NLP501 - NATURAL LANGUAGE PROCESSING

Sentiment Analysis with Naïve Bayes

Session 02

Session Outline

PART 1

Probability Foundations

Conditional probability, Bayes' theorem

PART 2

Naïve Bayes Classifier

Independence, training, inference

PART 3

Laplacian Smoothing

Zero problem, add-α solution

PART 4

Log-Likelihood

Underflow, log transform

PART 5

Worked Example

Step-by-step classification

PART 6-7

Implementation & More

Algorithm, comparison, apps

Session 01 Review: Text Preprocessing

Preprocessing Pipeline

- Tokenization: split text into words
- Lowercasing: "Hello" → "hello"
- Stop word removal: remove "the", "is", "a"
- Stemming/Lemmatization
- Punctuation handling

Feature Extraction

- Bag of Words (BoW)
- Word frequencies
- Positive/negative word counts
- Feature vector: [pos_count, neg_count]
- Vocabulary building

Example: "I love this movie!" → tokens: ["i", "love", "this", "movie"] → features: [1, 0]

Session 01 Review: Logistic Regression

Model

$$\hat{y} = \sigma(w \cdot x + b)$$

Sigmoid function:

$$\sigma(z) = 1 / (1 + e^{-z})$$

Training

Cross-entropy loss:

$$L = -[y \log(\hat{y}) + (1-y) \log(1-\hat{y})]$$

Gradient descent update:

$$w := w - \alpha \cdot \nabla L$$

Key insight: Logistic Regression is a discriminative model - directly models P(class|features)

Classification Approaches

Discriminative

Models P(class | features) directly

$$P(y \mid x)$$

Examples: Logistic Regression, SVM, Neural Networks

Generative

Models P(features | class) then uses Bayes

$$P(x \mid y) \times P(y)$$

Examples: Naïve Bayes, HMM, Gaussian Mixture

Today: Naïve Bayes - a generative approach to text classification

Why Generative Models?

Advantages

- Works well with small datasets
- Fast training (just counting)
- Easy to interpret
- Handles missing features naturally
- Can generate new samples

Trade-offs

- Strong independence assumption
- May not capture feature interactions
- Probability estimates not calibrated
- Sensitive to feature distribution

Despite simplifying assumptions, Naïve Bayes often performs surprisingly well in practice!

PART 1

Probability Foundations

P(A|B)

Sample Space and Events

Definitions

Sample Space (Ω): Set of all possible outcomes

Event (A): Subset of the sample space

P(A): Probability that event A occurs

NLP Example

Sentiment classification:

Ω = {positive, negative}

A = "tweet is positive"

P(positive) = 0.6, P(negative) = 0.4

$$P(\Omega) = 1 \text{ and } 0 \leq P(A) \leq 1$$

Probability Fundamentals

Joint Probability

$$P(A \cap B)$$

Both A and B occur

Marginal Probability

$$P(A) = \Sigma P(A \cap B_i)$$

Sum over all B

Conditional Probability

$$P(A|B)$$

A given B occurred

$$P(A|B) = P(A \cap B) / P(B)$$

Conditional Probability Example

Sentiment Dataset

Total tweets: 10,000

- Positive tweets: 6,000
- Negative tweets: 4,000
- Tweets with "happy": 3,500
- Positive + "happy": 3,000

Question

If tweet contains "happy", what is probability it's positive?

