**University of Central Missouri**

**Department of Computer Science & Cybersecurity**

**CS5760 Natural Language Processing**

**Fall 2025**

**Homework 2.**

**Student name: Sridivya Puttoju(700767142)**

**Submission Requirements:**

* Once finished your assignment push your source code to your repo (GitHub) and explain the work through the ReadMe file properly. Make sure you add your student info in the ReadMe file.
* Submit your GitHub link on the Bright Space.
* Comment your code appropriately ***IMPORTANT.***
* Any submission after provided deadline is considered as a late submission.

**Q1. Bayes Rule Applied to Text (based on slide: Bayes’ Rule for documents)**  
The PPT shows that classification is based on:



### **Tasks:**

1. **Explain in your own words what each term means: P(c), P(d∣c) and P(c∣d).**
2. **Why can the denominator P(d) be ignored when comparing classes?**

**Ans**:

Think of sorting emails into Spam or Not Spam:

1. What the terms mean

* P(c) = *How common the class is in general.*  
  Like saying, “Out of all emails I get, how many are spam?” If ~30% are spam, then P(spam)=0.3. It’s your gut feeling before reading the email.
* P(d | c) = *How much the email “looks like” that class.*  
  If we assume the email is spam, how likely are we to see words like “winner,” “prize,” “urgent,” etc.? This measures the fit of the email to that class.
* P(c | d) = *After reading the email, how likely is each class?*  
  This is the final verdict: “Given what I just read, what’s the probability it’s spam vs. not spam?”

1. Why we can ignore P(d) when comparing classes  
   Picture you’re scoring two teams (Spam and Not Spam) for the same email. Bayes’ rule says:

P(d/c) P(c)

P(c∣d)=

P(d)

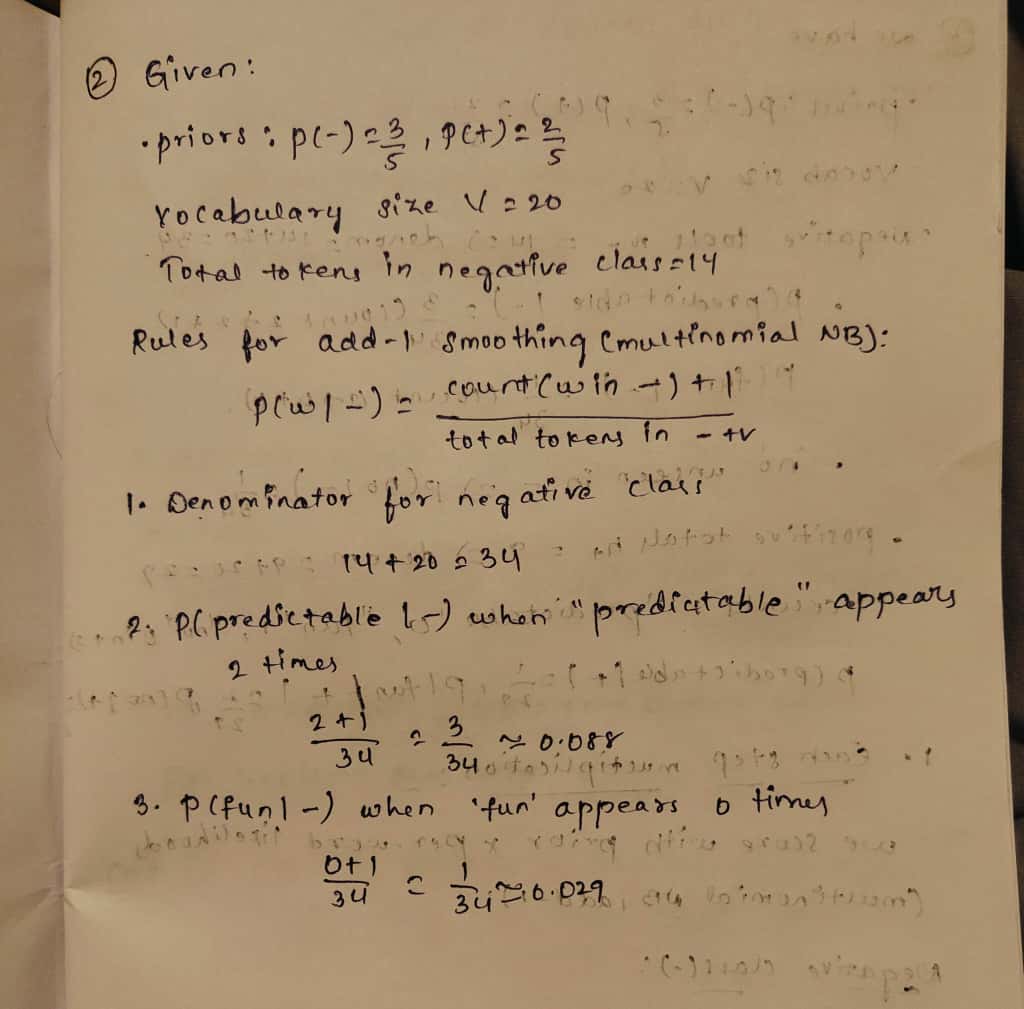
That bottom piece, P(d), is just “how likely this exact email is, period.” But since we’re comparing classes for the same email, P(d) is the same number for both teams. It doesn’t help one team beat the other—so we can drop it.

