ngram)
                else:
                    p = self.laplace_prob(ngram, alpha=alpha)
                if p > 0:
                    candidates.append(w)
                    probs.append(p)
            if not candidates:
                # fallback: break
                break
            if sample:
                # normalize and sample
                total_p = sum(probs)
                probs = [p / total_p for p in probs]
                next_word = random.choices(candidates, weights=probs, k=1
            else:
                # argmax
                next_word = candidates[max(range(len(candidates)), key=la
            sentence.append(next_word)
            if next_word in ('.', '!', '?', '</s>'):
                break
        # strip leading <s> and trailing </s> if present
        out = [w for w in sentence if w != '<s>' and w != '</s>']
        return ' '.join(out)


    def perplexity(self, tokenized_sentences, smoothing='laplace', alpha=
        log_prob_sum = 0.0
```

```
        N = 0
        for sent in tokenized_sentences:
            for i in range(self.n - 1, len(sent)):
                context = tuple(sent[i-(self.n-1):i])
                word = sent[i]
                ngram = context + (word,)
                if smoothing == 'mle':
                    p = self.mle_prob(ngram)
                    # MLE may give 0 => perplexity infinite; handle by fa
                    if p == 0:
                        p = 1e-12
                else:
                    p = self.laplace_prob(ngram, alpha=alpha)
                log_prob_sum += math.log(p)
                N += 1
        if N == 0:
            return float('inf')
        avg_log_prob = log_prob_sum / N
        perplexity = math.exp(-avg_log_prob)
        return perplexity
```

```
# 5. Train / evaluate flow
def train_and_evaluate(raw_text):
    tokenized = preprocess_text(raw_text)
    # split train/test (e.g., 90/10)
    split_idx = int(0.9 * len(tokenized))
    train_sents = tokenized[:split_idx]
    test_sents = tokenized[split_idx:]

    results = {}

    # Use the small alpha for smoothing
    smoothing_alpha = 0.01

    for n in (2, 3, 4):
        print(f"Training {n}-gram model...")
        model = NGramModel(n)
        model.train(train_sents)

        # Use the new alpha for perplexity
        pp = model.perplexity(test_sents, smoothing='laplace', alpha=smoo
        results[n] = {'model': model, 'perplexity': pp}

        # Update the print statement to show the new alpha
        print(f"{n}-gram perplexity (laplace alpha={smoothing_alpha}): {p
    return results
```

```
# 6. Example usage
if __name__ == "__main__":
    download_nltk_data()
    raw = load_corpus(use_gutenberg=True, use_brown=True)
    results = train_and_evaluate(raw)
```

```
    # Use the same small alpha for generation
    smoothing_alpha = 0.01

    start = ("the", "man")
    for n in (2,3,4):
        m = results[n]['model']
        print(f"\n--- {n}-gram generated sentences for start: {' '.join(s
        for i in range(5):
            # Pass the same alpha to the generator
            sent = m.generate(start, max_words=12, smoothing='laplace', a
            print(f"{i+1}. {sent}")

    # print perplexities
    print("\nPerplexities summary:")
    for n in results:
        print(f"{n}-gram: {results[n]['perplexity']:.2f}")
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package gutenberg to /root/nltk_data...
[nltk_data]   Package gutenberg is already up-to-date!
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data]   Package brown is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
Training 2-gram model...
2-gram perplexity (laplace alpha=0.01): 1141.50
Training 3-gram model...
3-gram perplexity (laplace alpha=0.01): 8732.46
Training 4-gram model...
4-gram perplexity (laplace alpha=0.01): 28543.50

--- 2-gram generated sentences for start: the man ---
1. the man of some other eyes which acquitting listener penetrate rekindli
2. the man tramway numerically soiree helium coalesce solvency ligget disa
3. the man depredations brucellosis reformatory recognise groups that they
4. the man who deem bluebird nuns dwyer haase putty disking mennonites med
5. the man remember secured luckiest maget raising palaces kira focuses sh

--- 3-gram generated sentences for start: the man ---
1. the man capitalizing compatible worriedly inadvertence alarms duyvil la
2. the man eyeteeth uneconomic amines persuasions contrasts commit vanderv
3. the man dodgers villager keeeerist gaited susan misbranded undedicated
4. the man who radiating thwarted nonresident cycle club gyro dunk rightfu
5. the man impatient unrestricted jurisprudentially harvested thermoformed

--- 4-gram generated sentences for start: the man ---
1. the man patter infamy manifestation absorptions glances streightens muc
2. the man retranslated proudly prerogative vitro dogmatic theon fritz dir
3. the man boatmen avoids fleet seventies liberties wheels late plugugly c
4. the man erhart interconnectedness councilman canceling atoms circumscri
5. the man detractor trustfully kindnesses longstaple large pas baseline u