$P(\text{pos} \mid \text{"happy"}) = ?$

$$P(\text{pos} \mid \text{"happy"}) = P(\text{pos} \cap \text{"happy"}) / P(\text{"happy"}) = 3000/3500 = 0.857$$

Deriving Bayes' Theorem

Step 1: Start with conditional probability definition

$$P(A|B) = P(A \cap B) / P(B)$$

Step 2: Similarly, we can write

$$P(B|A) = P(A \cap B) / P(A)$$

Step 3: Rearrange to get P(A ∩ B)

$$P(A \cap B) = P(B|A) \times P(A)$$

Step 4: Substitute into Step 1

$$P(A|B) = P(B|A) \times P(A) / P(B)$$

Bayes' Theorem

$$P(A|B) = P(B|A) \times P(A) / P(B)$$

$P(A\|B)$	$P(B\|A)$	$P(A)$	$P(B)$
Posterior	**Likelihood**	**Prior**	**Evidence**
What we want	Evidence given class	Background belief	Normalizing constant

Bayes' Theorem for Classification

For text classification (class c given text x):

$$P(c|x) = P(x|c) \times P(c) / P(x)$$

P(c|x) - Posterior

Probability of class c given we observed text x

P(x|c) - Likelihood

Probability of seeing text x in class c documents

P(c) - Prior

Background probability of class c (from training data)

P(x) - Evidence

Same for all classes - can ignore for comparison!

Classification Decision

Since $P(x)$ is the same for all classes, we classify based on:

$$\hat{c} = \text{argmax } P(c|x) = \text{argmax } P(x|c) \times P(c)$$

Binary Classification

Compare two classes:

$P(\text{pos}|x) \times P(\text{pos})$ vs $P(\text{neg}|x) \times P(\text{neg})$

Choose the class with higher value

Multi-class Classification

Compare all K classes:

$\hat{c} = \text{argmax}_{c \in C} P(x|c) \times P(c)$

Choose the class with highest score

PART 2

Naïve Bayes Classifier

$$\prod P(w_i|c)$$

The Text Classification Problem

Problem Statement

Given a document d represented as words:

$$d = (w_1, w_2, \ldots, w_n)$$

Find the most likely class $c \in C$

Challenge

Computing $P(w_1, w_2, \ldots, w_n \mid c)$ directly:

- Requires exponential parameters
- Most word combinations never seen
- Impossible to estimate from data

Solution: Make a simplifying assumption about word independence!

The Naïve Independence Assumption

Assume words are conditionally independent given the class:

$$P(w_1, w_2, \ldots, w_n \mid c) = \prod_i P(w_i \mid c)$$

Without Assumption

$P(w_1, w_2, \ldots, w_n \mid c)$

Parameters: $|V|^n$ for each class

Impossible to estimate!

With Assumption

$\prod_i P(w_i \mid c)$

Parameters: $|V| \times |C|$

Easy to estimate!

Why "Naïve"?

The Assumption is Wrong!

Words are NOT independent in natural language:

- "New York" - words co-occur
- "not good" - negation depends on context
- "machine learning" - compound phrase
- Grammar enforces dependencies

But It Works Well!

Despite the wrong assumption:

- Classification accuracy still high
- Ranking of classes preserved
- Dependencies often cancel out
- Simplicity enables better estimation

"Naïve" refers to the simplifying assumption, not the performance!

Complete Naïve Bayes Formula

Classification rule:

$$\hat{c} = \text{argmax}\ P(c) \times \prod_i P(w_i|c)$$

P(c) - Prior

$P(c) = N_c / N$

Documents of class c / Total documents

P(w|c) - Likelihood

$P(w|c) = \text{count}(w,c) / \Sigma\, \text{count}(w',c)$

Word count in class / Total words in class

∏ - Product

$\prod_i P(w_i|c)$

Multiply all word probabilities

Training: Computing Prior P(c)

Formula

$$P(c) = N^c / N$$

N^c = Number of documents in class c

N = Total number of documents

Example

Training data: 10,000 tweets

- Positive: 6,000 tweets
- Negative: 4,000 tweets

P(pos) = 6000/10000 = 0.6

P(neg) = 4000/10000 = 0.4

Training: Computing Likelihood P(w|c)

Formula

$$P(w|c) = \text{freq}(w, c) / \Sigma \text{freq}(w', c)$$

freq(w, c) = Count of word w in class c

Σ freq(w', c) = Total words in class c

Example

Word "happy" in positive class:

- freq("happy", pos) = 3,000
- Total words in pos = 60,000

P("happy"|pos) = 3000/60000 = 0.05

Maximum Likelihood Estimation (MLE): Simply count and divide!

