University of Central Missouri

Department of Computer Science & Cybersecurity

CS5760 Natural Language Processing

Fall 2025

Homework 2.

Student name:Duggineni Sesha Rao

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:

$$c_{MAP} = \arg\max_{c \in C} P(c)\, P(d \mid c)$$

Tasks:

1. Explain in your own words what each term means: P(c), P(d|c) and P(c|d).

   We are trying to classify a document d into one of several possible classes c∈C

Where:

- **P(c)** → *Prior Probability of Class*
  - How likely class ccc is **before** seeing the document.
  - Example: If 70% of emails are spam, then P(spam)=0.7P(\text{spam}) = 0.7P(spam)=0.7.
- **P(d|c)** → *Likelihood*
  - Probability of seeing document ddd **if it really belongs to class ccc**.
  - Example: Probability of seeing certain words (like "discount", "offer") if the email is spam.
- **P(c|d)** → *Posterior Probability*
  - Probability that document ddd belongs to class ccc **after** looking at its contents.
  - This is what we ultimately want to maximize (choose the most like

2. Why can the denominator P(d) be ignored when comparing classes?

   Bayes' Rule says:

   p(c/d)=p(c)p(d/c)/p(d)

   ? **P(d)** = Probability of the document regardless of class (same for all classes).

 When we are just comparing which class gives the highest probability, P(d)P(d)P(d) does not change between classes — it is a constant.

 So we can ignore it and just maximize

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.

   Given:

- **P(−) = 3/5 , P(+) = 2/5** (not needed for this part)
- **Vocabulary size (V) = 20**
- **Total token count for negative class = 14**

Denominator for likelihood estimation

Add-1 smoothing denominator =

total tokens in negative class+V=14+20=34

2. Compute P(predictable|−) if the word "predictable" occurs 2 times in the negative documents.

   Word count of "predictable" = 2
   P(predictable | −) = (2 + 1) / 34 = **3/34 ≈ 0.0882**

3. Compute P(fun|−) if "fun" never appeared in any negative documents.

   Word count of "fun" = 0
   P(fun | −) = (0 + 1) / 34 = **1/34 ≈ 0.0294**

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.

   Negative class (using P(no|−) = (0+1)/34 = 1/34):
   P(−) × P(predictable|−) × P(no|−) × P(fun|−)
   = (3/5) × (3/34) × (1/34) × (1/34)

   Stepwise:
   (3/5) × (3/34) = 9/170 ≈ 0.05294117647
   (9/170) × (1/34) = 9/5780 ≈ 0.00155702811
   (9/5780) × (1/34) = 9/196520 ≈ **0.0000457969**

Score(−) = 9 / 196520 ≈ 0.0000457969

Positive class (example assumption: N_+ = 14 and all three words unseen in + ⇒ each (0+1)/34 = 1/34):

P(+) × P(predictable|+) × P(no|+) × P(fun|+)

= (2/5) × (1/34) × (1/34) × (1/34)

Stepwise:
(2/5) × (1/34) = 2/170 = 1/85 ≈ 0.01176470588
(1/85) × (1/34) = 1/2890 ≈ 0.00034602076
(1/2890) × (1/34) = 1/98260 ≈ 0.0000101780

Score(+) = 1 / 98,260 ≈ 0.0000101780

2. Which class should the system assign to this document?

Score(−) ≈ 0.0000457969 > Score(+) ≈ 0.0000101780 → **Assign to NEGATIVE class.**

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.

   Representational harm happens when a system reinforces stereotypes or unfairly represents a group.
   Kiritchenko & Mohammad (2018) showed that sentiment and emotion lexicons gave systematically **more negative scores** to words related to certain demographic groups (e.g., African American names), which can reinforce bias and misrepresent those groups.

