CS336: Language Modeling from Scratch

Lecture 14: Data 2 - Filtering and Deduplication

Stanford University - Spring 2025

Abstract

Abstract: This lecture provides a deep technical dive into the algorithmic foundations of data processing for language models, focusing on filtering and deduplication techniques. We explore three fundamental filtering approaches: n-gram models with Kneser-Ney smoothing, fastText linear classification, and importance sampling methods. The lecture then covers deduplication algorithms including exact matching with Bloom filters and approximate matching using MinHash with Locality Sensitive Hashing (LSH). Each method is analyzed for computational complexity, practical implementation considerations, and trade-offs between accuracy and efficiency.

Contents

1 Introduction: Data Processing Mechanics

1.1 Lecture Focus and Scope

Building on Lecture 13: Previous lecture covered historical evolution and high-level data sources. This lecture focuses on the technical mechanics of how data processing actually works.

Key Principle: Data doesn't fall from the sky - it exists in live services, must be explicitly crawled or dumped, and requires extensive processing pipeline implementation.

Processing Pipeline Components:

- HTML-to-text conversion with tool selection impact
- Quality filtering using multiple algorithmic approaches
- Language identification and classification
- Toxicity filtering for safety compliance
- Deduplication at various granularities

Algorithmic Focus: This lecture emphasizes classical big data processing algorithms with mathematical foundations rather than high-level data strategy.

2 Filtering Algorithm Framework

2.1 Abstract Filtering Primitive

Problem Formulation: Given:

- Target dataset T (small, high-quality)
- Raw dataset R (large, mixed quality - e.g., Common Crawl)

Goal: Find subset T' of R such that $T' \sim T$ (similar distribution)

Algorithm Requirements:

1. **Generalization:** Must extrapolate beyond exact T examples
2. **Efficiency:** Extremely fast execution on web-scale data
3. **Scalability:** Linear time complexity for billion+ document processing

Cost Constraint: Filtering compute cost must be significantly less than training cost, otherwise direct training is preferable.

Algorithm Insight

Universal Filtering Recipe:

1. Estimate model/scoring function from target and raw data
2. Apply scoring function to all raw data examples
3. Retain examples exceeding threshold score

This pattern underlies all filtering approaches despite different mathematical foundations.

3 N-gram Models for Filtering

3.1 Kneser-Ney Smoothing Foundation

Historical Context: Inherited from statistical language processing era, implemented in KenLM (open-source, originally for machine translation).

Maximum Likelihood Estimation:

$$P(w_n|w_1, \ldots, w_{n-1}) = \frac{\text{count}(w_1, \ldots, w_n)}{\text{count}(w_1, \ldots, w_{n-1})}$$

Sparse Count Problem: Many reasonable n-grams have zero counts, especially as n increases, leading to curse of dimensionality.

Kneser-Ney Solution: Handle unseen n-grams by backing off to lower-order models:

- Remove one word from context when insufficient data
- Interpolate or back off to $(n-1)$-gram estimates
- Sophisticated smoothing preserves probability mass

3.2 Practical Implementation

Model Training: Simple counting and normalization - no neural network complexity.

Scoring Process:

1. Compute log probability under target n-gram model
2. Convert to perplexity: $\text{PPL} = \exp(-\frac{1}{N}\sum \log P(w_i))$
3. Lower perplexity indicates higher similarity to target

Example Results:

- Wikipedia text: Perplexity = 187 (reasonable baseline)
- CS336 website text: Higher perplexity (domain mismatch)
- Random gibberish: Very high perplexity
- Some edge cases: Unexpected low perplexity due to model limitations

3.3 CCNet Implementation

Processing Unit: Paragraphs of text as atomic filtering units.

Selection Strategy: Sort paragraphs by increasing perplexity, retain top 1/3.

Historical Success: Used to create first Llama dataset, demonstrating effectiveness despite heuristic nature.

Performance Characteristics:

- **Speed:** Very fast - linear time counting operations
- **Accuracy:** Crude but sufficient for initial filtering
- **Scalability:** Handles web-scale data efficiently

4 FastText Linear Classification

4.1 Motivation and Architecture

Historical Context: 2016 Facebook research showing linear classifiers competitive with complex neural models while maintaining superior speed.

Bag-of-Words Baseline: Traditional approach requires $V \times K$ parameter matrix (vocabulary size × classes), leading to sparsity issues.

FastText Innovation: Dimensionality reduction through hidden layer:

$$\text{Vocab} \rightarrow \text{Hidden}(H) \rightarrow \text{Classes}(K)$$

where $H \ll V$, reducing parameters from $V \times K$ to $(V \times H) + (H \times K)$.

Key Insight: No non-linearities in forward pass - purely linear classification with matrix factorization interpretation.

4.2 N-gram Extension and Hashing

N-gram Challenge: Vocabulary size becomes unbounded when including bigrams, trigrams, etc.