Building the Frequency Table

Count occurrences of each word in each class:

Word Frequency Table

Word	Positive	Negative
happy	3,000	500
sad	200	2,500
love	4,000	800
hate	300	2,800
Total	**60,000**	**40,000**

Probability Table P(w|c)

Word	P(w|pos)	P(w|neg)
happy	0.050	0.0125
sad	0.0033	0.0625
love	0.0667	0.020
hate	0.005	0.070

Prediction Example Setup

New tweet to classify:

"I am happy and love this"

After Preprocessing

Tokens: ["happy", "love"]

(Removed stop words: "I", "am", "and", "this")

Known Parameters

P(pos) = 0.6, P(neg) = 0.4

P("happy"|pos) = 0.050

P("happy"|neg) = 0.0125

P("love"|pos) = 0.0667

P("love"|neg) = 0.020

Goal: Compare P(pos) × P("happy"|pos) × P("love"|pos) vs P(neg) × P("happy"|neg) × P("love"|neg)

Prediction Calculation

Score for Positive

P(pos) × P("happy"|pos) × P("love"|pos)

= 0.6 × 0.050 × 0.0667

= 0.6 × 0.003335

= 0.002001

Score for Negative

P(neg) × P("happy"|neg) × P("love"|neg)

= 0.4 × 0.0125 × 0.020

= 0.4 × 0.00025

= 0.0001

0.002001 > 0.0001 → Prediction: POSITIVE ✓

Likelihood Ratio Interpretation

Ratio of scores (ignoring priors for now):

$$P(\text{"happy"}|pos)/P(\text{"happy"}|neg) \times P(\text{"love"}|pos)/P(\text{"love"}|neg)$$

"happy" ratio

0.050 / 0.0125

= 4.0

4× more likely in positive

"love" ratio

0.0667 / 0.020

= 3.3

3.3× more likely in positive

Combined ratio

4.0 × 3.3

= 13.2

13× more likely positive!

PART 3

Laplacian Smoothing

+ α

The Zero Probability Problem

Critical Issue: If a word never appears in a class, P(word|class) = 0

Scenario

Training data never contains:

- "extraordinary" in negative tweets
- freq("extraordinary", neg) = 0

$P(\text{"extraordinary"}|\text{neg}) = 0/40000 = 0$

Catastrophic Consequence

Entire product becomes zero:

$P(\text{neg}|\text{text}) \propto P(\text{neg}) \times \ldots \times 0 \times \ldots$

$$= 0$$

One unseen word destroys entire calculation!

Zero Problem: Detailed Example

Tweet: "This movie was extraordinary and amazing"

Positive Score

P(pos) × P("extraordinary"|pos) × P("amazing"|pos)

= 0.6 × 0.001 × 0.003

= 0.0000018

Negative Score

P(neg) × P("extraordinary"|neg) × P("amazing"|neg)

= 0.4 × 0 × 0.001

= 0

Result: Always predicts positive whenever "extraordinary" appears! Model is broken.

Laplacian (Add-α) Smoothing

Smoothed probability estimate:

$$P(w|c) = (freq(w,c) + \alpha) / (N^c + \alpha \times |V|)$$

α	$freq(w,c)$	N^c	$\|V\|$
Smoothing parameter	Word count in class	Total words in class c	Vocabulary size
Usually α = 1	Can be 0	Denominator base	Unique words

Key insight: Even unseen words get a small positive probability!