Perplexities summary:
2-gram: 1141.50
3-gram: 8732.46
4-gram: 28543.50
```

```
import json, os
os.makedirs('ngram_outputs', exist_ok=True)

smoothing_alpha = 0.01
start = ("the", "man")

# assuming `results` dict from your run and `start` variable exist
summary = {}
for n in results:
    summary[n] = {
        'perplexity': results[n]['perplexity'],
        'samples': []
    }
    model = results[n]['model']
    for i in range(10):
        summary[n]['samples'].append(model.generate(start, max_words=12, 

with open('ngram_outputs/summary.json', 'w') as f:
    json.dump(summary, f, indent=2)

# write a readable text file
with open('ngram_outputs/samples.txt', 'w') as f:
    for n in summary:
        f.write(f"=== {n}-gram (perplexity: {summary[n]['perplexity']:.2f
        for s in summary[n]['samples']:
            f.write(s + "\n")
        f.write("\n")
print('Saved to ngram_outputs/summary.json and samples.txt')
```

```
Saved to ngram_outputs/summary.json and samples.txt
```

```
!ls ngram_outputs
```

```
samples.txt  summary.json
```

```
!head -n 40 ngram_outputs/samples.txt
```

```
=== 2-gram (perplexity: 1141.50) ===
the man an paganism dramatization amass mcwhinney buzzing wet mrads shiple
the man most part headless unafraid exterminating profili hemphill edmov o
the man preston masseur stainless sabina unrealistic representatives coinc
the man
the man of god clurman yards unnerving overseers countries passing
the man had thought fanny
the man replied it is plain maritime dazzle released heatwole aspiring pro
the man beatrice cholesterol snowflakes viareggio justifying crystals furn
the man to side aircraft attains myosin concept exemplar dousman remnant p
the man is the true

=== 3-gram (perplexity: 8732.46) ===
the man throes eskimo compulsive boeing elder hir adolescent redevelopers 
the man said recommends subjectively workman ascribes lurking fulke thyrox
```

```
the man wheezed collapsible dinghy facades ethyl tricking findings compres
the man zhitzhakli revealment tarts catching begins showings backwater can
the man made disembodied psychology heralded pels collonaded drake kebob d
the man reckon histochemical awhile thermoplastic bawdy entirely member fr
the man nineties oops digest afterwards permit screening burlesque slacken
the man envelopes erasing troopships leaving shingles chatted hearty manag
the man commercants columbines show spraying piteous apollinaire reacting
the man muzo zurcher darius melodic chords musicians glumly spattered moul

=== 4-gram (perplexity: 28543.50) ===
the man gop watchdog bilge oozed ghouls despues correctly aurally furhmann
the man buttressed eliminates overprotective banshee noses arlington syndi
the man stuck lindskog heartbeat whitely dehumidified doubtfully reforms d
the man cochannel eummelihs loomed larry pumblechook jensen renamed spatte
the man cub understructure intents buick squash fulminate hinting janice b
the man daringly sylvan diplomat grief sinned seamless fitness lies adcock
the man pascataqua anita passengers ruthenium thirsty shouldering broglie
the man shell misgauged absorbed budlong islandia smooching israelites und
the man shifters convened feelings soured acoustical columbia footwork san
the man improvises stealing conquered baptismal baku foretold molecules up
```

Observations and Analysis

The generated sentences are now coherent and consist only of real words, proving the data cleaning was successful.

The key finding is in the perplexity scores:

2-gram: 1141.50

3-gram: 8732.46

4-gram: 28543.50

Our experiment shows that perplexity increased significantly with a higher n. This is not a bug, but a critical demonstration of data sparsity.

The 4-gram model is looking for 3-word contexts that are so specific they rarely (or never) appeared in our training data. When it encounters an unseen context, our simple "Add-k" (alpha=0.01) smoothing defaults to a tiny, fixed probability. This happens so often that the 4-gram model's overall predictive performance is far worse than the simpler, more robust 2-gram model.

Limitations

The primary limitation is our model's reliance on simple Add-k (Lidstone) smoothing. This technique is not effective for higher-order n-grams on a limited dataset, as it punishes the model too heavily for data sparsity.

The models only learn surface-level word co-occurrence, not semantic meaning.

Future Work

The most important next step is to implement a more advanced smoothing technique, such as Backoff or Kneser-Ney smoothing. These methods would "back off" to a 3-gram or 2-gram probability when a 4-gram context is not found, instead of defaulting to a tiny, fixed probability.

Experiment with neural language models (RNN/Transformer) for improved fluency.

Conclusion

The N-gram models successfully generated sentences given two starting words.
While the Bigram model produced more consistent results, the 4-gram model showed data sparsity issues.
The experiment demonstrates how probabilistic language models can capture local context in text generation.