```
-->
```

Chapter 3: Classification and Language Modeling

Covered Material	Value
Chapter	Chapters 3, 4, 5 (Jurafsky & Martin)
Book	Speech and Language Processing (3rd ed.)
Related Week	Week 3
Related Slides / Files	week3.md, week3_inperson_slides.md

Index

Brief Content Summary

This chapter provides a comprehensive treatment of foundational methods in statistical Natural Language Processing (NLP), encompassing both language modeling and text classification. The chapter begins with n-gram language models, introducing the chain rule of probability and the Markov assumption that enables tractable estimation of word sequence probabilities. Maximum Likelihood Estimation (MLE) is formalized, alongside smoothing techniques—

including Laplace smoothing and interpolation—that address the zero-frequency problem. Perplexity, defined as the geometric mean of inverse probabilities, serves as the primary intrinsic evaluation metric.

The chapter then examines classification algorithms, distinguishing generative models (Naive Bayes) from discriminative models (logistic regression). Naive Bayes classifiers exploit the bag-of-words assumption and conditional independence to enable efficient probabilistic classification. Logistic regression introduces the sigmoid and softmax functions, with optimization achieved via stochastic gradient descent on the cross-entropy loss function. Regularization techniques (L1 and L2) mitigate overfitting.

Additional classification paradigms—Decision Trees (using entropy-based information gain), Support Vector Machines (maximum margin and kernel methods), and k-Nearest Neighbors (instance-based learning)—complete the survey. The chapter concludes with evaluation methodology, including precision-recall metrics, cross-validation, and statistical significance testing.

Abstract

This chapter presents a unified treatment of probabilistic language modeling and supervised text classification, two foundational pillars of statistical Natural Language Processing. We begin with n-gram language models, formalizing the estimation of word sequence probabilities under the Markov assumption. Maximum Likelihood Estimation provides the baseline approach, while smoothing algorithms—Laplace smoothing, add-k smoothing, and linear interpolation—address the zero-frequency problem inherent in finite training corpora. Perplexity is established as the standard intrinsic evaluation metric, with its information-theoretic foundations in cross-entropy explained.

The chapter then develops the theory of probabilistic classification, contrasting generative models (Naive Bayes) with discriminative models (logistic regression). For Naive Bayes, we derive the classification rule from Bayes' theorem under the conditional independence assumption. For logistic regression, we introduce the sigmoid and softmax activation functions, derive the cross-entropy loss, and present the stochastic gradient descent optimization algorithm with regularization. Additional classification algorithms—Decision Trees with information gain, Support Vector Machines with kernel methods, and k-Nearest Neighbors—are formalized.

The chapter concludes with model evaluation methodology, including confusion matrices, precision-recall-F1 metrics, cross-validation procedures, and the paired bootstrap test for statistical significance. Together, these methods constitute the essential toolkit for building and evaluating NLP classification systems.

Keywords

N-gram language models; Markov assumption; Maximum Likelihood Estimation; perplexity; smoothing; Laplace smoothing; interpolation; Naive Bayes; Bayes' theorem; conditional independence; bag-of-words; sentiment analysis; logistic regression; sigmoid function; softmax function; cross-entropy loss; gradient descent; regularization; Decision Trees; information gain; entropy; Support Vector Machines; kernel trick; k-Nearest Neighbors; precision; recall; F-measure; cross-validation; overfitting

1. Introduction

This chapter establishes the mathematical foundations of statistical language modeling and text classification. These two interconnected domains underpin a vast array of Natural Language Processing applications, from spelling correction and machine translation to sentiment analysis and spam detection.

Language models assign probabilities to sequences of words, enabling systems to distinguish likely utterances from improbable ones. The n-gram model, despite its simplicity, introduces fundamental concepts—the Markov assumption, maximum likelihood estimation, and smoothing—that recur throughout modern NLP. Classification, meanwhile, addresses the task of assigning discrete labels to input observations. We develop both generative approaches (which model the joint distribution of features and labels) and discriminative approaches (which directly model the conditional distribution of labels given features).

The chapter synthesizes material from Chapters 3, 4, and 5 of Jurafsky and Martin's *Speech and Language Processing*, supplemented by lecture materials covering Decision Trees, Support Vector Machines, and k-Nearest Neighbors. The emphasis throughout is on mathematical rigor: derivations are provided for key results, and worked examples illustrate computational procedures.

2. Background

Key References

- **Jurafsky, D. & Martin, J.H.** (2024). *Speech and Language Processing* (3rd ed., draft). Chapters 3–5.
- **Shannon, C.E.** (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27(3), 379–423.
- **Markov, A.A.** (1913). Statistical investigation of the text of "Eugene Onegin."
- **Mosteller, F. & Wallace, D.L.** (1964). *Inference and Disputed Authorship: The Federalist*.
- **Berger, A., Della Pietra, S., & Della Pietra, V.** (1996). A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1), 39–71.

Historical Context

The mathematical foundations of language modeling trace to Markov's (1913) pioneering work on sequential dependence in text. Shannon (1948) later formalized information-theoretic concepts—entropy, cross-entropy, and the noisy channel model—that remain central to NLP. Bayesian classification for text was developed independently by Maron (1961) for document categorization and by Mosteller and Wallace (1964) for authorship attribution. Logistic regression, originating in statistics (Cox, 1969), entered NLP in the 1990s under the name "maximum entropy modeling" (Berger et al., 1996).

3. Notation and Preliminaries

This section establishes the mathematical notation used throughout the chapter.

Sequences and Tokens

- $W = w_1, w_2, \ldots, w_n$ denotes a sequence of n tokens.
- w_i denotes the i-th token in the sequence.

- $w_{i:j}$ denotes the subsequence $w_i, w_{i+1}, \ldots, w_j$.
- $w_{<n}$ denotes all tokens preceding position n: $w_1, w_2, \ldots, w_{n-1}$.

Feature Vectors and Labels

- $\mathbf{x} = [x_1, x_2, \ldots, x_d]$ denotes a feature vector of dimension d.
- $y \in \{1, 2, \ldots, K\}$ denotes a discrete class label.
- $\hat{y}$ denotes the predicted class label.
- X denotes the feature matrix with rows $\mathbf{x}^{(i)}$ for training instances $i = 1, \ldots, m$.

Probability Notation

- $P(A)$ denotes the probability of event A.
- $P(A \mid B)$ denotes the conditional probability of A given B.
- $C(w)$ denotes the count of token w in a corpus.
- $C(w_i, w_j)$ denotes the count of bigram (w_i, w_j).

Statistical Measures

- $\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$ denotes the sample mean.
- V denotes the vocabulary size (number of unique token types).
- N denotes the total number of token instances in a corpus.