Smoothing Example Calculation

Given: $\alpha = 1$, $|V| = 10{,}000$ unique words, $N^{neg} = 40{,}000$

Word "extraordinary": freq("extraordinary", neg) = 0

Without Smoothing (MLE)

$$P(\text{"extraordinary"}|neg) = 0 / 40{,}000$$

$$= 0$$

With Smoothing ($\alpha = 1$)

$$P(\text{"extraordinary"}|neg) = (0 + 1) / (40{,}000 + 10{,}000)$$

$$= 1 / 50{,}000$$

$$= 0.00002$$

Small but non-zero! The product won't collapse to zero anymore.

Smoothing Effect Comparison

Word	freq(w,neg)	MLE P(w|neg)	Smoothed P(w|neg)
sad	2,500	0.0625	0.05002
hate	2,800	0.070	0.05602
happy	500	0.0125	0.01002
extraordinary	**0**	**0 ✗**	**0.00002 ✓**

Note: Smoothing slightly reduces high-frequency word probabilities to give mass to unseen words.

PART 4

Log-Likelihood

$\log \Sigma$

Numerical Underflow Problem

Multiplying many small probabilities leads to numerical underflow!

The Calculation

For a document with 50 words:

$$P(c|doc) \propto P(c) \times P(w_1|c) \times \ldots \times P(w_{50}|c)$$

If each $P(w_i|c) \approx 0.01$:

$$\approx 0.6 \times (0.01)^{50}$$

$$= 6 \times 10^{-101}$$

Computer Limitation

64-bit floating point (double):

- Smallest positive: ~10^{-308}
- But precision lost around 10^{-15}
- Very small numbers → 0.0
- Cannot compare two tiny numbers!

Logarithm Properties

Key Properties

$$\log(a \times b) = \log(a) + \log(b)$$

$$\log(a^n) = n \times \log(a)$$

log is monotonically increasing

Why This Helps

- Products become sums
- Small numbers become negative (manageable)
- log(0.01) = -2 (not 0.01)
- $\log(10^{-100}) = -100$ (not underflow!)
- Comparison order preserved

If a > b, then log(a) > log(b) — ranking preserved!

Log Transformation Solution

Original classification rule:

$$\hat{c} = \text{argmax}\ P(c) \times \prod_i P(w_i|c)$$

Apply log to both sides:

$$\hat{c} = \text{argmax}\ [\ \log P(c) + \Sigma_i \log P(w_i|c)\ ]$$

Products → Sums

No multiplication of tiny numbers

Numerically Stable

Sums of negative numbers

Same Result

Ranking order preserved

Log-Likelihood Example

Tweet: "I am happy and love this" → ["happy", "love"]

Log Score for Positive

log P(pos) + log P("happy"|pos) + log P("love"|pos)

= log(0.6) + log(0.05) + log(0.0667)

= -0.51 + (-3.00) + (-2.71)

= -6.22

Log Score for Negative

log P(neg) + log P("happy"|neg) + log P("love"|neg)

= log(0.4) + log(0.0125) + log(0.02)

= -0.92 + (-4.38) + (-3.91)

= -9.21

-6.22 > -9.21 → Prediction: POSITIVE ✓

Log-Likelihood Ratio (Lambda)

For binary classification, compute the ratio:

$$\lambda(w) = \log P(w|pos) / P(w|neg) = \log P(w|pos) - \log P(w|neg)$$

$\lambda(w) > 0$

Word indicates positive

e.g., "love", "happy", "great"

$\lambda(w) < 0$

Word indicates negative

e.g., "hate", "sad", "terrible"

$\lambda(w) \approx 0$

Word is neutral

e.g., "the", "is", "movie"

$$\text{Total score} = \log P(pos)/P(neg) + \Sigma_i \lambda(w_i)$$

PART 5

Example

Step-by-Step

Example: Training Data

Positive Documents (3)

1. "I love this movie"
2. "Great film really good"
3. "Love the great acting"

Words: love(2), movie(1), great(2), film(1), really(1), good(1), acting(1)

Negative Documents (2)

1. "Terrible movie really bad"
2. "Bad film hate it"

Words: terrible(1), movie(1), really(1), bad(2), film(1), hate(1), it(1)

Vocabulary $|V| = 11$ unique words. Total: 5 documents, 3 positive, 2 negative.

