Laplace smoothing rationale

Excellent — you're asking for a **hands-on lab** that connects *N-gram Language Modeling* and *Naïve Bayes Text Classification* — two pillars of classical NLP.

Here's a complete **lab plan** you can use for a university-level NLP course or student assignment.
It's structured, practical, and includes theory, exercises, and sample code (Python).

🧪 NLP Lab: N-Gram Language Modeling and Naïve Bayes Text Classification

🎯 Learning Objectives

By the end of this lab, students will be able to:

1. Build **unigram, bigram, and trigram** language models from text.
2. Implement **additive smoothing** and compute **sentence probabilities**.
3. Compute **perplexity** as an intrinsic evaluation metric.
4. Use **Naïve Bayes** for **sentiment classification** on a small corpus.
5. Understand how *lexicon features* or *negation handling* modify counts and probabilities.

🧩 Part 1: Setup

Requirements:

```bash
pip install nltk pandas numpy scikit-learn matplotlib
```

Imports:

```python
import nltk
from nltk.util import ngrams
from nltk.corpus import reuters, movie_reviews
from collections import Counter, defaultdict
```

```
import math, random, numpy as np
from sklearn.model_selection import train_test_split
```

If you don't have NLTK corpora:

```
python

nltk.download('reuters')
nltk.download('movie_reviews')
nltk.download('punkt')
```

🧮 Part 2: N-Gram Language Model

Step 1: Load and tokenize text

```
python

from nltk.corpus import reuters

docs = reuters.sents(categories='acq')[:2000]
tokens = ['<s>'] + [w.lower() for sent in docs for w in sent] + ['</s>']
```

Step 2: Build n-grams and counts

```
python

def build_ngram_counts(tokens, n):
    return Counter(ngrams(tokens, n))

unigram_counts = build_ngram_counts(tokens, 1)
bigram_counts  = build_ngram_counts(tokens, 2)
trigram_counts = build_ngram_counts(tokens, 3)
```

Step 3: Calculate conditional probabilities (with add-1 smoothing)

```
python

V = len(set(tokens))
```

```
def bigram_prob(w1, w2):
    return (bigram_counts[(w1, w2)] + 1) / (unigram_counts[(w1,)] + V)
```

Step 4: Compute sentence probability and perplexity

```
python

def sentence_prob(sentence):
    sent = ['<s>'] + nltk.word_tokenize(sentence.lower()) + ['</s>']
    prob = 1
    for i in range(1, len(sent)):
        prob *= bigram_prob(sent[i-1], sent[i])
    return prob

def perplexity(test_sents):
    N = sum(len(s) for s in test_sents)
    log_prob = 0
    for s in test_sents:
        sent = ['<s>'] + s + ['</s>']
        for i in range(1, len(sent)):
            log_prob += math.log(bigram_prob(sent[i-1], sent[i]))
    return math.exp(-log_prob / N)
```

✅ **Exercise 1:**

- Compute sentence probabilities for:
  - `"the company made a profit"`
  - `"profit company the made"`
- Compare which is higher (more grammatical).
- Compute **perplexity** on 10 unseen sentences from Reuters.

🧠 Part 3: Naïve Bayes Text Classification

Step 1: Load sentiment dataset

```
python

from nltk.corpus import movie_reviews

docs = [(list(movie_reviews.words(fileid)), category)
        for category in movie_reviews.categories()
```

```
    for fileid in movie_reviews.fileids(category)]

train, test = train_test_split(docs, test_size=0.2, random_state=42)
```

Step 2: Build word counts per class

python

```
word_counts = {'pos': Counter(), 'neg': Counter()}
for words, label in train:
    word_counts[label].update(w.lower() for w in words)

V = len(set(w for w, _ in word_counts['pos'].items()) | set(word_counts['neg'].keys()))
total_counts = {c: sum(word_counts[c].values()) for c in ['pos','neg']}
```

Step 3: Define probability functions

python

```
def P_word_given_class(word, c):
    return (word_counts[c][word] + 1) / (total_counts[c] + V)

def P_class_given_doc(words):
    probs = {}
    for c in ['pos', 'neg']:
        log_prob = math.log(0.5)  # assume equal priors
        for w in words:
            log_prob += math.log(P_word_given_class(w, c))
        probs[c] = log_prob
    return max(probs, key=probs.get)
```

Step 4: Evaluate

python

```
correct = 0
for words, label in test[:200]:
```

```
    pred = P_class_given_doc(words)
    correct += (pred == label)

print("Accuracy:", correct / len(test[:200]))
```

✅ **Exercise 2:**

- Try removing stopwords and see how accuracy changes.
- Add negation handling: prepend `NOT_` after “not”, “never”, etc.
- Add lexicon features: append token `"LEX_POS"` if any word in a positive lexicon occurs.

🧩 Part 4: Lexicon Feature Experiment (optional)

```python
positive_lexicon = {"good", "excellent", "great", "amazing", "love", "nice"}
negative_lexicon = {"bad", "awful", "terrible", "hate", "boring", "poor"}

def add_lexicon_tokens(words):
    if any(w in positive_lexicon for w in words):
        words.append("LEX_POS")
    if any(w in negative_lexicon for w in words):
        words.append("LEX_NEG")
    return words
```

Then re-train your Naïve Bayes model using these augmented word lists.
Compare accuracies **with and without lexicon features**.

🧮 Part 5: Discussion & Analysis

Concept	Practice Link
Smoothing	Add-1 smoothing to avoid zero probs
Perplexity	Evaluates language model quality

Concept	Practice Link
Naïve Bayes independence assumption	Word counts independent given sentiment
Lexicon augmentation	Adds prior sentiment knowledge
Negation handling	Adds `NOT_` prefix to invert sentiment words

✅ Exercise 3:

- Explain how N-gram models capture syntax (local dependencies) while NB models capture semantics (word sentiment).
- Try combining N-gram features (bigrams) with Naïve Bayes and observe effects.

🧠 Optional Conceptual Questions

1. Why does smoothing improve perplexity on unseen text?
2. How does perplexity relate to branching probability of a language model?
3. Why might Naïve Bayes misclassify “not good”?
4. How can we use a held-out corpus to estimate interpolation weights between unigrams, bigrams, trigrams?

📘 Deliverables (for students)

1. Python notebook (`.ipynb`) with:
   - N-gram LM implementation
   - Sentence probability & perplexity results
   - Naïve Bayes classifier code
   - Results with & without lexicon / negation handling
2. A 1-page reflection:
   - Compare **statistical vs lexical** approaches to sentiment detection.

Would you like me to prepare this as a **ready-to-run Jupyter Notebook (.ipynb)** lab file (with code cells, theory markdown, and exercises)?
You could distribute it directly to students.