Vectors and Matrices

- Boldface lowercase $\mathbf{w}$ denotes a column vector.
- Boldface uppercase $\mathbf{W}$ denotes a matrix.
- $\mathbf{w} \cdot \mathbf{x}$ denotes the dot product: $\sum_{i=1}^{d} w_i x_i$.
- $\|\mathbf{w}\|_2 = \sqrt{\sum_i w_i^2}$ denotes the ℓ_2 (Euclidean) norm.
- $\|\mathbf{w}\|_1 = \sum_i |w_i|$ denotes the ℓ_1 (Manhattan) norm.

4. N-gram Language Models

This section formalizes n-gram language models, which estimate the probability of word sequences by exploiting limited context windows.

4.1 The Chain Rule of Probability

The goal of a language model is to assign a probability $P(W) = P(w_1, w_2, \ldots, w_n)$ to a sequence of words. By the **chain rule of probability**, any joint distribution can be decomposed into a product of conditional distributions:

$$P(w_{1:n}) = P(w_1) \cdot P(w_2 \mid w_1) \cdot P(w_3 \mid w_{1:2}) \cdots P(w_n \mid w_{1:n-1})$$

This can be written compactly as:

$$P(w_{1:n}) = \prod_{k=1}^{n} P(w_k \mid w_{1:k-1}) \tag{4.1}$$

Equation (4.1) is exact but intractable: estimating $P(w_n \mid w_{1:n-1})$ requires observing every possible history, which grows exponentially with sequence length.

4.2 The Markov Assumption

The **Markov assumption** approximates the full history by a fixed-length context window. For an n-gram model of order N, we assume:

$$P(w_n \mid w_{1:n-1}) \approx P(w_n \mid w_{n-N+1:n-1}) \tag{4.2}$$

Special cases include:

Model	Order N	Conditioning Context	Approximation
Unigram	1	None	$P(w_n)$
Bigram	2	Previous word	$P(w_n \mid w_{n-1})$
Trigram	3	Previous two words	$P(w_n \mid w_{n-2}, w_{n-1})$

For bigrams, the joint probability of a sequence becomes:

$$P(w_{1:n}) \approx \prod_{k=1}^{n} P(w_k \mid w_{k-1}) \tag{4.3}$$

where w_0 is typically a special start symbol $\langle s \rangle$.

4.3 Maximum Likelihood Estimation

Maximum Likelihood Estimation (MLE) estimates n-gram probabilities from corpus counts. For bigrams:

$$P_{\text{MLE}}(w_n \mid w_{n-1}) = \frac{C(w_{n-1}, w_n)}{C(w_{n-1})} \tag{4.4}$$

where $C(w_{n-1}, w_n)$ is the count of the bigram (w_{n-1}, w_n) and $C(w_{n-1})$ is the count of the unigram w_{n-1}.

General n-gram MLE:

$$P_{\text{MLE}}(w_n \mid w_{n-N+1:n-1}) = \frac{C(w_{n-N+1:n})}{C(w_{n-N+1:n-1})} \tag{4.5}$$

Worked Example

Consider the following corpus:

```
<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>
```

Unigram counts:

Token	$\langle s\rangle$	I	am	Sam	do	not	like	green	eggs	and	ham	$\langle /s\rangle$
Count	3	3	2	2	1	1	1	1	1	1	1	3

Selected bigram probabilities:

$$P(\mathrm{I} \mid \langle s\rangle) = \frac{C(\langle s\rangle, \mathrm{I})}{C(\langle s\rangle)} = \frac{2}{3} \approx 0.67$$

$$P(\mathrm{am} \mid \mathrm{I}) = \frac{C(\mathrm{I}, \mathrm{am})}{C(\mathrm{I})} = \frac{2}{3} \approx 0.67$$

$$P(\mathrm{Sam} \mid \mathrm{am}) = \frac{C(\mathrm{am}, \mathrm{Sam})}{C(\mathrm{am})} = \frac{1}{2} = 0.50$$

4.4 Perplexity

Perplexity is the standard intrinsic evaluation metric for language models. It measures how "surprised" the model is by a test sequence. Lower perplexity indicates a better model.

Definition: The perplexity of a language model on a test sequence $W = w_1, \ldots, w_N$ is:

$$\mathrm{PP}(W) = P(w_1, w_2, \ldots, w_N)^{-\frac{1}{N}} \tag{4.6}$$

Equivalently, using the chain rule:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i \mid w_{1:i-1})}} \tag{4.7}$$

For a bigram model:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i \mid w_{i-1})}} \tag{4.8}$$

Interpretation: Perplexity can be understood as the **weighted average branching factor**—the effective number of equally probable choices the model faces at each position. A perfect model that always assigns probability 1 to the correct word has perplexity 1.

Worked Example

Given the corpus from Section 4.3 with Laplace smoothing ($V = 6$), compute the perplexity of the sentence:

$$W = \langle s\rangle \text{ sentence another sentence } \langle /s\rangle$$

Smoothed bigram probabilities (using $P_{\text{Laplace}}(w_n \mid w_{n-1}) = \frac{C(w_{n-1}, w_n)+1}{C(w_{n-1})+V}$):

- $P(\text{sentence} \mid \langle s\rangle) = \frac{0+1}{3+6} = \frac{1}{9} \approx 0.111$
- $P(\text{another} \mid \text{sentence}) = \frac{0+1}{3+6} = \frac{1}{9} \approx 0.111$
- $P(\text{sentence} \mid \text{another}) = \frac{2+1}{2+6} = \frac{3}{8} = 0.375$

- $P(\langle/s\rangle \mid \text{sentence}) = \frac{3+1}{3+6} = \frac{4}{9} \approx 0.444$

Perplexity calculation:

$$\text{PP}(W) = \sqrt[4]{\frac{1}{0.111} \cdot \frac{1}{0.111} \cdot \frac{1}{0.375} \cdot \frac{1}{0.444}} = \sqrt[4]{487.46} \approx 4.70$$

4.5 Smoothing Techniques

The fundamental problem with MLE n-gram estimation is **data sparsity**: many valid n-grams never appear in the training corpus, yielding zero probability estimates. Since any zero probability in the product (Equation 4.3) forces the entire sequence probability to zero, smoothing is essential.

4.5.1 Laplace (Add-One) Smoothing

Laplace smoothing adds 1 to every n-gram count, ensuring no zero probabilities:

$$P_{\text{Laplace}}(w_n \mid w_{n-1}) = \frac{C(w_{n-1}, w_n) + 1}{C(w_{n-1}) + V} \tag{4.9}$$

where V is the vocabulary size.

Adjusted counts: Laplace smoothing can be interpreted as using adjusted counts:

$$C^*(w_{n-1}, w_n) = \frac{(C(w_{n-1}, w_n) + 1) \cdot C(w_{n-1})}{C(w_{n-1}) + V} \tag{4.10}$$

Discount ratio: The discount d applied to observed counts is:

$$d = \frac{C^*}{C} = \frac{C(w_{n-1}) + V \cdot C(w_{n-1})/(C(w_{n-1}) + V)}{C(w_{n-1}) + V} \tag{4.11}$$

Limitations: Laplace smoothing transfers too much probability mass from observed to unobserved events, particularly for large vocabularies. It is primarily useful for pedagogical purposes and applications where probabilities are not combined multiplicatively.