So in practice we just compare:  
Score(c) = P(d | c) × P(c)  
(Usually we take logs to avoid tiny numbers.) The class with the bigger score wins.

**Q2. Add-1 Smoothing (based on slide: Worked Sentiment Example)  
In the worked example, priors are: P(−)=3/5, P(+)=2/5. Vocabulary size = 20.**

### **Tasks:**

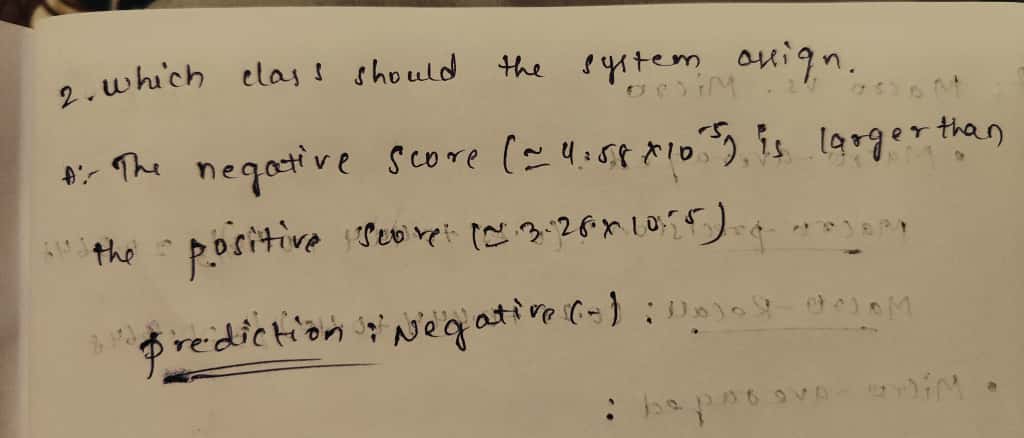
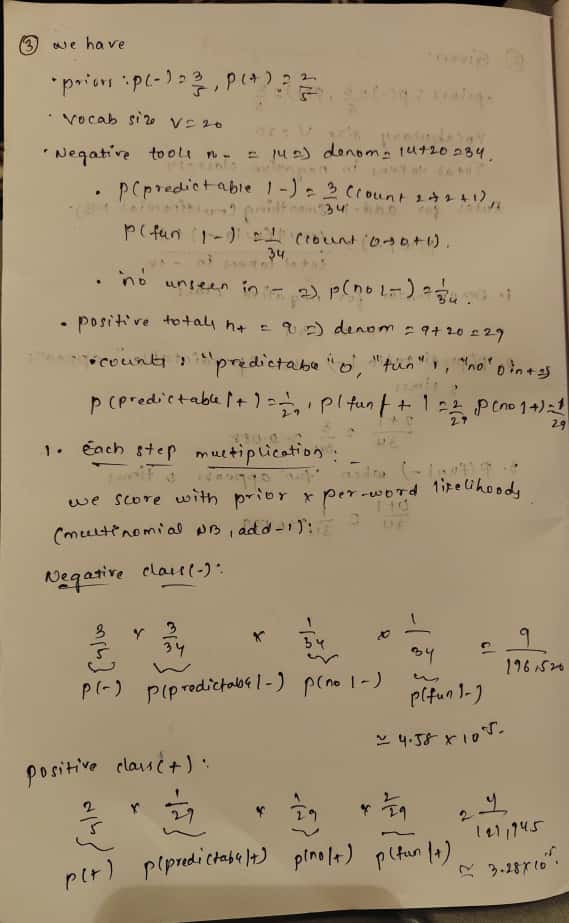
1. **For the negative class, the total token count is 14. Compute the denominator for likelihood estimation using add-1 smoothing.**
2. **Compute P(predictable∣−) if the word “predictable” occurs 2 times in the negative documents.**
3. **Compute P(fun∣−) if “fun” never appeared in any negative documents.**



