Statistical NLP Methods on a Folklore Corpus

N-gram Language Modeling, Sentiment Classification, and Sentence Boundary Detection

Course: Natural Language Processing
Instructor: Dr.Samir Rustamov

Presented by:
Laman Guluzada
Abdullah Kazimov

Motivation

- Azerbaijani folklore text corpus
- N-gram Language Modeling + Smoothing
- Sentiment Classification (NB, Binary NB, Logistic
- Sentence boundary detection: L1/L2 regularization
- Insights from experiments
- Evaluation: accuracy, perplexity, stats tests

Dataset

- Corpus collected from folklore books
- Language: Azerbaijani
- Preprocessing:
  - Lowercasing
  - Tokenization with punctuation separation
  - Train–test split (80% / 20%)

N-gram Language Models

Constructed:

- Unigram
- Bigram
- Trigram language models

Probability estimation (MLE):

- $P(w) = \frac{c(w)}{N}$
- $P(w_i|w_{i-1}) = \frac{c(w_{i-1},w_i)}{c(w_{i-1})}$
- $P(w_i|w_{i-2}, w_{i-1}) = \frac{c(w_{i-2},w_{i-1},w_i)}{c(w_{i-2},w_{i-1})}$

- Evaluation metric: Perplexity

Perplexity

- Measures how well a language model predicts unseen text:

$$\mathrm{PP}(W) = \exp\left(-\frac{1}{N}\sum \log P(w_i|\text{history})\right)$$

- Lower PP indicates better language model
- Raw n-gram models suffer from zero probabilities

Smoothing Motivation

Problem:

- Many n-grams never appear in training data
- Leads to zero probabilities and very high perplexity

```
Unigram Perplexity: 3748.743988425603
Bigram Perplexity: 35134490.199565485
Trigram Perplexity: 66949748825.63585
```

Solution:

- Apply smoothing techniques to redistribute probability mass

We implemented:

- Laplace
- Interpolation
- Backoff
- Kneser–Ney

Applied Smoothing Functions

Laplace Smoothing

- Applied to:
  - Unigram
  - Bigram
  - Trigram
- Pros: simple, avoids zero probabilities
- Cons: over-smooths, hurts higher-order models

$$P(w_i|h) = \frac{c(h, w_i) + 1}{c(h) + |V|}$$

Interpolation & Backoff

- Uses all n-gram orders together.
- Backoff:
  - Use trigram if exists
  - Else back off to bigram or unigram with discount
- Better handles sparsity than Laplace.

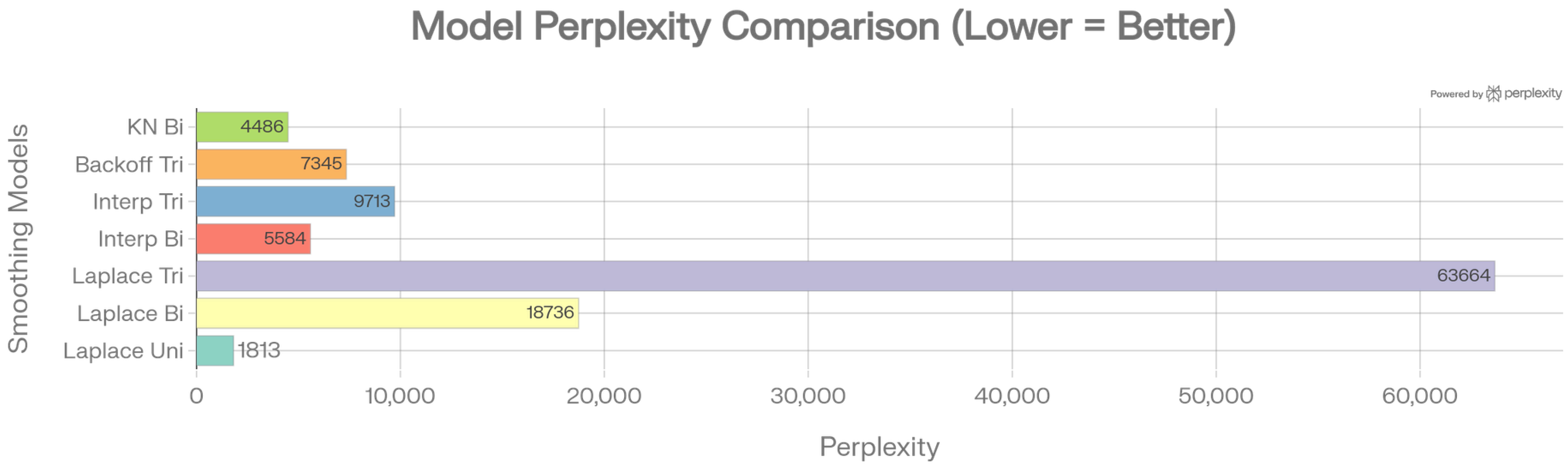
$$P = \lambda_3 P_{tri} + \lambda_2 P_{bi} + \lambda_1 P_{uni}$$

Kneser–Ney Smoothing

- Used for bigram model.
- Probability depends on how many different contexts a word appears in

$$P_{KN}(w|h) = \frac{\max(c(h, w) - D, 0)}{c(h)} + \lambda(h) P_{cont}(w)$$

Results



- Laplace Unigram has lowest PP but:
  - Does not model word dependencies
- Higher-order Laplace performs poorly due to over-smoothing
- Kneser–Ney Bigram achieves best balance:
  - Models context
  - Handles sparsity effectively