4.5.2 Add-k Smoothing

Add-k smoothing generalizes Laplace by adding a fractional count $k < 1$:

$$P_{\text{add-}k}(w_n \mid w_{n-1}) = \frac{C(w_{n-1}, w_n) + k}{C(w_{n-1}) + k \cdot V} \tag{4.12}$$

The parameter k is tuned on held-out data. Typical values range from 0.01 to 0.5.

4.5.3 Linear Interpolation

Linear interpolation combines multiple n-gram orders using weighted averages:

$$\hat{P}(w_n \mid w_{n-2}, w_{n-1}) = \lambda_1 P(w_n) + \lambda_2 P(w_n \mid w_{n-1}) + \lambda_3 P(w_n \mid w_{n-2}, w_{n-1})$$

where the interpolation weights satisfy $\sum_i \lambda_i = 1$ and $\lambda_i \geq 0$.

The weights λ are learned on held-out data using algorithms such as **Expectation-Maximization (EM)** to maximize the likelihood of the held-out set.

Context-conditioned interpolation: The weights can depend on the context:

$$\hat{P}(w_n \mid w_{n-2}, w_{n-1}) = \lambda_1(w_{n-2:n-1})P(w_n) + \lambda_2(w_{n-2:n-1})P(w_n \mid w_{n-1}) + \lambda_3(w_{n-2:n-1})P(w_n \mid w_{n-2:n-1})$$

This captures the intuition that sparse contexts should rely more heavily on lower-order models.

4.5.4 Backoff

Backoff uses higher-order n-grams when available, falling back to lower orders only when necessary:

$$P_{\text{backoff}}(w_n \mid w_{n-N+1:n-1}) = \begin{cases} P^*(w_n \mid w_{n-N+1:n-1}) & \text{if } C(w_{n-N+1:n}) > 0 \\ \alpha(w_{n-N+1:n-1}) \cdot P_{\text{backoff}}(w_n \mid w_{n-N+2:n-1}) & \text{otherwise} \end{cases}$$

where P^* is a discounted probability (e.g., using Good-Turing discounting) and α is a normalization factor ensuring the distribution sums to 1.

Stupid Backoff: A simplified variant used in large-scale systems (Brants et al., 2007) replaces probabilities with scores:

$$S(w_n \mid w_{n-k+1:n-1}) = \begin{cases} \frac{C(w_{n-k+1:n})}{C(w_{n-k+1:n-1})} & \text{if } C(w_{n-k+1:n}) > 0 \\ 0.4 \cdot S(w_n \mid w_{n-k+2:n-1}) & \text{otherwise} \end{cases} \tag{4.16}$$

The fixed backoff weight 0.4 is empirically effective for very large corpora.

4.5.5 Kneser-Ney Smoothing

Kneser-Ney smoothing (Kneser & Ney, 1995) is considered the most effective smoothing method. It combines absolute discounting with a novel continuation probability.

Key insight: The unigram probability $P(w)$ should not simply be the frequency of w, but rather reflect how likely w is to appear in novel contexts. Words like "Francisco" may be frequent but only after "San"; they should receive low continuation probability.

Continuation probability:

$$P_{\text{continuation}}(w) = \frac{|\{v : C(v, w) > 0\}|}{\sum_{w'} |\{v : C(v, w') > 0\}|} \tag{4.17}$$

This counts the number of different word types that precede w in the corpus.

Interpolated Kneser-Ney:

$$P_{\text{KN}}(w_n \mid w_{n-1}) = \frac{\max(C(w_{n-1}, w_n) - d, 0)}{C(w_{n-1})} + \lambda(w_{n-1})P_{\text{continuation}}(w_n) \tag{4.18}$$

where d is the absolute discount (typically 0.75) and λ is the interpolation weight:

$$\lambda(w_{n-1}) = \frac{d}{\sum_w C(w_{n-1}, w)} \cdot |\{w : C(w_{n-1}, w) > 0\}| \tag{4.19}$$

4.6 Entropy and Cross-Entropy

Information theory provides the foundation for understanding perplexity and evaluating language models.

4.6.1 Entropy

The **entropy** $H(X)$ of a discrete random variable X with probability distribution $p(x)$ measures the average information content (in bits):

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log_2 p(x) \tag{4.20}$$

Properties:

- $H(X) \geq 0$, with equality iff X is deterministic.
- $H(X) \leq \log_2 |\mathcal{X}|$, with equality iff X is uniform.
- Entropy measures uncertainty: higher entropy means more unpredictability.

Entropy rate of a language: For a stochastic process generating a sequence $W = w_1, w_2, \ldots$, the entropy rate is:

$$H(W) = -\lim_{n \to \infty} \frac{1}{n} \sum_{w_{1:n}} P(w_{1:n}) \log_2 P(w_{1:n}) \tag{4.21}$$

For a stationary ergodic process, this equals:

$$H(W) = \lim_{n \to \infty} H(w_n \mid w_1, \ldots, w_{n-1}) \tag{4.22}$$

4.6.2 Cross-Entropy

Cross-entropy measures the average number of bits needed to encode data from distribution p using a model m:

$$H(p, m) = -\sum_{x} p(x) \log_2 m(x) \tag{4.23}$$

For language modeling, we estimate the cross-entropy on a test corpus W:

$$H(W) = -\frac{1}{N} \sum_{i=1}^{N} \log_2 P(w_i \mid w_{1:i-1}) \tag{4.24}$$

Relationship to perplexity:

$$\text{PP}(W) = 2^{H(W)} \tag{4.25}$$

Thus, minimizing cross-entropy is equivalent to minimizing perplexity.

4.6.3 Bounds on Cross-Entropy

For any model m of a true distribution p:

$$H(p) \leq H(p, m) \tag{4.26}$$

The difference $H(p, m) - H(p) = D_{KL}(p \| m) \geq 0$ is the **Kullback-Leibler divergence**, measuring how much the model differs from the true distribution.

5. Naive Bayes Classification

This section develops the Naive Bayes classifier, a generative model for text classification that exploits conditional independence assumptions for computational tractability.

5.1 Bayesian Inference and Bayes' Rule

The classification task assigns a document d to a class $c \in C$. A **generative classifier** models the joint probability $P(d, c)$ and uses Bayes' rule to compute the posterior:

$$P(c \mid d) = \frac{P(d \mid c)P(c)}{P(d)} \tag{5.1}$$

Since $P(d)$ is constant across classes, classification reduces to:

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}}\, P(d \mid c)P(c) \tag{5.2}$$

The components are:

- **Prior** $P(c)$: probability of class c before observing any features.
- **Likelihood** $P(d \mid c)$: probability of generating document d given class c.
- **Posterior** $P(c \mid d)$: probability of class c after observing d.