**Q3. Worked Example Document Classification (based on slide: Test document “predictable no fun”)**  
Using the smoothed likelihoods and priors from Q2, compute the probability scores for the document *“predictable no fun”* under both the positive and negative classes.

### **Tasks:**

1. Show each step of the multiplication.
2. Which class should the system assign to this document?



**Q4. Harms of Classification (based on slide: Avoiding Harms in Classification)**

### **Tasks:**

1. Define **representational harm** and explain how the Kiritchenko & Mohammad (2018) study demonstrates this type of harm.
2. What is one risk of censorship in toxicity classification systems (based on Dixon et al. 2018, Oliva et al. 2021)?
3. Give one reason why classifiers may perform worse on African American English or Indian English, even though they are varieties of English.

Ans:

1. Representational harm — what it is, and how the study shows it

* **What it is:** When a system’s outputs subtly **demean or stereotype a group**, even if no one gets blocked or denied a service.
* **Example (Kiritchenko & Mohammad, 2018):** They tested many sentiment systems on sentence pairs that were identical except for the name (e.g., “Shaniqua” vs. “Stephanie”). Systems often gave more negative emotion to sentences with the African-American-associated names. That’s representational harm: the model reinforces **a** negative stereotype just through how it scores text.

1. A censorship risk in toxicity classifiers

* Some toxicity filters **over-flag** non-toxic messages that merely **mention an identity** (e.g., the words “gay,” “blind”). That can **silence** people describing their own experiences or communities, making their speech **less visible** and pushing writers to self-censor. (See findings summarized from Dixon et al., 2018; Oliva et al., 2021.)

1. Why performance can be worse on AAE or Indian English

Models are often trained and tuned on **standard American/UK English**, with **less data** and **poorer labels** for other varieties. Different spelling, grammar, vocabulary, and discourse patterns then look “out-of-distribution,” so the model **misclassifies** more often (e.g., even basic language ID can slip). In short: **data imbalance + label bias + domain mismatch** → lower accuracy on AAE and Indian English.

