**University of Central Missouri**

**Department of Computer Science & Cybersecurity**

**CS5760 Natural Language Processing**

**Fall 2025**

**Homework 2.**

**Student name: Indukuri Shanmukha Shiva**

**Kesava Varma**

**( 700766407 )**

**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:**

Bayes Rule Applied to Text

Task 1: Explain each term

* **P(c):** Prior probability of class c. This represents how likely class c is before seeing any document. It's calculated as the proportion of documents in the training set that belong to class c.
* **P(d|c)**: Likelihood of document d given class c. This represents how likely we are to see document d if we know it belongs to class c. In practice, this is computed as the product of probabilities of all words in the document given the class.
* **P(c|d)**: Posterior probability of class c given document d. This is what we want to find - the probability that document d belongs to class c after observing the document's content.

Task 2:

P(d) can be ignored when comparing classes because it's the same for all classes. Since we only care about which class has the highest probability, we can compare P(c|d) ∝ P(d|c) × P(c) for each class. The denominator P(d) acts as a normalization constant that doesn't affect the relative ordering of classes.

**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.

**Ans:**

**Given:**

* P(−) = 3/5, P(+) = 2/5
* Vocabulary size = 20
* Negative class total token count = 14

**Task 1 : Denominator for add-1 smoothing**

Denominator = Total token count + Vocabulary size = 14 + 20 = **34**

**Task 2: P(predictable|−) with "predictable" occurring 2 times**

P(predictable|−) = (count + 1) / (total + vocabulary) = (2 + 1) / (14 + 20) = 3/34 = **0.088**

**Task 3: P(fun|−) with "fun" never appearing**

P(fun|−) = (0 + 1) / (14 + 20) = 1/34 = **0.029**

**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?

**Ans:**

We need P(predictable|+) and P(no|+) from the positive class as well. Assuming we have these values, the calculation would be:

For Negative Class:

P(−|d) ∝ P(−) × P(predictable|−) × P(no|−) × P(fun|−) P(−|d) ∝ (3/5) × (3/34) × P(no|−) × (1/34)

For Positive class:

P(+|d) ∝ P(+) × P(predictable|+) × P(no|+) × P(fun|+) P(+|d) ∝ (2/5) × P(predictable|+) × P(no|+) × P(fun|+)

**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:**

**Task 1: Representational Harm**

**Representational harm** occurs when systems reinforce demeaning portrayals or negative stereotypes about social groups. The Kiritchenko & Mohammad (2018) study demonstrated this by showing that sentiment analysis systems consistently rated sentences about certain demographic groups (particularly women and minorities) more negatively, even when the content was neutral, perpetuating harmful stereotypes.

**Task 2: Risk of Censorship**

One major risk is **over-censorship of marginalized voices**. Toxicity classifiers may flag discussions of discrimination, hate speech experiences, or reclaimed language used by marginalized communities as toxic, effectively silencing legitimate discourse about important social issues.  
  
**Task 3: Performance Issues with Language Varieties**

Classifiers may perform worse on African American English or Indian English because:

1. **Training data bias**: Models are typically trained on Standard American English, lacking sufficient representation of other varieties
2. **Different linguistic features**: These varieties have distinct grammatical structures, vocabulary, and expressions that may be misinterpreted by models trained on standard varieties

**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.

# **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.

**Ans:**Task 1: Bigram Sentence Probabilities

**For S1: <s> I love NLP </s>**

* P(<s> I) = 2/3 = 0.667
* P(I love) = 2/2 = 1.0
* P(love NLP) = 1/2 = 0.5
* P(NLP </s>) = 1/1 = 1.0

P(S1) = 0.667 × 1.0 × 0.5 × 1.0 = **0.333**

**For S2: <s> I love deep learning </s>**

* P(<s> I) = 2/3 = 0.667
* P(I love) = 2/2 = 1.0
* P(love deep) = 1/2 = 0.5
* P(deep learning) = 2/2 = 1.0
* P(learning </s>) = 1/2 = 0.5

P(S2) = 0.667 × 1.0 × 0.5 × 1.0 × 0.5 = **0.167**

**S1 is more probable** under the bigram model.

### **Task 2: Zero-Probability Problem**

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

**Problem**: This creates a zero probability for any sentence containing this bigram, making the entire sentence probability zero regardless of other words.

**With Laplace smoothing:** P(noodle|ate) = (0 + 1)/(12 + 10) = 1/22 = **0.045**

### **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.

Ans:

Given counts:

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

**Task 1: P(cats|I,like)**

Since we have "I like" occurring 2 times, and we need to find how many times "I like cats" occurs. From the training data: "I like cats" appears in 2 out of 2 "I like" instances. P(cats|I,like) = 2/2 = **1.0**

**Task 2: P(dogs|You,like) using trigram → bigram backoff**

"You like dogs" appears 1 time out of 1 "You like" instance. However, if we use backoff to bigram: P(dogs|like) = 1/3 = **0.333**

**Task 3: Why backoff is necessary**

Backoff is necessary because we have sparse data. For trigrams, we may not have seen specific three-word sequences in our training data, so we fall back to bigrams (or unigrams) to get reasonable probability estimates rather than assigning zero probability.

### **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.