2. What is one risk of censorship in toxicity classification systems (based on Dixon et al. 2018, Oliva et al. 2021)?

   A key risk is **over-blocking or silencing marginalized voices.**
   Dixon et al. (2018) and Oliva et al. (2021) showed that toxicity classifiers often flag reclaimed slurs or community-specific language as toxic, causing harmless speech (e.g., LGBTQ+ discussions) to be removed unfairly.

3. Give one reason why classifiers may perform worse on African American English or Indian English, even though they are varieties of English.

Because these varieties have **different vocabulary, grammar, and spelling patterns** compared to Standard American/British English.

If the training data mostly contains Standard English, the model sees fewer examples of AAE or Indian English, leading to **lower accuracy** and more false positives/negatives.

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).

     Cat:

     TP = 5

     FP = predicted Cat but not actually Cat = 10 + 5 = 15

     FN = actually Cat but not predicted Cat = 15 + 0 = 15

     Precision(Cat) = 5 / (5+15) = **0.25**
     Recall(Cat) = 5 / (5+15) = **0.25**

     Dog:

     TP = 20

     FP = predicted Dog but not actually Dog = 15 + 10 = 25

     FN = actually Dog but not predicted Dog = 10 + 15 = 25

     Precision(Dog) = 20 / (20+25) = 0.4444
     Recall(Dog) = 20 / (20+25) = 0.4444

     Rabbit:

     TP = 10

     FP = predicted Rabbit but not actually Rabbit = 0 + 15 = 15

FN = actually Rabbit but not predicted Rabbit = 5 + 10 = 15

Precision(Rabbit) = 10 / (10+15) = 0.4
Recall(Rabbit) = 10 / (10+15) = 0.4

2. Macro vs. Micro Averaging
   - Compute the macro-averaged precision and recall.

     Macro-Averaged Precision & Recall

     Macro = average of per-class values

     Macro-Precision = (0.25 + 0.4444 + 0.4) / 3 = **0.3648**
     Macro-Recall = (0.25 + 0.4444 + 0.4) / 3 = **0.3648**

   - Compute the micro-averaged precision and recall.

     Micro-Averaged Precision & Recall

     Micro = compute global TP, FP, FN across all classes

     Total TP = 5 + 20 + 10 = **35**

     Total FP = 15 + 25 + 15 = **55**

     Total FN = 15 + 25 + 15 = **55**

     Micro-Precision = TP / (TP+FP) = 35 / (35+55) = 35/90 = **0.3889**
     Micro-Recall = TP / (TP+FN) = 35 / (35+55) = 35/90 = **0.3889**

   - Briefly explain the difference in interpretation between macro and micro averaging.

   Macro averaging gives **equal weight to each class**, treating all classes as equally important regardless of frequency.

   Micro averaging gives **equal weight to each individual prediction**, so frequent classes dominate the score.

3. Programming Implementation
   Write Python code that:
   1. Accepts the confusion matrix above as input.
   2. Computes per-class precision and recall.
   3. Computes macro-averaged and micro-averaged precision and recall.
   4. Prints all results clearly.

```python
import numpy as np

# Confusion matrix
conf_matrix = np.array([
    [5, 10, 5],    # Predicted Cat
    [15, 20, 10], # Predicted Dog
    [0, 15, 10]    # Predicted Rabbit
])

# Step 1: Per-class precision and recall
num_classes = conf_matrix.shape[0]
precisions = []
recalls = []

for i in range(num_classes):
    TP = conf_matrix[i, i]
    FP = conf_matrix[i, :].sum() - TP
    FN = conf_matrix[:, i].sum() - TP

    precision = TP / (TP + FP) if (TP + FP) > 0 else 0
    recall = TP / (TP + FN) if (TP + FN) > 0 else 0

    precisions.append(precision)
    recalls.append(recall)

    print(f"Class {i} -> Precision: {precision:.4f}, Recall: {recall:.4f}")
```

```python
# Step 2: Macro averages
macro_precision = np.mean(precisions)
macro_recall = np.mean(recalls)

# Step 3: Micro averages
TP_total = np.trace(conf_matrix)
FP_total = conf_matrix.sum(axis=1).sum() - TP_total
FN_total = conf_matrix.sum(axis=0).sum() - TP_total

micro_precision = TP_total / (TP_total + FP_total)
micro_recall = TP_total / (TP_total + FN_total)

print("\nMacro-Averaged Precision:", round(macro_precision, 4))
print("Macro-Averaged Recall:", round(macro_recall, 4))
print("Micro-Averaged Precision:", round(micro_precision, 4))
print("Micro-Averaged Recall:", round(micro_recall, 4))
```

```
Class 0 -> Precision: 0.2500, Recall: 0.2500
Class 1 -> Precision: 0.4444, Recall: 0.4444
Class 2 -> Precision: 0.4000, Recall: 0.4000

Macro-Averaged Precision: 0.3648
Macro-Averaged Recall: 0.3648
Micro-Averaged Precision: 0.3889
Micro-Averaged Recall: 0.3889
```