Example: Computing Probabilities

Prior Probabilities

$P(pos) = 3/5 = 0.6$

$P(neg) = 2/5 = 0.4$

Likelihoods with Smoothing (α=1)

N^{pos} = 9 words, N^{neg} = 8 words, $|V|$ = 11

P("love"|pos) = (2+1)/(9+11) = 3/20 = 0.15

P("love"|neg) = (0+1)/(8+11) = 1/19 = 0.053

P("bad"|pos) = (0+1)/(9+11) = 1/20 = 0.05

P("bad"|neg) = (2+1)/(8+11) = 3/19 = 0.158

Note: "love" never in negative, "bad" never in positive - smoothing gives them small non-zero probabilities!

Example: Classifying New Document

New document to classify:

"love this film" → tokens: ["love", "film"]

Positive Log Score

log P(pos) + log P("love"|pos) + log P("film"|pos)

= log(0.6) + log(0.15) + log(0.1)

= -0.51 + (-1.90) + (-2.30)

= -4.71

Negative Log Score

log P(neg) + log P("love"|neg) + log P("film"|neg)

= log(0.4) + log(0.053) + log(0.105)

= -0.92 + (-2.94) + (-2.25)

= -6.11

-4.71 > -6.11 → Prediction: POSITIVE ✓

PART 6

Implementation

{ code }

Training Algorithm

Algorithm: Train Naïve Bayes

Input: Documents D, Labels Y, Smoothing α

Output: log_prior, log_likelihood

- Build vocabulary V from all documents
- For each class c ∈ {pos, neg}:
- Count documents: $N^c = |\{d : label(d) = c\}|$
  - Count documents: $N^c = |\{d : label(d) = c\}|$
- Compute log_prior[c] = log(N^c / N)
  - Compute log_prior[c] = log(N^c / N)
- For each class c and word w ∈ V:
- Count: freq(w, c) = occurrences of w in class c
  - Count: freq(w, c) = occurrences of w in class c
- Total: N^c_words = total words in class c
  - Total: N^c_words = total words in class c
- Compute log_likelihood[w, c] = log((freq(w,c) + α) / (N^c_words + α|V|))
  - Compute log_likelihood[w, c] = log((freq(w,c) + α) / (N^c_words + α|V|))
- Return log_prior, log_likelihood

Prediction Algorithm

Algorithm: Predict with Naïve Bayes

Input: Document d, log_prior, log_likelihood, V

Output: Predicted class ĉ

- Preprocess document d → tokens [w_1, w_2, …, w_n]
- For each class c ∈ {pos, neg}:
- Initialize: score[c] = log_prior[c]
- For each token w_i in d:
- If w_i ∈ V: score[c] += log_likelihood[w_i, c]
    - If w_j ∈ V: score[c] += log_likelihood[w_i, c]
- Else: skip (unknown word)
    - Else: skip (unknown word)
- Return ĉ = argmax score[c]

Time complexity: O(|d| × |C|) - Very fast!

Python Implementation Sketch

```
# Training
def train(docs, labels, alpha=1):
    vocab = build_vocabulary(docs)
    log_prior = {c: log(count(c)/len(docs)) for c in classes}
    word_counts = count_words_per_class(docs, labels)
    log_likelihood = {}
    for c in classes:
        total = sum(word_counts[c].values())
        for w in vocab:
            log_likelihood[(w,c)] = log((word_counts[c].get(w,0)+alpha)/(total+alpha*len(vocab)))
    return log_prior, log_likelihood, vocab

# Prediction
def predict(doc, log_prior, log_likelihood, vocab):
    tokens = preprocess(doc)
    scores = {c: log_prior[c] for c in classes}
    for w in tokens:
        if w in vocab:
            for c in classes: scores[c] += log_likelihood[(w,c)]
    return max(scores, key=scores.get)
```

Evaluation Metrics

Confusion Matrix

	Pred +	Pred -
Actual +	TP	FN
Actual -	FP	TN

Key Metrics

$$\text{Accuracy} = (TP + TN) / (TP + TN + FP + FN)$$

$$\text{Precision} = TP / (TP + FP)$$

$$\text{Recall} = TP / (TP + FN)$$

$$F1 = 2 \times (\text{Precision} \times \text{Recall}) / (\text{Precision} + \text{Recall})$$

For imbalanced datasets, F1-score is more informative than accuracy alone.