5.2 The Naive Bayes Assumption

Representing a document d as a sequence of features (words) $f_1, f_2, \ldots, f_n$, the likelihood becomes:

$$P(d \mid c) = P(f_1, f_2, \ldots, f_n \mid c) \tag{5.3}$$

The **naive Bayes assumption** asserts that features are conditionally independent given the class:

$$P(f_1, \ldots, f_n \mid c) = \prod_{i=1}^{n} P(f_i \mid c) \tag{5.4}$$

This simplification—though unrealistic—enables efficient estimation and often yields surprisingly effective classifiers.

The Naive Bayes Classifier:

$$c_{\text{NB}} = \underset{c \in C}{\operatorname{argmax}}\, P(c) \prod_{i \in \text{positions}} P(w_i \mid c) \tag{5.5}$$

Log-space computation: To avoid numerical underflow from multiplying small probabilities:

$$c_{\text{NB}} = \underset{c \in C}{\operatorname{argmax}} \left[\log P(c) + \sum_{i \in \text{positions}} \log P(w_i \mid c) \right] \tag{5.6}$$

5.3 Training Naive Bayes

Training involves estimating the prior $P(c)$ and likelihood $P(w \mid c)$ from labeled data.

Prior estimation:

$$\hat{P}(c) = \frac{N_c}{N_{\text{doc}}} \tag{5.7}$$

where N_c is the number of documents in class c and N_{doc} is the total number of documents.

Likelihood estimation with add-one smoothing:

$$\hat{P}(w_i \mid c) = \frac{\text{count}(w_i, c) + 1}{\sum_{w \in V}(\text{count}(w, c) + 1)} = \frac{\text{count}(w_i, c) + 1}{\left(\sum_{w \in V} \text{count}(w, c)\right) + |V|} \tag{5.8}$$

where $\text{count}(w_i, c)$ is the number of times word w_i occurs in documents of class c, and $|V|$ is the vocabulary size.

Training algorithm:

```
TRAIN_NAIVE_BAYES(D, C):
    for each class c ∈ C:
        N_doc[c] ← count of documents in D with class c
        prior[c] ← N_doc[c] / |D|

        V ← vocabulary of all documents in D
        bigdoc[c] ← concatenation of all documents with class c

        for each word w ∈ V:
            count[w,c] ← occurrences of w in bigdoc[c]
            likelihood[w,c] ← (count[w,c] + 1) / (|bigdoc[c]| + |V|)

    return prior, likelihood, V
```

Classification algorithm:

```
CLASSIFY_NAIVE_BAYES(d, prior, likelihood, C, V):
    for each class c ∈ C:
        score[c] ← log(prior[c])
        for each position i in document d:
            score[c] ← score[c] + log(likelihood[d[i], c])

    return argmax_c score[c]
```

5.4 Multinomial vs. Bernoulli Naive Bayes

Two variants of Naive Bayes differ in their document representation:

Multinomial Naive Bayes:

- Features are word counts (or frequencies).
- Each word position is an independent trial.
- The likelihood $P(d \mid c)$ is a product over all word positions.
- Suitable when word frequency is informative.

Bernoulli (Binary) Naive Bayes:

- Features are binary: word present (1) or absent (0).
- Explicitly models word absence:

$$P(d \mid c) = \prod_{w \in V} \left[P(w \mid c)^{b_w} \cdot (1 - P(w \mid c))^{1-b_w} \right] \tag{5.9}$$

where $b_w = 1$ if word w appears in d, else 0.

Comparison: Binary Naive Bayes often outperforms multinomial for short texts (e.g., tweets) where word presence is more informative than frequency.

5.5 Sentiment Analysis Applications

Sentiment analysis classifies text by expressed opinion (positive, negative, neutral). Naive Bayes is a common baseline.

Example: Movie review classification.

Class	Documents
Positive (+)	"just plain boring", "entirely predictable and lacks energy", "no surprises and very few laughs"
Negative (−)	"very powerful", "the most fun film of the summer"

Training computations:

$$P(+) = \frac{3}{5} = 0.6, \quad P(-) = \frac{2}{5} = 0.4$$

Vocabulary: $V = \{\text{just}, \text{plain}, \text{boring}, \ldots\}$, $|V| = 20$.

Word likelihoods (with add-1 smoothing):

$$P(\text{predictable} \mid -) = \frac{1+1}{14+20} = \frac{2}{34} \approx 0.059$$

$$P(\text{predictable} \mid +) = \frac{0+1}{9+20} = \frac{1}{29} \approx 0.034$$

Classification of test document "predictable with no fun":

$$P(-) \prod_i P(w_i \mid -) = 0.6 \times 0.059 \times 0.029 \times 0.059 \times 0.029 = 1.8 \times 10^{-6}$$

$$P(+) \prod_i P(w_i \mid +) = 0.4 \times 0.034 \times 0.034 \times 0.034 \times 0.103 = 1.6 \times 10^{-7}$$

The document is classified as **negative** since $1.8 \times 10^{-6} > 1.6 \times 10^{-7}$.

6. Logistic Regression

This section develops logistic regression, a discriminative classifier that directly models the posterior probability $P(y \mid \mathbf{x})$ without modeling the class-conditional distributions.

6.1 Generative vs. Discriminative Classifiers

Generative classifiers (e.g., Naive Bayes) model the joint distribution $P(\mathbf{x}, y)$ and use Bayes' rule:

$$P(y \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid y)P(y)}{P(\mathbf{x})} \tag{6.1}$$

Discriminative classifiers (e.g., logistic regression) model the posterior $P(y \mid \mathbf{x})$ directly.

Aspect	Generative	Discriminative
Models	$P(\mathbf{x} \mid y)$ and $P(y)$	$P(y \mid \mathbf{x})$ directly
Training	Count-based, often faster	Optimization-based
Missing features	Handles naturally	Requires imputation
Performance	Often competitive	Often better with sufficient data

Key advantage of discriminative models: They focus modeling capacity on the decision boundary rather than the full data distribution.

6.2 The Sigmoid Function

For binary classification with $y \in \{0, 1\}$, logistic regression models the probability of class 1 using the **sigmoid** (logistic) function:

$$\sigma(z) = \frac{1}{1 + e^{-z}} = \frac{e^z}{1 + e^z} \tag{6.2}$$

Properties of the sigmoid:

- Range: $(0, 1)$, making it suitable for probabilities.
- Symmetric: $\sigma(-z) = 1 - \sigma(z)$.
- Derivative: $\sigma'(z) = \sigma(z)(1 - \sigma(z))$.
- Monotonic: strictly increasing.

Logistic regression model:

$$P(y = 1 \mid \mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x} + b) = \frac{1}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)}} \tag{6.3}$$

where:

- $\mathbf{w} = [w_1, w_2, \ldots, w_d]$ is the weight vector.
- b is the bias term.
- $\mathbf{x} = [x_1, x_2, \ldots, x_d]$ is the feature vector.