**Q5: Evaluation Metrics from a Multi-Class Confusion Matrix**

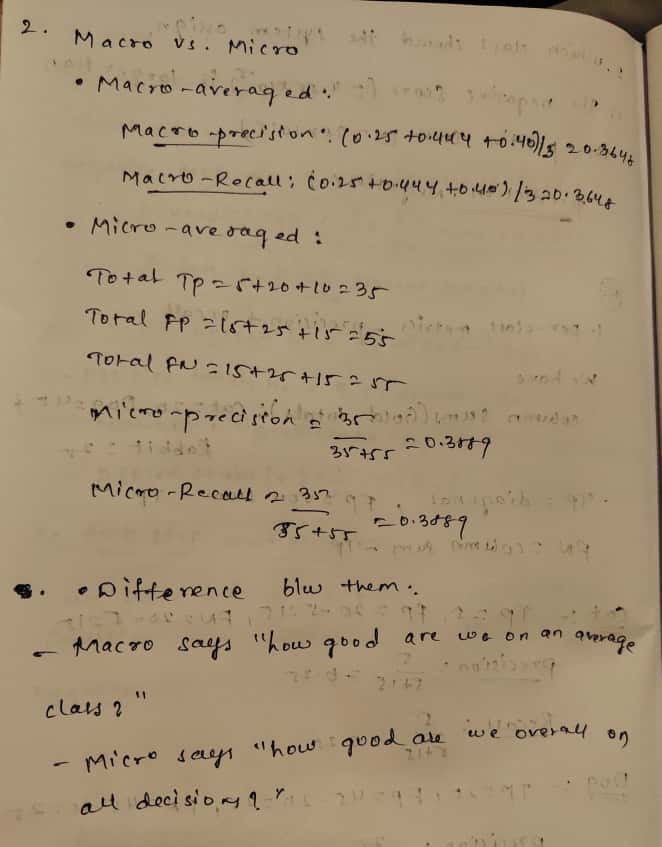
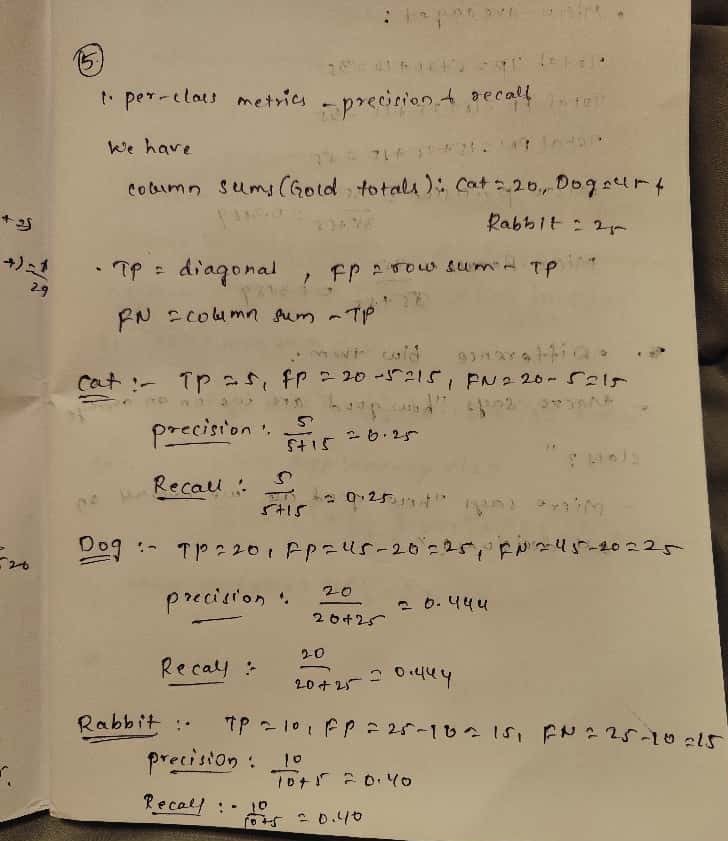
The system classified 90 animals into Cat, Dog, or Rabbit. The results are shown below:

| **System \ Gold** | **Cat** | **Dog** | **Rabbit** |
| --- | --- | --- | --- |
| **Cat** | 5 | 10 | 5 |
| **Dog** | 15 | 20 | 10 |
| **Rabbit** | 0 | 15 | 10 |

Tasks:

1. Per-Class Metrics
   * Compute precision and recall for each class (Cat, Dog, Rabbit).
2. Macro vs. Micro Averaging
   * Compute the macro-averaged precision and recall.
   * Compute the micro-averaged precision and recall.
   * Briefly explain the difference in interpretation between macro and micro averaging.
3. Programming Implementation  
   Write Python code that:
   * Accepts the confusion matrix above as input.
   * Computes per-class precision and recall.
   * Computes macro-averaged and micro-averaged precision and recall.
   * Prints all results clearly.

