University of Central Missouri

Department of Computer Science & Cybersecurity

CS5760 Natural Language Processing

Spring 2026

Homework 2.

Student name: Nikhilesh Katakam

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.

Part I. Writing Calculation

Q1. 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?

   Answer:

   Task1: Probability calculation

   Class probability = prior × probability of each word in that class

   Start with the prior probability of the positive class:

   P (Positive)

   0.5 P(Positive)=0.5

   Now multiply it with the likelihoods of each word:

   . Probability of predictable given Positive = $\frac{1}{9}$

   . Probability of no given Positive = $\frac{1}{9}$

   . Probability of fun given Positive = $\frac{2}{9}$

So, we multiply everything:

$$= 0.5 \times \frac{1}{9} \times \frac{1}{9} \times \frac{2}{9}$$

First multiply the word probabilities:

$$= \frac{1}{9} \times \frac{1}{9} \times \frac{2}{9} \times \frac{1}{9} = \frac{2}{729}$$

Now multiply by the prior:

$$= 0.5 \times \frac{2}{729} = \frac{1}{729}$$

So, the **Positive class score** is:

$$\frac{1}{729}$$

. For the negative class:

$P(\text{Negative}) = 0.5$

Now multiply with the word likelihoods:

- Probability of **predictable** given Negative = $\frac{2}{8}$
- Probability of **no** given Negative = $\frac{1}{8}$
- Probability of **fun** given Negative = $\frac{1}{8}$
-
- Multiply them step by step:

$$= 0.5 \times \frac{2}{8} \times \frac{1}{8} \times \frac{1}{8}$$

- First the word probabilities:

$$= \frac{2}{8} \times \frac{1}{8} \times \frac{1}{8} = \frac{2}{512}$$

- Now multiply by the prior:

$$= 0.5 \times \frac{2}{512} = \frac{1}{512}$$

- So, the **Negative class score** is:

$$= \frac{1}{512}$$

Task 2: Final classification

Now we compare the two scores:

. Positive score $= \frac{1}{729}$

. Negative score $= \frac{1}{512}$

Since the Document is more likely under the Negative class.

Q2. 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.

   Answer:

   Representational harm occurs when a system reflects or spreads biased or stereotypical views about certain groups, even if it is not directly making decisions that affect them.

   In the study by Kiritchenko & Mohammad (2018), the researchers found that many emotion and sentiment lexicons link different genders and racial groups with different emotions. For instance, words associated with women were more frequently connected to emotions like sadness or fear, while words associated with men were more often linked to anger or strength. This shows how NLP systems can unintentionally reinforce social stereotypes, which is an example of representational harm.

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

   Answer: A key risk is that individuals from marginalized groups may end up being unfairly silenced.

   Research such as Dixon et al. (2018) and Oliva et al. (2021) shows that toxicity detection systems often label content as toxic simply because it contains identity-related words or

non-standard language. This can cause harmless, self-expressive, or political speech to be incorrectly flagged and removed, resulting in over-censorship and limiting the voices of certain communities

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

   Answer:

   Classifiers often show lower accuracy because they are primarily trained on standard varieties of English.

   African American English and Indian English follow different grammatical patterns, vocabulary choices, and styles of expression. Since these language varieties are often underrepresented in training datasets, models may misinterpret them or classify them incorrectly. Although they are legitimate and meaningful forms of English, insufficient representation in the data leads to higher error rates and biased outcomes.

Q3. 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>.
   - Compute the probability of sentence S2: <s> I love deep learning </s>.
   - Which sentence is more probable under the bigram model?