The decision boundary is the hyperplane where $P(y = 1 \mid \mathbf{x}) = 0.5$, i.e., $\mathbf{w} \cdot \mathbf{x} + b = 0$.

Log-odds interpretation:

$$\log \frac{P(y = 1 \mid \mathbf{x})}{P(y = 0 \mid \mathbf{x})} = \mathbf{w} \cdot \mathbf{x} + b \tag{6.4}$$

The linear combination $\mathbf{w} \cdot \mathbf{x} + b$ equals the log-odds (logit) of class 1.

6.3 The Softmax Function

For **multinomial** (multiclass) classification with $y \in \{1, 2, \ldots, K\}$, the sigmoid generalizes to the **softmax** function:

$$P(y = k \mid \mathbf{x}) = \frac{e^{\mathbf{w}_k \cdot \mathbf{x} + b_k}}{\sum_{j=1}^{K} e^{\mathbf{w}_j \cdot \mathbf{x} + b_j}} \tag{6.5}$$

where $\mathbf{w}_k$ and b_k are the weight vector and bias for class k.

Properties:

- Outputs a valid probability distribution: $\sum_{k=1}^{K} P(y = k \mid \mathbf{x}) = 1$.
- Reduces to sigmoid when $K = 2$.
- Amplifies differences: larger logits receive disproportionately more probability.

Numerical stability: In practice, softmax is computed as:

$$P(y = k \mid \mathbf{x}) = \frac{e^{z_k - \max_j z_j}}{\sum_{j=1}^{K} e^{z_j - \max_j z_j}} \tag{6.6}$$

where $z_k = \mathbf{w}_k \cdot \mathbf{x} + b_k$.

6.4 Cross-Entropy Loss

Training logistic regression requires a **loss function** measuring the discrepancy between predicted probabilities $\hat{y}$ and true labels y.

6.4.1 Binary Cross-Entropy

For binary classification, the **cross-entropy loss** (negative log-likelihood) for a single instance is:

$$L_{\text{CE}}(\hat{y}, y) = -\left[y \log \hat{y} + (1 - y) \log(1 - \hat{y})\right] \tag{6.7}$$

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

Interpretation: When $y = 1$, the loss is $-\log \hat{y}$; when $y = 0$, the loss is $-\log(1 - \hat{y})$. The loss is 0 when predictions are perfect and increases as predictions deviate from truth.

Total loss over dataset:

$$\mathcal{L}(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})\right] \tag{6.8}$$

where $\theta = \{\mathbf{w}, b\}$ and m is the number of training examples.

6.4.2 Categorical Cross-Entropy

For multiclass classification with one-hot encoded labels $\mathbf{y} = [y_1, \ldots, y_K]$ where $y_k = 1$ for the true class:

$$L_{\mathrm{CE}}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{k=1}^{K} y_k \log \hat{y}_k \tag{6.9}$$

Since only one $y_k = 1$, this simplifies to:

$$L_{\mathrm{CE}} = -\log \hat{y}_c \tag{6.10}$$

where c is the true class.

6.5 Gradient Descent Optimization

Unlike Naive Bayes, logistic regression lacks a closed-form solution. Training proceeds via **gradient descent**, iteratively updating parameters to minimize the loss.

6.5.1 The Gradient

For binary logistic regression, the gradient of the loss with respect to weight w_j is:

$$\frac{\partial \mathcal{L}}{\partial w_j} = \frac{1}{m} \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)} \right) x_j^{(i)} \tag{6.11}$$

For the bias:

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{1}{m} \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)} \right) \tag{6.12}$$

In vector form:

$$\nabla_{\mathbf{w}} \mathcal{L} = \frac{1}{m} \mathbf{X}^{\top} (\hat{\mathbf{y}} - \mathbf{y}) \tag{6.13}$$

where $\mathbf{X}$ is the $m \times d$ feature matrix.

6.5.2 Gradient Descent Algorithm

Batch gradient descent updates all parameters after computing the gradient over the entire dataset:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_{\theta} \mathcal{L}(\theta^{(t)}) \tag{6.14}$$

where η is the **learning rate**.

Algorithm:

```
GRADIENT_DESCENT(X, y, η, num_iterations):
    Initialize θ = [w, b] randomly or to zeros

    for t = 1 to num_iterations:
        ŷ ← σ(X·w + b)                  // Forward pass
        ∇L ← compute_gradient(X, y, ŷ)   // Compute gradient
        θ ← θ - η · ∇L                   // Update parameters

    return θ
```

6.5.3 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) approximates the gradient using a single training example:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta L(\theta^{(t)}; x^{(i)}, y^{(i)}) \tag{6.15}$$

Advantages:

- Faster convergence for large datasets.
- Inherent regularization from noise.
- Can escape local minima.

Disadvantages:

- High variance in updates.
- Requires careful learning rate tuning.

Mini-batch SGD uses a small batch of B examples, balancing efficiency and variance:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \cdot \frac{1}{B} \sum_{i \in \mathcal{B}} \nabla_\theta L(\theta^{(t)}; x^{(i)}, y^{(i)}) \tag{6.16}$$

Typical batch sizes range from 16 to 256.

6.6 Regularization

Regularization prevents overfitting by penalizing large parameter values.

6.6.1 L2 Regularization (Ridge)

L2 regularization adds the squared ℓ_2 norm of weights to the loss:

$$\mathcal{L}_{\text{reg}}(\theta) = \mathcal{L}(\theta) + \lambda \sum_{j=1}^{d} w_j^2 = \mathcal{L}(\theta) + \lambda \|\mathbf{w}\|_2^2 \tag{6.17}$$

where $\lambda > 0$ is the regularization strength.

Effect: L2 regularization shrinks weights toward zero but rarely sets them exactly to zero. The gradient contribution from regularization is:

$$\frac{\partial}{\partial w_j} \lambda \|\mathbf{w}\|_2^2 = 2\lambda w_j \tag{6.18}$$

6.6.2 L1 Regularization (Lasso)

L1 regularization adds the ℓ_1 norm of weights:

$$\mathcal{L}_{\text{reg}}(\theta) = \mathcal{L}(\theta) + \lambda \sum_{j=1}^{d} |w_j| = \mathcal{L}(\theta) + \lambda \|\mathbf{w}\|_1 \tag{6.19}$$

Effect: L1 regularization induces **sparsity**—many weights become exactly zero, performing automatic feature selection.

6.6.3 Elastic Net

Elastic Net combines L1 and L2 regularization:

$$\mathcal{L}_{\text{reg}}(\theta) = \mathcal{L}(\theta) + \lambda_1 \|\mathbf{w}\|_1 + \lambda_2 \|\mathbf{w}\|_2^2 \tag{6.20}$$

This combines sparsity (from L1) with stability (from L2).