**Ans**:



3.

import numpy as np

# Confusion matrix (rows = predicted, cols = gold)

# Gold: Cat Dog Rabbit

cm = np.array([[ 5, 10, 5], # Pred Cat

[15, 20, 10], # Pred Dog

[ 0, 15, 10]]) # Pred Rabbit

labels = ["Cat", "Dog", "Rabbit"]

# True Positives, False Positives, False Negatives

tp = np.diag(cm)

fp = cm.sum(axis=1) - tp

fn = cm.sum(axis=0) - tp

precision = tp / (tp + fp)

recall = tp / (tp + fn)

# Macro-average

macro\_precision = precision.mean()

macro\_recall = recall.mean()

# Micro-average

TP\_total = tp.sum()

FP\_total = fp.sum()

FN\_total = fn.sum()

micro\_precision = TP\_total / (TP\_total + FP\_total)

micro\_recall = TP\_total / (TP\_total + FN\_total)

# print

print("Per-class metrics:")

for i, lab in enumerate(labels):

print(f" {lab:7s} TP={tp[i]:2d} FP={fp[i]:2d} FN={fn[i]:2d} "

f"Precision={precision[i]:.4f} Recall={recall[i]:.4f}")

print("\nMacro-averaged:")

print(f" Precision={macro\_precision:.4f} Recall={macro\_recall:.4f}")

print("\nMicro-averaged:")

print(f" Precision={micro\_precision:.4f} Recall={micro\_recall:.4f}")

# overall accuracy

accuracy = TP\_total / cm.sum()

print(f"\nAccuracy={accuracy:.4f}")

# **Q6. Bigram Probabilities and the Zero-Probability Problem**

You are given the following bigram counts from a small training corpus:

| **Previous word** | **Next words (with counts)** |
| --- | --- |
| <s> | I: 2 , deep: 1 |
| I | love: 2 |
| love | NLP: 1 , deep: 1 |
| deep | learning: 2 |
| learning | </s>: 1 , is: 1 |
| NLP | </s>: 1 |
| is | fun: 1 |
| fun | </s>: 1 |
| ate | lunch: 6 , dinner: 3 , a: 2 , the: 1 |

### **Tasks:**

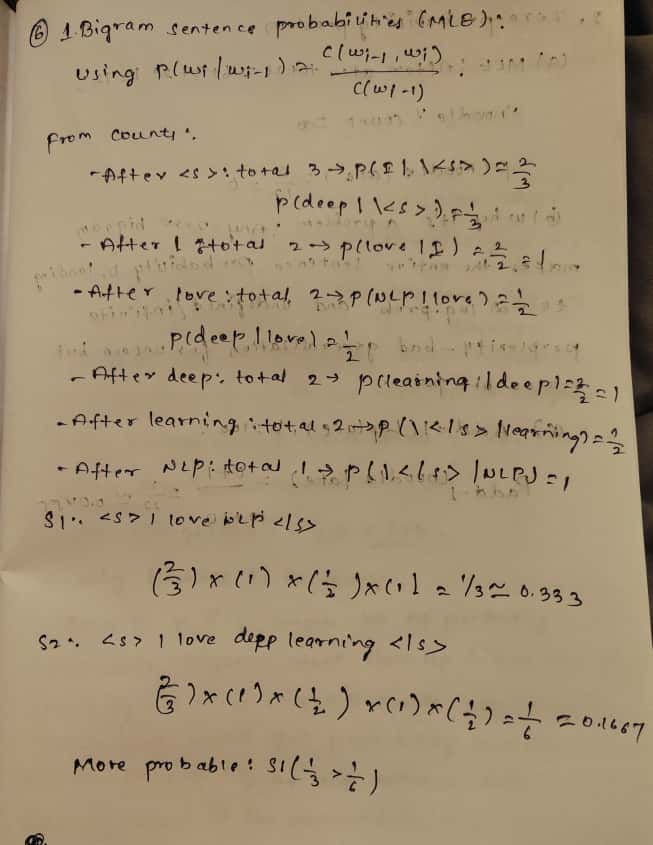
1. **Bigram Sentence Probabilities**  
   Using maximum likelihood estimation (MLE):