     Answer:

     Step 1: First, compute total counts for each **previous word**

. C(<s>) = 2 + 1 = 3

. C(I) = 2

. C(love) = 1 + 1 = 2

. C(deep) = 2

.C(learning) = 1 + 1 = 2

.C(NLP) = 1

i. Sentence S1

<S> I Love NLP </S>

Break into bigrams:

1. **<s> - I**
2. **I – Love**
3. **Love – NLP**
4. **NLP - </s>**

Probability

- **P(I|<s>) = $\frac{2}{3}$**
- $\boldsymbol{P(love \mid I) = \frac{2}{2} = 1}$

- $P(NLP \mid love) = \frac{1}{2}$
- $P(</s> \mid NLP) = \frac{1}{1} = 1$

Multiply them

$$\mathbf{P(S1)} = \frac{2}{3} \times 1 \times \frac{1}{2} \times 1 = \frac{1}{3}$$

ii. Sentence 2

<S> I love deep learning </s>

Bigrams:

1. **<s> → I**
2. **I → love**
3. **love → deep**
4. **deep → learning**
5. **learning → </s>**

Probabilities:

$\mathbf{P(I \mid <s>)} = \frac{2}{3}$

$P(love \mid I) = 1$

$P(deep \mid love) = \frac{1}{2}$

$P(learning \mid deep) = \frac{2}{2} = 1$

$P(</s> \mid learning) = \frac{1}{2}$

Multiply them:

$$\mathbf{P(S2)} = \frac{2}{3} \times 1 \times \frac{1}{2} \times 1 \times \frac{1}{2} = \frac{1}{6}$$

S1: $= \frac{1}{3}$

S2: $= \frac{1}{6}$

iii. S1 sentence is more probable sentence

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

     Answer:

     From the table:

     After **ate**, we have:

     . lunch: 6

     . dinner: 3

     . a: 2

     . the: 1

     Total count after **ate** = **12**

     MLE Formula:

$$P(noodle \mid ate) = \frac{c(ate, noodle)}{C(ate)}$$

     Result:

$$= \frac{0}{12} = 0$$

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

     Answer: The model entirely excludes word combinations that were not seen during training, even if they are valid.

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

     Answer:

     Add -1 smoothing formula

$$p(wi|wi-1) = \frac{C(wi-1,wi)+1}{C(wi-1)+V}$$

Given

C (ate, noodle) =0

C (ate) = 12

Vocabulary size V = 10

Compute:

$$\text{P(noodle | ate)} = \frac{0+1}{12+10} = \frac{1}{22}$$

Q4. 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).

   Answer:

   Using MLE trigram:

$$\text{P (cats |I, like)} = \frac{C\ (I,like,cats)}{C(I,like)} = \frac{1}{2}$$

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

Answer: trigram:

. trigram (you, like, dogs) does not appear training data.

So,C (you, like, dogs) = 0

Using Back off to bigram

$$P(\text{dogs} \mid \text{I, like}) = \frac{C(like,cats)}{C(like)}$$

Like cats = 2

Like dogs = 1

$$P(\text{dogs} \mid \text{I, like}) = \frac{1}{3}$$

3. Explain why backoff is necessary in this example.

   Answer:

   Backoff is necessary to prevent assigning zero probability to word sequences that were not seen in the training data but are still valid**.**

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).

     Answer: Per-class Metrics

     Formula: Precision for class X= $\frac{TP}{TP+FP}$

     Recall for class X= = $\frac{TP}{TP+FN}$

i.CAT

TP=5

FP=15

FN=15

Precision: $\frac{5}{5+15} = \frac{5}{20} = 0.25$

Recall: $\frac{5}{5+12} = \frac{5}{20} = 0.25$

ii. Dog

TP:20

FP:25

FN:25

Precision: = $\frac{20}{20+25} = \frac{20}{45} = 0.444$

Recall: = $\frac{20}{20+25} = \frac{20}{45} = 0.444$

iii. Rabbit

TP= 10

FP= 15

FN= 15

Precision: = $\frac{10}{10+15} = \frac{10}{25} = 0.4$

Recall: = $\frac{10}{10+15} = \frac{10}{25} = 0.4$

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

     Answer:

     Step 2a: Macro-Averaged Precision & Recall

     Macro-average = **average of per-class metrics**

$$\text{Macro Precision} = \frac{0.25 + 0.444 + 0.4}{3} = \frac{1.094}{3} \approx 0.365$$
$$\text{Macro Recall} = \frac{0.25 + 0.444 + 0.4}{3} = 0.365$$

   - Compute the micro-averaged precision and recall.