Best performing model: Kneser–Ney smoothed bigram

Sentiment Classification

Corpus:

- splitted into sentences
- Labeled automatically using sentiment lexicons
- Neutral sentences removed

Features:

- Bag-of-Words
- Binary Bag-of-Words
- Lexicon features (positive/negative counts)

Methodology

Feature Extraction

- Bag of Words
- Binary Bag of Words
- Sentiment Lexicon

Classifiers

- Naïve Bayes
- Binary Naïve Bayes
- Logistic Regression

Statistical Testing

McNemar’s test

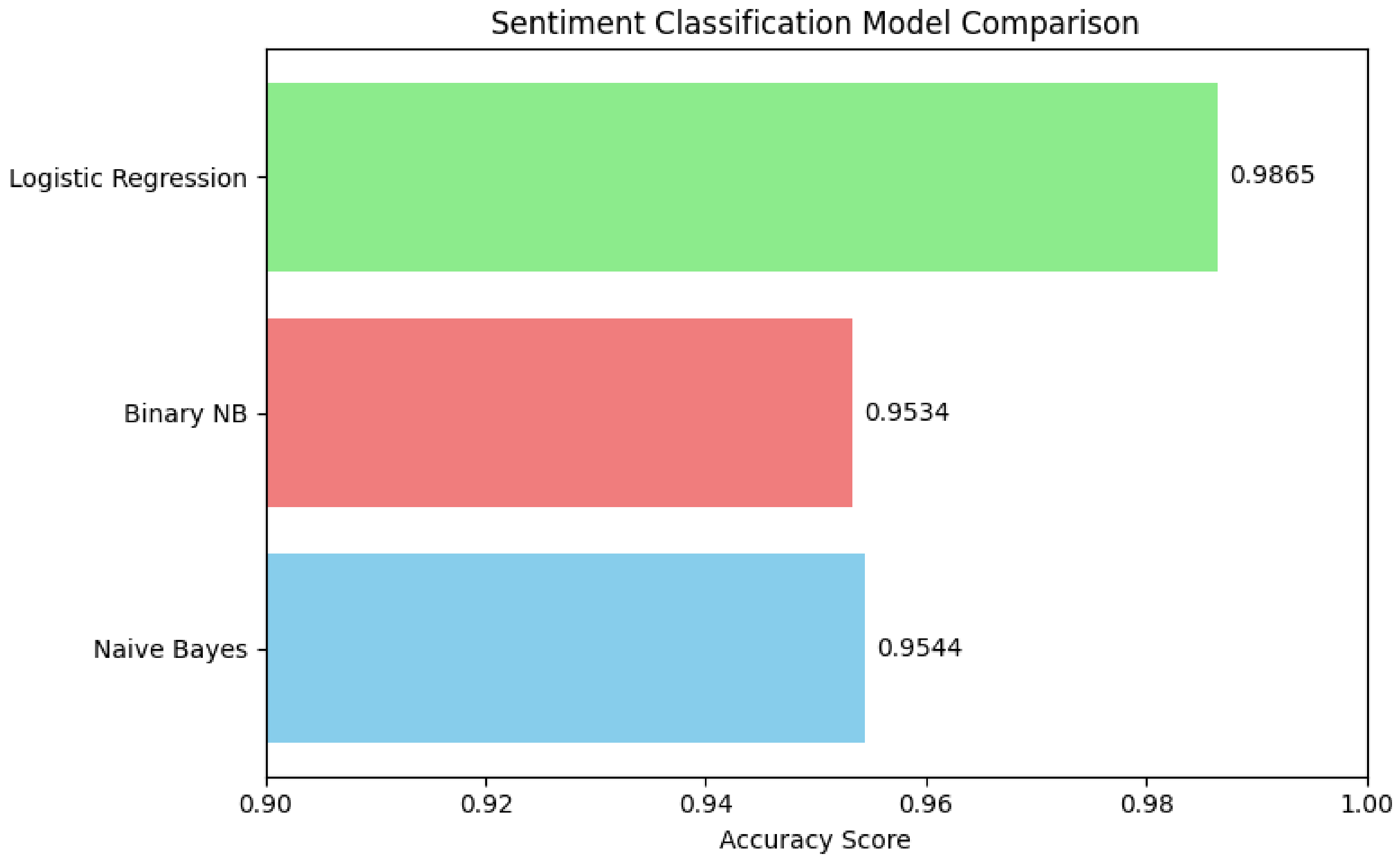
Classifiers & Results

Significance

We used McNemar's test:

- NB vs Logistic: $p<10^{(-5)}$
- Binary NB vs Logistic: $p<10^{(-6)}$

- Differences are statistically significant
- Logistic Regression performs better.



Sentence Boundary Detection

- Goal:
  - Decide whether a dot "." ends a sentence
- Binary classification.
- Features:
  - Is next character whitespace?
  - Is previous/next character uppercase?
  - Length of previous word

- Logistic Regression with:
  - L1 (Lasso)
  - L2 (Ridge)
- Results:
  - Accuracy: 100% for both
  - L1 selects only the most important feature
  - L2 spreads weights across features
- L1 is more interpretable.

```
L1 Coefficients: [[ 0.          0.          17.40122871  0.         ]]
L2 Coefficients: [[-0.07509966 -0.01545112 12.76347032  0.         ]]
```

THANK YOU