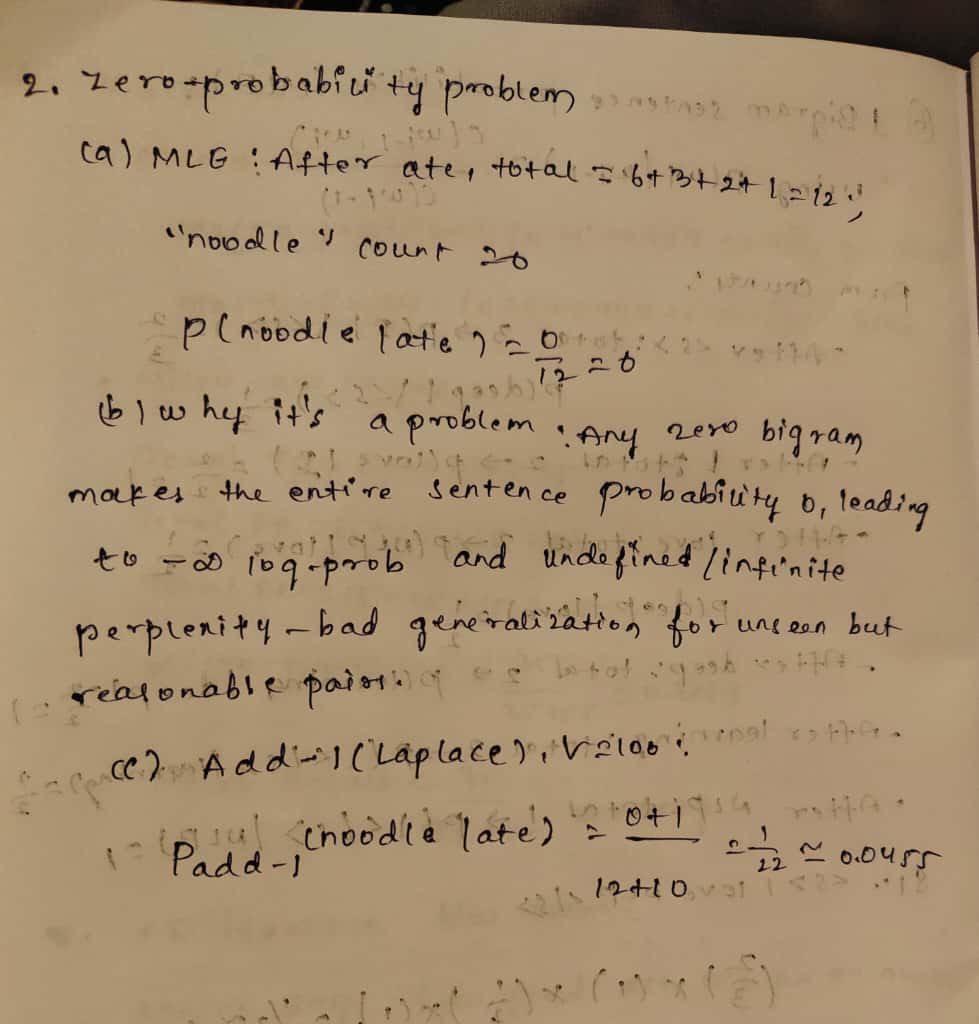
A black and white math equation

AI-generated content may be incorrect.