Regularization	Effect	Sparsity	Use Case
None	Risk of overfitting	No	Small, clean datasets
L2 (Ridge)	Shrinks weights	No	Many small effects
L1 (Lasso)	Shrinks and zeros	Yes	Feature selection
Elastic Net	Combined	Partial	High-dimensional, correlated features

7. Decision Trees

Decision trees are non-parametric classifiers that partition the feature space via recursive binary splits. The **ID3 algorithm** constructs trees using information-theoretic criteria.

7.1 Entropy and Information Gain

Entropy measures the impurity of a node (see Equation 4.20):

$$H(Y) = -\sum_{k=1}^{K} p_k \log_2 p_k \tag{7.1}$$

where p_k is the proportion of instances belonging to class k.

Conditional entropy measures impurity after splitting on feature X:

$$H(Y \mid X) = \sum_{v \in \text{values}(X)} \frac{|S_v|}{|S|} H(Y_{S_v}) \tag{7.2}$$

where S_v is the subset of instances with feature value v.

Information gain is the reduction in entropy from splitting on X:

$$\text{IG}(Y, X) = H(Y) - H(Y \mid X) \tag{7.3}$$

Higher information gain indicates a more informative feature.

7.2 The ID3 Algorithm

ID3 (Iterative Dichotomiser 3) greedily selects the feature with maximum information gain at each node:

```
ID3(S, Features, Target):
    if all instances in S have same Target value:
        return Leaf(Target value)
    if Features is empty:
        return Leaf(majority Target in S)

    A ← feature in Features with maximum IG(Target, A)
    tree ← new Decision Node(A)

    for each value v of A:
        S_v ← subset of S where A = v
        if S_v is empty:
            tree.add_child(v, Leaf(majority Target in S))
        else:
            tree.add_child(v, ID3(S_v, Features - {A}, Target))

    return tree
```

Worked Example: Play Tennis

Outlook	Temperature	Humidity	Wind	Play?
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

Step 1: Compute root entropy

$$H(\text{Play}) = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} \approx 0.940 \text{ bits}$$

Step 2: Compute information gain for each feature

For *Outlook* with values {Sunny, Overcast, Rain}:

- Sunny: 5 instances, 2 Yes, 3 No → $H = 0.971$
- Overcast: 4 instances, 4 Yes, 0 No → $H = 0.0$
- Rain: 5 instances, 3 Yes, 2 No → $H = 0.971$

$$H(\text{Play} \mid \text{Outlook}) = \frac{5}{14}(0.971) + \frac{4}{14}(0) + \frac{5}{14}(0.971) = 0.693$$

$$\text{IG}(\text{Play}, \text{Outlook}) = 0.940 - 0.693 = 0.247 \text{ bits}$$

Similarly:

- $\mathrm{IG}(\mathrm{Play}, \mathrm{Temperature}) = 0.029$
- $\mathrm{IG}(\mathrm{Play}, \mathrm{Humidity}) = 0.152$
- $\mathrm{IG}(\mathrm{Play}, \mathrm{Wind}) = 0.048$

Step 3: Select root

Outlook has maximum IG, so it becomes the root.

7.3 Advantages and Limitations

Advantages:

- Interpretable (can be visualized as flowcharts).
- Handles mixed feature types.
- No feature scaling required.
- Implicit feature selection.

Limitations:

- Prone to overfitting (deep trees memorize training data).
- Unstable (small data changes → different trees).
- Greedy construction may miss global optima.
- Bias toward features with many values.

Mitigation: Pruning, ensemble methods (Random Forests, Gradient Boosting).

8. Support Vector Machines

Support Vector Machines (SVMs) find the maximum-margin hyperplane separating classes. They are particularly effective in high-dimensional spaces.

8.1 Maximum Margin Classifiers

For linearly separable data, infinitely many hyperplanes separate the classes. SVM selects the hyperplane with **maximum margin**—the largest distance to the nearest training points.

Decision boundary:

$$\mathbf{w} \cdot \mathbf{x} + b = 0 \tag{8.1}$$

Classification rule:

$$\hat{y} = \mathrm{sign}(\mathbf{w} \cdot \mathbf{x} + b) \tag{8.2}$$

Margin: The distance from the hyperplane to the closest point is $\frac{1}{\|\mathbf{w}\|}$. The margin is $\frac{2}{\|\mathbf{w}\|}$.

Support vectors are the training instances lying on the margin boundaries. The decision boundary depends only on these points.

Optimization problem:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{subject to} \quad y^{(i)}(\mathbf{w} \cdot \mathbf{x}^{(i)} + b) \geq 1 \quad \forall i \tag{8.3}$$

8.2 The Kernel Trick

For non-linearly separable data, SVMs use the **kernel trick**: map inputs to a higher-dimensional space where linear separation is possible.

Kernel function:

$$K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{z}) \tag{8.4}$$

where ϕ is the (implicit) mapping to feature space.

Common kernels:

Kernel	Formula	Use Case
Linear	$K(\mathbf{x}, \mathbf{z}) = \mathbf{x} \cdot \mathbf{z}$	Linearly separable
Polynomial	$K(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z} + c)^d$	Moderate nonlinearity
RBF (Gaussian)	$K(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{\lvert\mathbf{x}-\mathbf{z}\rvert^2}{2\sigma^2}\right)$	Complex boundaries

Soft-margin SVM: For noisy data, allow some misclassifications by introducing slack variables ξ_i:

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m} \xi_i \tag{8.5}$$

where C controls the trade-off between margin size and misclassification penalty.

9. k-Nearest Neighbors

k-Nearest Neighbors (k-NN) is an instance-based (lazy) learner that classifies new points based on the labels of the k closest training examples.

9.1 Distance Metrics

The choice of distance metric significantly affects k-NN performance.

Euclidean distance (ℓ_2):

$$d(\mathbf{x}, \mathbf{z}) = \sqrt{\sum_{j=1}^{d}(x_j - z_j)^2} = \|\mathbf{x} - \mathbf{z}\|_2 \tag{9.1}$$

Manhattan distance (ℓ_1):

$$d(\mathbf{x}, \mathbf{z}) = \sum_{j=1}^{d} |x_j - z_j| = \|\mathbf{x} - \mathbf{z}\|_1 \tag{9.2}$$

Cosine distance (for text):

$$d(\mathbf{x}, \mathbf{z}) = 1 - \frac{\mathbf{x} \cdot \mathbf{z}}{\|\mathbf{x}\|\|\mathbf{z}\|} \tag{9.3}$$

Minkowski distance (generalized):

$$d(\mathbf{x}, \mathbf{z}) = \left(\sum_{j=1}^{d} |x_j - z_j|^p \right)^{1/p} \tag{9.4}$$

where $p = 1$ gives Manhattan and $p = 2$ gives Euclidean.