Hashing Solution: Map n-grams to fixed number of bins (e.g., 10 million):

- Hash function: "the cat" $>$ bin 2, "cat in" $>$ bin 1
- Collision handling: Weight represents average of colliding n-grams
- Loss minimization accounts for collisions automatically

Binary Classification: Most filtering applications use $K = 2$ (good/bad document classification).

4.3 Implementation Optimizations

Parallelization: Optimized implementation with asynchronous SGD.

Trade-off Consideration: Could use more sophisticated models (BERT, Llama) but computational cost may exceed training budget.

Efficiency Constraint: For 1/100 filtering rate, classifier compute must be $<$ 1/100 of full data processing cost.

5 Importance Sampling Approach

5.1 Theoretical Foundation

Problem Setup: Have proposal distribution Q (can sample from), want target distribution P (cannot directly sample from).

Importance Resampling Process:

1. Sample from Q: $x_1, x_2, \ldots, x_n \sim Q$
2. Compute importance weights: $w_i = \frac{P(x_i)}{Q(x_i)}$
3. Normalize weights: $\tilde{w}_i = \frac{w_i}{\sum_j w_j}$

4. Resample according to $\tilde{w}_i$

Mathematical Foundation

Importance Sampling Mathematics: For target distribution P and proposal Q:

$$\mathbb{E}_P[f(X)] = \mathbb{E}_Q\left[f(X) \cdot \frac{P(X)}{Q(X)}\right]$$

This enables sampling from P using only samples from Q.

5.2 Data Selection Application

Dataset Adaptation: Treat small target dataset D_p and large raw dataset D_q as representing distributions P and Q.

Small Dataset Challenge: D_p too small for reliable distribution estimation.

Hash N-gram Solution:

1. Hash all unigrams/n-grams to fixed number of bins
2. Estimate probability of each hash bin in both datasets
3. Handle hash collisions (e.g., "the" and "hat" $>$ same bin)
4. Compute importance ratio for new documents

Probability Evaluation: For new text, hash all tokens and multiply individual bin probabilities.

5.3 Advantages and Limitations

Theoretical Appeal: More principled than binary classification - attempts to match distributions rather than just distinguish them.

Diversity Benefits: Better for maintaining dataset diversity since it models full distribution rather than binary decision boundary.

Empirical Results: Modest improvements over fastText on GLUE benchmarks with BERT-style models.

Computational Similarity: Comparable speed to fastText while providing distributional guarantees.

6 Filtering Applications

6.1 Language Identification

Motivation: Focus computational resources on specific languages rather than diffusing across all languages in multilingual data.

Compute Trade-off Example: BLOOM (2022) trained on only 30/100 English, potentially limiting English performance to improve other languages.

FastText Language ID: Off-the-shelf classifier supporting many languages, trained on Wikipedia, translation sites, and Southeast European news.

DOLMA Implementation: Keep pages with English probability > 0.5.

Practical Challenges:

- Short sentences: Less reliable due to limited information
- Low-resource languages: Insufficient training data
- Similar languages: Confusion between closely related languages
- Code-switching: Mixed language content classification difficulty
- Dialects: May be classified as non-English despite mutual intelligibility

6.2 Mathematical Content Filtering (OpenWebMath)

Case Study: Large corpus of mathematical text curation demonstrating domain-specific filtering effectiveness.

Processing Pipeline:

1. Rule-based initial filtering for mathematical indicators
2. KenLM training on ProofPile (mathematical proofs dataset)
3. Perplexity threshold filtering
4. FastText classifier for mathematical writing identification
5. Adaptive thresholds: Lower for rule-identified math, higher for others

Results: 15 billion tokens of mathematical content, models outperform baselines trained on 20× more general data.

Domain Specialization Value: Demonstrates data efficiency gains from targeted curation versus general web training.

6.3 Quality Filtering Evolution

Historical Resistance: Early papers avoided model-based filtering to prevent bias, protect marginalized content.

Current Consensus: Model-based approaches significantly outperform rule-based methods on benchmarks.

GPT-3 Approach:

- Positive samples: High-quality sources (non-Common Crawl)
- Negative samples: General Common Crawl
- Linear classifier with stochastic retention based on scores

Llama 1 Innovation: Use pages referenced by Wikipedia (not Wikipedia itself) as positive examples, capturing high-quality content that doesn't resemble encyclopedia articles.

Phi-1 Methodology:

- Target: "textbook-like" high-quality educational content
- GPT-4 evaluation: 100K Python code examples for educational value
- Random forest classifier using pre-trained embeddings
- Synthetic target generation: Use LLM to define quality criteria

6.4 Toxicity Filtering

Jigsaw Toxic Comments Dataset: Wikipedia talk page comments annotated for toxic, severe toxic, obscene, threat, insult, identity hate.