     Micro-average = **compute metrics using global TP, FP, FN across all classes**

.TP = 35

. **FP** = 55

. **FN** = 55

$$\text{Micro Precision} = \frac{\text{Total TP}}{\text{Total TP + Total FP}} = \frac{35}{35 + 55} = \frac{35}{90} \approx 0.389$$
$$\text{Micro Recall} = \frac{\text{Total TP}}{\text{Total TP + Total FN}} = \frac{35}{35 + 55} = \frac{35}{90} \approx 0.389$$

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

     Answer:

     . **Macro averaging:** Treats each class equally, **ignoring class size**. Useful to see performance **per class**, even if some classes are small.

     . **Micro averaging:** Treats **all instances equally**, so larger classes dominate the metric. Useful for **overall system performance**.

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.

Code:

```python
import numpy as np

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

classes = ['Cat', 'Dog', 'Rabbit']

# Initialize per-class precision and recall
precision = []
recall = []

# Compute per-class metrics
for i in range(len(classes)):
    tp = conf_matrix[i, i]
    fp = conf_matrix[i, :].sum() - tp
    fn = conf_matrix[:, i].sum() - tp
    prec = tp / (tp + fp) if (tp + fp) > 0 else 0
    rec = tp / (tp + fn) if (tp + fn) > 0 else 0
    precision.append(prec)
    recall.append(rec)
    print(f"{classes[i]} -> Precision: {prec:.3f}, Recall: {rec:.3f}")
```

```python
# Macro-averaged metrics
macro_precision = np.mean(precision)
macro_recall = np.mean(recall)
print(f"\nMacro-Averaged Precision: {macro_precision:.3f}")
print(f"Macro-Averaged Recall: {macro_recall:.3f}")

# Micro-averaged metrics
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(f"\nMicro-Averaged Precision: {micro_precision:.3f}")
print(f"Micro-Averaged Recall: {micro_recall:.3f}")
```

Output:

Cat -> Precision: 0.250, Recall: 0.250

Dog -> Precision: 0.444, Recall: 0.444

Rabbit -> Precision: 0.400, Recall: 0.400

Macro-Averaged Precision: 0.365

Macro-Averaged Recall: 0.365

Micro-Averaged Precision: 0.389

Micro-Averaged Recall: 0.389

Part II. Programming

Q1. 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.

Code:

```
from collections import defaultdict
import math

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

# 2. Tokenize corpus
tokenized_corpus = [sentence.split() for sentence in corpus]

# 3. Compute unigram counts
unigram_counts = defaultdict(int)
for sentence in tokenized_corpus:
    for word in sentence:
        unigram_counts[word] += 1
```

```
# 4. Compute bigram counts
bigram_counts = defaultdict(int)
for sentence in tokenized_corpus:
    for i in range(len(sentence)-1):
        bigram = (sentence[i], sentence[i+1])
        bigram_counts[bigram] += 1

# 5. Estimate bigram probabilities using MLE
def bigram_prob(w1, w2):
    return bigram_counts[(w1, w2)] / unigram_counts[w1]

# 6. Function to calculate probability of a sentence
def sentence_prob(sentence):
    words = sentence.split()
    prob = 1.0
    for i in range(len(words)-1):
        w1, w2 = words[i], words[i+1]
        bp = bigram_prob(w1, w2)
        prob *= bp
    return prob
```

```
# 7. Test sentences
sentences = [
    "<s> I love NLP </s>",
    "<s> I love deep learning </s>"
]

for sent in sentences:
    prob = sentence_prob(sent)
    print(f"Sentence: '{sent}' -> Probability: {prob:.6f}")

# 8. Determine which sentence the model prefers
probs = [sentence_prob(s) for s in sentences]
preferred_sentence = sentences[probs.index(max(probs))]
print(f"\nThe model prefers: '{preferred_sentence}' because it has higher probability.")
```

Output:

```
Sentence: '<s> I love NLP </s>' -> Probability: 0.333333
Sentence: '<s> I love deep learning </s>' -> Probability: 0.166667

The model prefers: '<s> I love NLP </s>' because it has higher probability.
```