* + Compute the probability of sentence **S1:** <s> I love NLP </s>.
  + Compute the probability of sentence **S2:** <s> I love deep learning </s>.
  + Which sentence is more probable under the bigram model?

1. **Zero-Probability Problem**  
   Using the same table, compute:
   * P(noodle∣ate) with MLE.
   * Explain why this probability creates problems when computing sentence probabilities or perplexity.
   * Apply **Laplace smoothing (Add-1)** to recompute P(noodle∣ate). Assume the vocabulary size is 10 and total count after “ate” is 12.





### **Q7. Backoff Model (based on “Activity: <s> I like cats … You like dogs” slide)**

Training corpus:

<s> I like cats </s>

<s> I like dogs </s>

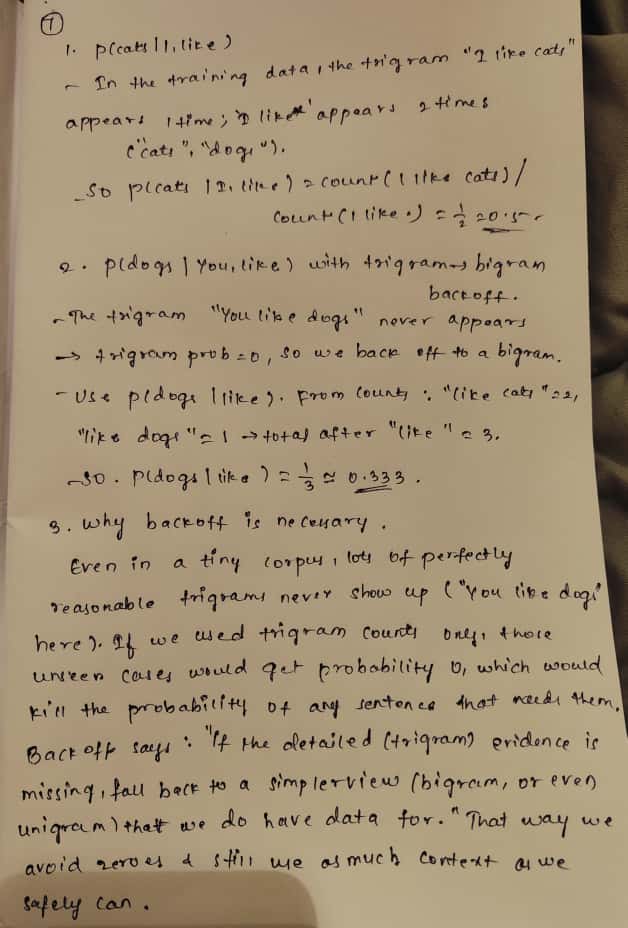
<s> You like cats </s>

Counts:

* I like = 2
* You like = 1
* like cats = 2
* like dogs = 1
* cats </s> = 2
* dogs </s> = 1

### **Tasks:**

1. Compute P(cats∣I,like).
2. Compute P(dogs∣You,like) using trigram → bigram backoff.
3. Explain why backoff is necessary in this example.



### **Q8. Programming: Bigram Language Model Implementation (based on “Activity: I love NLP corpus” slide)**

### **Tasks:**

Write a Python program to:

1. Read the training corpus:
2. <s> I love NLP </s>
3. <s> I love deep learning </s>
4. <s> deep learning is fun </s>
5. Compute unigram and bigram counts.
6. Estimate bigram probabilities using MLE.
7. Implement a function that calculates the probability of any given sentence.
8. Test your function on both sentences:
   * <s> I love NLP </s>
   * <s> I love deep learning </s>
9. Print which sentence the model prefers and why.