9.2 The k-NN Algorithm

```
k-NN(X_train, y_train, x_new, k):
    distances ← []
    for each (x_i, y_i) in training data:
        distances.append((d(x_new, x_i), y_i))

    distances.sort()
    neighbors ← distances[:k]

    return majority_vote([y for (d, y) in neighbors])
```

Weighted k-NN: Weight votes by inverse distance:

$$\hat{y} = \underset{c}{\operatorname{argmax}} \sum_{i \in N_k} \frac{\mathbf{1}[y_i = c]}{d(\mathbf{x}, \mathbf{x}_i)} \tag{9.5}$$

9.3 Choosing k

- **Small k** (e.g., 1): Low bias, high variance (sensitive to noise).
- **Large k** (e.g., n): High bias, low variance (over-smoothed).
- **Odd k**: Avoids ties in binary classification.
- **Typical values**: $k = 1, 3, 5, 7$ or $k = \sqrt{n}$.

Selection: Use cross-validation to find optimal k.

10. Linear Regression

Although primarily a regression technique, **linear regression** provides the foundation for understanding gradient-based optimization and regularization.

10.1 Ordinary Least Squares

Model:

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_d x_d = \beta_0 + \boldsymbol{\beta}^\top \mathbf{x} \tag{10.1}$$

Loss function (Mean Squared Error):

$$\text{MSE} = \frac{1}{m} \sum_{i=1}^{m} (y^{(i)} - \hat{y}^{(i)})^2 \tag{10.2}$$

Closed-form solution (normal equations):

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y} \tag{10.3}$$

where $\mathbf{X}$ is the design matrix with a column of 1s for the intercept.

10.2 Simple Linear Regression Formulas

For single-variable regression with $(x_1, y_1), \ldots, (x_n, y_n)$:

Slope:

$$\beta_1 = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n} (x_i - \bar{x})^2} = \frac{\text{Cov}(X, Y)}{\text{Var}(X)} \tag{10.4}$$

Intercept:

$$\beta_0 = \bar{y} - \beta_1 \bar{x} \tag{10.5}$$

10.3 Evaluation Metrics

Mean Absolute Error (MAE):

$$\text{MAE} = \frac{1}{m} \sum_{i=1}^{m} |y^{(i)} - \hat{y}^{(i)}| \tag{10.6}$$

Root Mean Squared Error (RMSE):

$$\text{RMSE} = \sqrt{\frac{1}{m} \sum_{i=1}^{m} (y^{(i)} - \hat{y}^{(i)})^2} \tag{10.7}$$

Coefficient of determination (R^2):

$$R^2 = 1 - \frac{\sum_i (y^{(i)} - \hat{y}^{(i)})^2}{\sum_i (y^{(i)} - \bar{y})^2} = 1 - \frac{\text{SS}_{\text{res}}}{\text{SS}_{\text{tot}}} \tag{10.8}$$

$R^2 \in [0, 1]$ measures the proportion of variance explained by the model.

11. Model Evaluation and Generalization

This section formalizes evaluation methodology, addressing how to measure classifier performance and ensure models generalize to unseen data.

11.1 Precision, Recall, and F-measure

For binary classification with classes positive (+) and negative (−):

Confusion matrix:

	Predicted +	Predicted −
Actual +	True Positive (TP)	False Negative (FN)
Actual −	False Positive (FP)	True Negative (TN)

Accuracy:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{11.1}$$

Precision (positive predictive value):

$$P = \frac{TP}{TP + FP} \tag{11.2}$$

"Of all instances predicted positive, how many are actually positive?"

Recall (sensitivity, true positive rate):

$$R = \frac{TP}{TP + FN} \tag{11.3}$$

"Of all actual positive instances, how many did we correctly identify?"

F-measure (harmonic mean of precision and recall):

$$F_1 = \frac{2 \cdot P \cdot R}{P + R} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \tag{11.4}$$

Generalized F-measure (F_β):

$$F_\beta = \frac{(1 + \beta^2) \cdot P \cdot R}{\beta^2 \cdot P + R} \tag{11.5}$$

- $\beta = 1$: Equal weight to precision and recall.
- $\beta < 1$: Emphasizes precision.
- $\beta > 1$: Emphasizes recall.

Multiclass extension:

- **Macro-averaged**: Compute metric for each class, then average.
- **Micro-averaged**: Aggregate TP, FP, FN across classes, then compute.

11.2 Training, Validation, and Test Sets

Three-way split:

Set	Purpose	Typical %
Training	Fit model parameters	60–80%
Validation (Development)	Tune hyperparameters	10–20%
Test	Final unbiased evaluation	10–20%

Critical principle: Test set should never influence model selection or hyperparameter tuning.

11.3 Cross-Validation

k-fold cross-validation partitions data into k folds, using each as validation once:

```
K-FOLD-CROSS-VALIDATION(D, k, algorithm):
    split D into k equal folds F_1, ..., F_k

    for i = 1 to k:
        train_set ← D - F_i
        validation_set ← F_i
        model_i ← algorithm.train(train_set)
        score_i ← algorithm.evaluate(model_i, validation_set)

    return mean(score_1, ..., score_k)
```

Common choices:

- $k = 10$ (10-fold): Standard choice.
- $k = 5$: For larger datasets.
- $k = n$ (leave-one-out, LOOCV): Maximum training data, high variance.

Stratified k-fold: Maintains class proportions in each fold—important for imbalanced datasets.

11.4 Overfitting and Underfitting

Overfitting: Model fits training data too closely, memorizing noise.

- Symptoms: Low training error, high test error.
- Causes: Model too complex, insufficient data, no regularization.
- Solutions: Regularization, more data, simpler models, early stopping.

Underfitting: Model is too simple to capture underlying patterns.

- Symptoms: High training error, high test error.
- Causes: Model too simple, insufficient features.
- Solutions: More complex models, better features, longer training.

Bias-variance trade-off:

$$\text{Expected Error} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Noise} \tag{11.6}$$

- **Bias**: Error from incorrect model assumptions (underfitting).
- **Variance**: Error from sensitivity to training data (overfitting).

11.5 Statistical Significance Testing

When comparing two classifiers A and B, observed performance differences may be due to chance. **Statistical significance testing** determines whether differences are meaningful.

Paired bootstrap test:

1. Sample b bootstrap datasets from test set (with replacement).
2. For each bootstrap sample, compute $\delta(x) = \text{eval}(A, x) - \text{eval}(B, x)$.
3. Compute $s = \frac{1}{b} \sum_i \mathbf{1}[\delta(x_i) > 2\bar{\delta}]$, where $\bar{\delta}$ is the mean.
4. p-value $\approx s$.

Interpretation: If $p < 0.05$ (or chosen threshold α), the difference is statistically significant.

12. Conclusion

This chapter has presented the mathematical foundations of statistical language modeling and text classification. Key contributions include:

Language modeling: N-gram models approximate sequence probabilities via the Markov assumption. Maximum likelihood estimation provides the baseline, while smoothing techniques—Laplace, add-k, interpolation, and Kneser-Ney—address the zero-frequency problem inherent in finite corpora. Perplexity, grounded in cross-entropy, serves as the standard intrinsic evaluation metric.

