NLP Project 2
Tasks 1–4

February 26, 2026

What We Built

- A full pipeline across four NLP tasks:
  - Task 1: unigram/bigram/trigram language modeling (MLE)
  - Task 2: smoothing comparison for n-gram LMs
  - Task 3: sentiment classification (NB and Logistic Regression)
  - Task 4: dot-as-sentence-boundary binary classification

Data and Split Setup

Task	Data	Train / Dev / Test
1	`Corpora/News/corpus.txt`	96,000 / 12,000 / 12,000 (80/10/10)
2	`Corpora/News/corpus.txt`	96,000 / 12,000 / 12,000 (80/10/10)
3	`sentiment_dataset/dataset_v1.csv`	25,600 / 6,400 / 8,000 (70/15/15)
4	`dot_labeled_data.csv`	140,000 / 30,000 / 30,000 (70/15/15)

- Task 3 and Task 4 use stratified splits.

Task 1: N-gram LM with MLE

Modeling decisions

- Tokenized news sentences and trained unigram, bigram, trigram MLE models.
- Replaced rare training tokens (< 2 frequency) with <UNK>.
- Evaluated by perplexity on held-out test sentences.

Key result

Metric	Value
Unigram MLE PPL	3296.2567
Bigram MLE PPL	∞
Trigram MLE PPL	∞

Takeaway: plain MLE fails for higher-order n-grams because unseen test events get zero probability.

Task 2: Smoothing Comparison

What We compared

- Laplace, interpolation, backoff, and Kneser–Ney for bigram and trigram models.
- Interpolation weights tuned on dev set:
  - Bigram: $(\lambda_1, \lambda_2) = (0.2, 0.8)$
  - Trigram: $(\lambda_1, \lambda_2, \lambda_3) = (0.2, 0.3, 0.5)$

Method	Bigram PPL	Trigram PPL
Laplace	3572.4528	12556.2180
Interpolation	167.9878	88.0896
Backoff	158.2786	74.7584
Kneser–Ney	**150.0579**	**70.7616**

Task 2: Interpretation

- Kneser–Ney was best for both bigram and trigram perplexity.
- Trigram + smoothing outperformed bigram + smoothing, so extra context was useful after sparsity was fixed.
- Laplace gave the worst perplexity by a large margin in this setup.

Task 3: Sentiment Classification Setup

Models

- Multinomial Naive Bayes
- Bernoulli Naive Bayes
- Logistic Regression (`liblinear`)

Feature sets

- `bow`: CountVectorizer with unigram+bigram, `min_df=2`, `max_features=30000`
- `lexicon`: 6 handcrafted sentiment/negation features
- `bow_lexicon`: concatenation of both

Tuning

- NB: $\alpha \in \{0.05, 0.1, 0.3, 0.5, 1.0, 2.0\}$
- LR: $C \in \{0.1, 0.3, 0.5, 1, 2, 5, 10\}$, class weights $\{\texttt{none}, \texttt{balanced}\}$
- Selection metric: dev macro-F1

Task 3: Main Results

Model	Best features	Best setting	Dev macro-F1	Test macro-F1
MNB	`bow`	$\alpha = 0.05$	0.8491	0.8490
BNB	`bow`	$\alpha = 0.10$	0.8528	0.8547
LR	`bow_lexicon`	$C = 0.3$, balanced	**0.8974**	**0.9009**

- Best classifier: Logistic Regression with `bow_lexicon`.
- McNemar tests show LR is significantly stronger than both NB variants ($p \approx 0$).

Task 3: What Helped and What Did Not

- Lexicon-only features were weak (≈ 0.64 macro-F1), so they cannot replace text n-grams.
- But adding lexicon cues on top of BoW gave LR a small but real lift:

$$0.8992 \rightarrow 0.9009 \text{ (test macro-F1)}.$$

- This suggests handcrafted polarity and negation cues still add signal when combined with data-driven features.

Task 4: Dot Boundary Classifier Setup

- Binary classification on `dot_labeled_data.csv`.
- Inputs: all columns except `label`; vectorized with `DictVectorizer`.
- Model: Logistic Regression with both L2 and L1 regularization.
- Tuned $C \in \{0.01, 0.1, 1, 10, 100\}$ on dev F1.
- Tuned decision threshold on dev, scanning 0.10 to 0.89 (step 0.01).

Task 4: L1 vs L2 Results

Penalty	C	Thr.	Dev F1	Test Acc.	Prec.	Rec.	Test F1
L2	10	0.64	0.9993	0.9992	0.9980	0.9990	0.9985
L1	10	0.59	0.9993	**0.9993**	**0.9985**	0.9990	**0.9988**

Final selection: **L1** (highest test F1).

UI and Demo Layer

- Added two interfaces:
  - `results_ui.py`: Tkinter dashboard to run Task 1–4 and inspect metrics quickly.
  - `localhost_ui.py`: Streamlit local web app with results plus interactive demos.
- Streamlit side includes practical demos for:
  - LM sentence analysis with smoothing comparison,
  - custom-text sentiment prediction,
  - dot-boundary predictions on user text.

Final Takeaways

- MLE alone is not enough for higher-order n-gram LMs; smoothing is essential.
- Kneser–Ney was the most reliable smoothing method in this dataset.
- For sentiment, Logistic Regression + combined BoW/lexicon features performed best.
- For dot boundary detection, both L1 and L2 were very strong, with L1 slightly better.
- The UI tools made it easier to verify and compare models without changing core code.