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):

$$P(w_i \mid w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}$$

   - Compute the probability of sentence S1: <s> I love NLP </s>.

     S1 = <s> I love NLP </s>

     P(<s>→I) = 2 / 3
     P(I→love) = 2 / 2 = 1
     P(love→NLP) = 1 / 2
     P(NLP→</s>) = 1 / 1 = 1

Multiply (step by step):
(2/3) × 1 = 2/3
(2/3) × (1/2) = 2/6 = 1/3
(1/3) × 1 = 1/3

S1 probability = 1/3 ≈ 0.3333333333

- Compute the probability of sentence S2: <s> I love deep learning </s>.

S2 = <s> I love deep learning </s>

P(<s>→I) = 2 / 3
P(I→love) = 1
P(love→deep) = 1 / 2
P(deep→learning) = 2 / 2 = 1
P(learning→</s>) = 1 / 2

Multiply (step by step):
(2/3) × 1 = 2/3
(2/3) × (1/2) = 2/6 = 1/3
(1/3) × 1 = 1/3
(1/3) × (1/2) = 1/6

S2 probability = 1/6 ≈ 0.1666666667

- Which sentence is more probable under the bigram model?

S1 (1/3 ≈ 0.3333) is more probable than S2 (1/6 ≈ 0.1667).

2. Zero-Probability Problem
   Using the same table, compute:
   - P(noodle|ate) with MLE.

P(noodle | ate) (MLE):
C(ate, noodle) = 0
C(ate) = 12
P(noodle | ate) = 0 / 12 = **0**

- Explain why this probability creates problems when computing sentence probabilities or perplexity.

 Multiplying probabilities to get a sentence probability will yield **0** if any single conditional probability is 0. That makes unseen events cause entire sentence probabilities to be zero and yields undefined/infinite perplexity — preventing meaningful ranking or evaluation.

- Apply Laplace smoothing (Add-1) to recompute P(noodle|ate). Assume the vocabulary size is 10 and total count after "ate" is 12.

  Laplace (Add-1) smoothing (V = 10, C(ate) = 12):
  P(noodle | ate) = (0 + 1) / (12 + 10) = 1 / 22 ≈ **0.0454545455**

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).

   C(I like cats)=1 (occurs in sentence 1)

   C(I like)=2C

   Use trigram MLE:
   P(cats | I, like) = C(I like cats) / C(I like)= 1 / 2= **0.5**

2. Compute P(dogs|You,like) using trigram → bigram backoff.

   C(You like dogs) = 0 → back off to bigram.

   P(dogs | like) = C(like dogs) / C(like)
   C(like) = C(like cats) + C(like dogs) = 2 + 1 = 3
   P(dogs | like) = 1 / 3 = **0.3333**

3. Explain why backoff is necessary in this example.

Trigram counts are sparse in a small corpus, giving many zeros.
Backoff avoids zero probabilities by using lower-order n-grams (bigram or unigram), producing non-zero and more reliable probability estimates.