Generative classification: Naive Bayes classifiers exploit conditional independence to enable efficient probabilistic text classification. Despite the unrealistic independence assumption, Naive Bayes often achieves competitive performance, particularly for high-dimensional text data.

Discriminative classification: Logistic regression directly models class posteriors using the sigmoid (binary) or softmax (multiclass) functions. Optimization via stochastic gradient descent on the cross-entropy loss, combined with L1/L2 regularization, yields robust classifiers.

Additional classifiers: Decision trees (ID3) provide interpretable rule-based classification via entropy-based splits. Support Vector Machines find maximum-margin separating hyperplanes, with kernel methods enabling nonlinear decision boundaries. k-Nearest Neighbors offers a simple instance-based alternative.

Evaluation: Rigorous evaluation methodology—including precision-recall-F1 metrics, cross-validation, and statistical significance testing—ensures reliable model comparison and selection.

Together, these methods constitute the foundational toolkit for building NLP classification systems, with applications ranging from sentiment analysis and spam detection to named entity recognition and machine translation.

13. References

- Berger, A., Della Pietra, S., & Della Pietra, V. (1996). A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1), 39–71.
- Brants, T., Popat, A. C., Xu, P., Och, F. J., & Dean, J. (2007). Large language models in machine translation. *Proceedings of EMNLP-CoNLL*, 858–867.
- Chen, S. F., & Goodman, J. (1999). An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4), 359–394.
- Cox, D. R. (1969). *Analysis of Binary Data*. Chapman and Hall.
- Jurafsky, D., & Martin, J. H. (2024). *Speech and Language Processing* (3rd ed., draft). Chapter 3: N-gram Language Models; Chapter 4: Naive Bayes, Text Classification, and Sentiment; Chapter 5: Logistic Regression.
- Kneser, R., & Ney, H. (1995). Improved backing-off for m-gram language modeling. *Proceedings of ICASSP*, 181–184.
- Markov, A. A. (1913). An example of statistical investigation of the text of "Eugene Onegin" illustrating the connection of trials in chains.
- Maron, M. E. (1961). Automatic indexing: An experimental inquiry. *Journal of the ACM*, 8(3), 404–417.
- McCallum, A., & Nigam, K. (1998). A comparison of event models for Naive Bayes text classification. *AAAI/ICML Workshop on Learning for Text Categorization*.
- Mosteller, F., & Wallace, D. L. (1964). *Inference and Disputed Authorship: The Federalist*. Addison-Wesley.
- Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1), 81–106.
- Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27(3), 379–423.
- Vapnik, V. (1995). *The Nature of Statistical Learning Theory*. Springer.

Appendix A: Derivation of the Gradient for Logistic Regression

This appendix derives the gradient of the cross-entropy loss with respect to the parameters of logistic regression.

Setup: For binary classification with model $\hat{y} = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$ and loss:

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

Step 1: Derivative of sigmoid

$$\frac{d\sigma}{dz} = \sigma(z)(1 - \sigma(z))$$

Step 2: Derivative of loss with respect to $z = \mathbf{w} \cdot \mathbf{x} + b$

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z}$$

$$\frac{\partial L}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{1 - y}{1 - \hat{y}}$$

$$\frac{\partial L}{\partial z} = \left(-\frac{y}{\hat{y}} + \frac{1 - y}{1 - \hat{y}} \right) \cdot \hat{y}(1 - \hat{y})$$

$$= -y(1-\hat{y}) + (1-y)\hat{y} = \hat{y} - y$$

Step 3: Gradient with respect to parameters

$$\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial w_j} = (\hat{y} - y) \cdot x_j$$

$$\frac{\partial L}{\partial b} = \hat{y} - y$$

Result: The gradient has the elegant form $(\hat{y} - y)\mathbf{x}$, enabling efficient computation.

Appendix B: Smoothing Algorithms Summary

Method	Formula	Pros	Cons
Laplace (add-1)	$\frac{C(w_{n-1},w_n)+1}{C(w_{n-1})+V}$	Simple, no tuning	Over-discounts, poor for large V
Add-k	$\frac{C(w_{n-1},w_n)+k}{C(w_{n-1})+kV}$	Tunable	Requires held-out data
Interpolation	$\sum_i \lambda_i P_i(w_n \mid \text{context})$	Combines orders	Many parameters
Kneser-Ney	Discounting + continuation	State-of-the-art	Complex implementation
Stupid Backoff	Fixed backoff (0.4)	Simple, scales	Not a probability

Recommendation: Use Kneser-Ney for production systems; use add-k or interpolation for teaching/prototyping.

Glossary

Term	Definition
Backoff	Smoothing method using lower-order n-grams only when higher-order counts are zero
Bag-of-words	Document representation ignoring word order, using only word counts
Bias	Systematic error from incorrect model assumptions
Cross-entropy	Loss function measuring divergence between predicted and true distributions
Cross-validation	Evaluation technique using rotated train/test splits
Discriminative classifier	Models $P(y \mid \mathbf{x})$ directly without modeling $P(\mathbf{x})$
Entropy	Measure of uncertainty or information content of a distribution
F-measure	Harmonic mean of precision and recall
Generative classifier	Models joint distribution $P(\mathbf{x}, y)$ and uses Bayes' rule
Gradient descent	Iterative optimization by moving in the direction of steepest descent
Information gain	Reduction in entropy from splitting on a feature
Interpolation	Smoothing by combining n-gram models of different orders
Kernel trick	Implicit mapping to high-dimensional space via kernel functions
Laplace smoothing	Adding 1 to all counts to avoid zero probabilities
Likelihood	Probability of observed data given model parameters
Logistic regression	Discriminative classifier using sigmoid/softmax activation
Markov assumption	Probability of a word depends only on the preceding $N-1$ words

Term	Definition
Maximum likelihood estimation (MLE)	Parameter estimation by maximizing data likelihood
Naive Bayes	Generative classifier assuming conditional feature independence
N-gram	Contiguous sequence of N tokens
Overfitting	Model memorizes training data, failing to generalize
Perplexity	Intrinsic LM evaluation: inverse geometric mean probability
Posterior	Probability of hypothesis given evidence (after Bayes' rule)
Precision	Fraction of positive predictions that are correct
Prior	Probability of hypothesis before observing evidence
Recall	Fraction of actual positives correctly identified
Regularization	Penalty on model complexity to prevent overfitting
Sigmoid	Function $\sigma(z) = 1/(1 + e^{-z})$ mapping reals to $(0, 1)$
Smoothing	Techniques redistributing probability mass to unseen events
Softmax	Generalization of sigmoid to multiple classes
Support vectors	Training instances lying on the SVM margin boundaries
Underfitting	Model too simple to capture data patterns
Variance	Error from sensitivity to training set fluctuations