PART 7

Comparison & Variants

NB vs LR

Naïve Bayes vs Logistic Regression

Naïve Bayes (Generative)

- Models P(features|class)
- Uses Bayes' theorem
- Strong independence assumption
- Training: just counting
- Very fast training
- Works well with small data
- Handles missing features

Logistic Regression (Discriminative)

- Models P(class|features) directly
- Uses sigmoid function
- No independence assumption
- Training: gradient descent
- Slower training (iterative)
- Needs more data
- Can learn feature interactions

Both often achieve similar accuracy on text classification, but NB is faster to train!

Detailed Comparison Table

Aspect	Naïve Bayes	Logistic Regression
Training Time	O(N × |V|) - Fast	O(iterations × N × |V|)
Prediction Time	O(|doc|)	O(|V|)
Small Dataset	Better ✓	May overfit
Large Dataset	Good	Better ✓
Probability Calibration	Poor	Good ✓
Feature Correlation	Ignores	Handles ✓
Interpretability	High ✓	High ✓

When to Use Each Model

Choose Naïve Bayes When:

- Limited training data available
- Need fast training time
- Features are relatively independent
- Quick baseline needed
- Real-time training required
- Memory is constrained

Choose Logistic Regression When:

- Large training dataset available
- Need calibrated probabilities
- Features are correlated
- Accuracy is top priority
- Regularization is needed
- Feature importance analysis needed

Tip: Start with Naïve Bayes as baseline, then try Logistic Regression!

Naïve Bayes Variants

Multinomial NB

Features: Word counts/frequencies

Best for: Text classification, document categorization

What we learned today!

Bernoulli NB

Features: Binary (word present/absent)

Best for: Short texts, binary features

P(w|c) if present, 1-P(w|c) if absent

Gaussian NB

Features: Continuous values

Best for: Numerical data, real-valued features

Assumes Gaussian distribution

All variants share the same independence assumption but differ in feature distribution modeling.

Multinomial vs Bernoulli NB

Multinomial NB

Document: "happy happy love"

$$P(doc|c) = P(\text{"happy"}|c)^2 \times P(\text{"love"}|c)$$

- Counts word frequency
- "happy" appears twice → counted twice
- Good for long documents

Bernoulli NB

Document: "happy happy love"

$$P(doc|c) = P(\text{"happy"}|c) \times P(\text{"love"}|c) \times (1\text{-}P(\text{"sad"}|c))$$

- Only presence/absence matters
- Penalizes absent words
- Good for short texts (tweets)

For text classification, Multinomial NB usually performs better on longer documents.

PART 8

Applications

Application: Spam Detection

How It Works

- Classes: spam, not-spam (ham)
- Features: words in email
- High P("free"|spam), P("winner"|spam)
- Low P("meeting"|spam)
- Very fast classification

Example

Email: "Congratulations! You won $1000 free prize!"

- "congratulations" → common in spam
- "won" → common in spam
- "free" → very common in spam
- "prize" → common in spam

→ Classified as SPAM

Gmail's original spam filter used Naïve Bayes! Still used as baseline in many systems.

Application: Document Classification

News Categorization

Classify articles into topics:

- Sports: "goal", "team", "match", "player"
- Politics: "election", "vote", "president"
- Technology: "AI", "software", "startup"
- Business: "stock", "market", "revenue"

Multi-class Extension

Same algorithm, multiple classes:

$$\hat{c} = \text{argmax}\ P(c) \times \prod_i P(w_i|c)$$

Just compute score for each class and pick the highest. No changes to algorithm!