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 wh

```python
from collections import Counter

# ---------------------
# 1. Read the training corpus
# ---------------------
corpus = [
    ["<s>", "I", "love", "NLP", "</s>"],
    ["<s>", "I", "love", "deep", "learning", "</s>"],
    ["<s>", "deep", "learning", "is", "fun", "</s>"]
]

# ---------------------
# 2. Compute unigram counts
# ---------------------
unigram_counts = Counter()
for sentence in corpus:
    for word in sentence:
        unigram_counts[word] += 1

# ---------------------
# 3. Compute bigram counts
# ---------------------
bigram_counts = Counter()
for sentence in corpus:
    for i in range(len(sentence) - 1):
        bigram = (sentence[i], sentence[i+1])
        bigram_counts[bigram] += 1
```

y.

```python
# -----------------------
# 4. Estimate bigram probabilities (MLE)
# -----------------------
bigram_probs = {}
for (w1, w2), count in bigram_counts.items():
    bigram_probs[(w1, w2)] = count / unigram_counts[w1]

# -----------------------
# 5. Function to calculate probability of any given sentence
# -----------------------
def sentence_probability(sentence_tokens):
    prob = 1.0
    for i in range(len(sentence_tokens) - 1):
        w1, w2 = sentence_tokens[i], sentence_tokens[i+1]
        prob *= bigram_probs.get((w1, w2), 0)
    return prob

# -----------------------
# 6. Test the function on the two sentences
# -----------------------
s1 = ["<s>", "I", "love", "NLP", "</s>"]
s2 = ["<s>", "I", "love", "deep", "learning", "</s>"]

p_s1 = sentence_probability(s1)
p_s2 = sentence_probability(s2)

# -----------------------
# 7. Print unigram and bigram counts
```

```python
# -----------------------
print("Unigram Counts:", unigram_counts)
print("Bigram Counts:", bigram_counts)

# -----------------------
# 8. Print bigram probabilities and sentence probabilities
# -----------------------
print("\nBigram Probabilities (MLE):")
for (w1, w2), p in bigram_probs.items():
    print(f"P({w2}|{w1}) = {p:.4f}")

print("\nSentence Probabilities:")
print(f"P(<s> I love NLP </s>) = {p_s1:.6f}")
print(f"P(<s> I love deep learning </s>) = {p_s2:.6f}")

# -----------------------
# 9. Print which sentence is preferred and why
# -----------------------
if p_s1 > p_s2:
    print("\nModel prefers: <s> I love NLP </s>")
else:
    print("\nModel prefers: <s> I love deep learning </s>")

print("Reason: The preferred sentence has a higher product of bigram probabilities,")
print("meaning it is considered more likely according to the patterns seen in the training data.")
```

```
Unigram Counts: Counter({'<s>': 3, '</s>': 3, 'I': 2, 'love': 2, 'deep': 2, 'learning': 2, 'NLP': 1, 'is': 1, 'fun': 1})
Bigram Counts: Counter({('<s>', 'I'): 2, ('I', 'love'): 2, ('deep', 'learning'): 2, ('love', 'NLP'): 1, ('NLP', '</s>'): 1, ('love', 'deep')
```

```
Bigram Probabilities (MLE):
P(I|<s>) = 0.6667
P(love|I) = 1.0000
P(NLP|love) = 0.5000
P(</s>|NLP) = 1.0000
P(deep|love) = 0.5000
P(learning|deep) = 1.0000
P(</s>|learning) = 0.5000
P(deep|<s>) = 0.3333
P(is|learning) = 0.5000
P(fun|is) = 1.0000
P(</s>|fun) = 1.0000

Sentence Probabilities:
P(<s> I love NLP </s>) = 0.333333
P(<s> I love deep learning </s>) = 0.166667

Model prefers: <s> I love NLP </s>
Reason: The preferred sentence has a higher product of bigram probabilities,
meaning it is considered more likely according to the patterns seen in the training data.
```