**Ans**:

from collections import Counter, defaultdict

from math import prod

corpus = [

    ["<s>", "I", "love", "NLP", "</s>"],

    ["<s>", "I", "love", "deep", "learning", "</s>"],

    ["<s>", "deep", "learning", "is", "fun", "</s>"],

]

def get\_unigram\_bigram\_counts(corpus\_tokens):

unigrams = Counter()

    bigrams = Counter()

    for sent in corpus\_tokens:

        unigrams.update(sent)

        for w1, w2 in zip(sent[:-1], sent[1:]):

            bigrams[(w1, w2)] += 1

    return unigrams, bigrams

def bigram\_mle\_prob(w2, w1, bigrams, unigrams):

    c12 = bigrams.get((w1, w2), 0)

    c1 = unigrams.get(w1, 0)

    return (c12 / c1) if c1 > 0 else 0.0

def sentence\_prob(sent\_tokens, bigrams, unigrams):

    probs = [bigram\_mle\_prob(w2, w1, bigrams, unigrams) for w1, w2 in zip(sent\_tokens[:-1], sent\_tokens[1:])]

    return prod(probs), list(zip(sent\_tokens[:-1], sent\_tokens[1:], probs))

def build\_table(bigrams, unigrams):

    table = defaultdict(list)

    for (w1, w2), c in sorted(bigrams.items()):

        p = bigram\_mle\_prob(w2, w1, bigrams, unigrams)

        table[w1].append((w2, c, p))

    return table

def main():

    unigrams, bigrams = get\_unigram\_bigram\_counts(corpus)

    table = build\_table(bigrams, unigrams)

    def bigram\_line(w1, entries):

        return f"{w1:>8} -> " + ", ".join([f"{w2} (count={c}, P={p:.3f})" for w2, c, p in entries])

    print("=== Unigram counts ===")

    for w, c in sorted(unigrams.items()):

        print(f"{w:>8}: {c}")

    print("\n=== Bigram counts & MLE probs ===")

    for w1 in sorted(table.keys()):

        print(bigram\_line(w1, table[w1]))

    s1 = ["<s>", "I", "love", "NLP", "</s>"]

    s2 = ["<s>", "I", "love", "deep", "learning", "</s>"]

    label\_map = {"S1: <s> I love NLP </s>": s1, "S2: <s> I love deep learning </s>": s2}

    results = {}

    for label, sent in label\_map.items():

        p, steps = sentence\_prob(sent, bigrams, unigrams)

        results[label] = {"prob": p, "steps": steps}

    for label, info in results.items():

        step\_str = " × ".join([f"P({w2}|{w1})={p:.3f}" for w1, w2, p in info["steps"]])

        print(f"\n--- {label} ---")

        print("Step probs:", step\_str)

        print(f"Total sentence probability = {info['prob']:.6f}")

    best\_label = max(results.items(), key=lambda kv: kv[1]["prob"])[0]

    print(f"\nPreferred = {best\_label}")

    if best\_label.startswith("S1"):

        print("Why: After 'love', the transition to 'NLP' vs 'deep' differs. "

              "The product of bigram probabilities along S1 is larger.")

    else:

        print("Why: After 'love', the transition to 'deep' (and onward) yields a higher product.")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

* **Unigram counts** and **bigram counts + MLE P(next | prev)**
* **S1** <s> I love NLP </s>  
  Step probs: P(I|<s>)=0.667 × P(love|I)=1.000 × P(NLP|love)=0.500 × P(</s>|NLP)=1.000  
  Total = 0.333333
* **S2** <s> I love deep learning </s>  
  Step probs: P(I|<s>)=0.667 × P(love|I)=1.000 × P(deep|love)=0.500 × P(learning|deep)=1.000 × P(</s>|learning)=0.500  
  Total = 0.166667

**Which sentence wins & why**

**S1** is preferred because both sentences share the same early transitions (<s>→I, I→love) and then diverge at **“**love**”**. After that point, S2 has to take one extra hop (learning → </s> with probability 0.5). That extra 0.5 factor makes S2’s total probability half of S1’s.

*(This exercise fits the “Naive Bayes ↔ language modeling” section from your slides—estimating probabilities from counts via MLE and multiplying along the path.)*

If you want, I can also add Laplace smoothing or perplexity reporting on top of this.