Real applications: Google News, Yahoo News, RSS feed categorization, library systems

Application: Sentiment Analysis

Use Cases

- Product review analysis
- Social media monitoring
- Brand reputation tracking
- Customer feedback analysis
- Stock market sentiment
- Political opinion mining

Challenges

- Sarcasm: "Great, another bug!"
- Negation: "not good" ≠ good
- Context: "sick beat" (positive)
- Domain-specific: "unpredictable plot"
- Emojis and slang

NB achieves ~80-85% accuracy on standard benchmarks. Good baseline before trying deep learning!

More Applications

Language Detection

Identify language from text using character n-grams

Author Attribution

Identify writing style to determine authorship

Topic Labeling

Auto-tag content for search and organization

Medical Diagnosis

Classify symptoms to suggest diagnoses

Intent Classification

Route customer queries to right department

Content Moderation

Flag inappropriate or harmful content

SUMMARY

Takeaways

Summary: Core Concepts

1. Bayes' Theorem: $P(c|x) = P(x|c) \times P(c) / P(x)$ — Inverts conditional probability

2. Naïve Assumption: $P(w_1, w_2, \ldots|c) = \prod P(w_i|c)$ — Words are conditionally independent

3. Laplacian Smoothing: $P(w|c) = (\text{freq} + \alpha) / (N + \alpha|V|)$ — Prevents zero probabilities

4. Log-Likelihood: $\log P(c) + \Sigma \log P(w_i|c)$ — Products become sums, avoids underflow

5. Classification: $\hat{c} = \text{argmax} [\log P(c) + \Sigma \log P(w_i|c)]$ — Choose class with highest score

Key Formulas Reference

Bayes' Theorem

$$P(c|d) = P(d|c) \times P(c) / P(d)$$

Prior Probability

$$P(c) = N^c / N$$

Likelihood (Smoothed)

$$P(w|c) = (\text{freq}(w,c) + \alpha) / (N^c + \alpha|V|)$$

Classification Rule

$$\hat{c} = \text{argmax}\,[\log P(c) + \Sigma_i \log P(w_i|c)]$$

Practical Insights

Strengths

- Very fast training & prediction
- Works with small datasets
- Easy to implement
- Interpretable results
- Handles high dimensions well
- No hyperparameter tuning needed

Limitations

- Independence assumption often wrong
- Probability estimates not calibrated
- Sensitive to feature selection
- Cannot learn feature interactions
- Zero-frequency problem

Best Practices

- Always use smoothing (α=1)
- Use log probabilities
- Good text preprocessing
- Remove stop words
- Consider n-grams
- Cross-validate α value

Session 01 vs Session 02 Summary

Session 01: Logistic Regression

- Discriminative model
- Models P(y|x) directly
- Uses gradient descent
- Sigmoid activation
- Feature vector [pos_count, neg_count]
- Iterative training

Session 02: Naïve Bayes

- Generative model
- Models P(x|y) then uses Bayes
- Uses counting + Bayes theorem
- Log-likelihood scores
- Full word probabilities
- Single-pass training

Both achieve similar accuracy (~80-85%) on sentiment analysis. NB faster, LR more flexible.

Lab Assignment: Implement Naïve Bayes

Tasks

- Build vocabulary from training data
- Compute prior probabilities P(c)
- Compute likelihoods P(w|c) with smoothing
- Implement log-likelihood prediction
- Evaluate on test set
- Compare with Logistic Regression

Dataset

Twitter Sentiment Dataset:

- 5,000 positive tweets
- 5,000 negative tweets
- 80% train / 20% test split

Expected accuracy: 75-80%

Bonus: Try different smoothing values (α = 0.1, 0.5, 1.0, 2.0) and compare results!