DOLMA Implementation: Two fastText classifiers for hate detection and NSFW content identification.

Classification Challenges:

- Context dependency: Harmful content varies by situation
- False positives: Legitimate content misclassified as toxic
- Cultural sensitivity: Different standards across communities
- Adversarial resistance: Intentional circumvention attempts

7 Deduplication Fundamentals

7.1 Duplicate Types and Sources

Exact Duplicates: Identical content from web mirroring, common with sites like Project Gutenberg appearing on multiple URLs.

Near Duplicates: Similar content with minor variations:

- Terms of service and licenses (MIT license variations)
- Copy-paste errors (missing punctuation)
- Template-generated content (location/entity substitutions)
- Paraphrasing ("best actor" -> "most impactful character")

Extreme Case: C4 contains single sentence appearing 61,000 times due to Amazon product description propagation.

Training Impact: Not quality issue but efficiency problem - 61,000 epochs over same content provides no additional learning value.

7.2 Benefits of Deduplication

Training Efficiency: Reduce token count without information loss, enabling longer training on diverse content.

Memorization Mitigation: Reduce risk of models memorizing and regurgitating training content, important for copyright and privacy.

Complementary to Quality Filtering: Quality filtering removes bad content entirely; deduplication reduces good content repetition.

7.3 Design Space Dimensions

Unit Selection: What constitutes an "item" for deduplication?

- Sentence-level: Fine-grained but computationally expensive
- Paragraph-level: Balance between granularity and efficiency

- Document-level: Coarse but efficient processing

Matching Criteria: How to define "duplicate"?

- Exact match: Identical content only
- Threshold-based: Fraction of common sub-items exceeds threshold
- Semantic similarity: Embedding-based similarity measures

Action Strategy: What to do with duplicates?

- Remove all instances: Aggressive deduplication
- Remove all but one: Preserve single copy
- Weight reduction: Decrease sampling probability

8 Hash Functions and Exact Deduplication

8.1 Hash Function Properties

Definition: Function mapping large items (documents, sentences) to small hash values (integers, strings).

Collision Behavior: Two distinct items mapping to same hash value.

- Cryptographic hashes: Collision-resistant but slow
- Non-cryptographic hashes: Fast but collision-prone
- Data processing choice: Fast hashes (e.g., MurmurHash)

Speed vs. Security Trade-off: Data processing prioritizes speed over cryptographic security.

8.2 Exact Deduplication Algorithm

MapReduce Implementation:

1. Compute hash for each item
2. Group items by hash value
3. Keep one item from each group
4. Handle hash collisions conservatively

Example: "hello" appears twice > keep one copy; "Hello!" is distinct item under exact matching.

Advantages:

- High precision: Never removes needed content
- Simple implementation: Straightforward MapReduce pattern
- Parallel scalability: Easy to distribute across machines

Limitations:

- No near-duplicate detection
- Sensitive to minor variations
- May fragment documents when operating on spans

8.3 C4 Implementation Details

Deduplication Unit: Three-sentence spans rather than full documents.

Surgical Removal: Extract duplicate spans from documents, potentially creating incoherent remaining text.

Pragmatic Approach: "Who cares about coherence" philosophy - prioritize efficiency over perfect document structure.

9 Bloom Filters for Approximate Membership

9.1 Bloom Filter Fundamentals

Properties:

- Memory-efficient set representation
- Supports insertion and membership queries
- No deletion capability in basic version
- False positives possible, false negatives impossible

Query Guarantees:

- Returns "No" > Definitely not in set
- Returns "Yes" > Probably in set (false positive possible)

Implementation Challenge

Bloom Filter Trade-offs: Hyperparameters control memory usage, computation cost, and false positive rate. Optimal configuration depends on dataset size, available memory, and acceptable error rates. Poor parameter choices can lead to either excessive memory usage or unacceptably high false positive rates.

9.2 Single Hash Function Construction

Basic Algorithm:

1. Initialize bit array of size m
2. For each item: hash item, set corresponding bit to 1
3. For queries: hash item, check if corresponding bit is 1

Example: With 8 bins, inserting "the", "cat", "in", "the", "hat":

- "the" $>$ hash(2) $>$ set bit[2] $=$ 1
- "cat" $>$ hash(7) $>$ set bit[7] $=$ 1
- Continue for all items

False Positive Analysis: Test items not in original set - some will hash to already-set bits, creating false positives.

Error Rate: With single hash function, error probability $\approx \frac{1}{m}$ (number of bins).

9.3 Multiple Hash Functions Improvement

Enhanced Construction:

1. Use k different hash functions (different seeds)
2. For insertion: set bits at all k hash locations
3. For queries: check all k locations, return positive only if all are set

Error Reduction: Multiple hash functions dramatically reduce false positive rate while maintaining efficiency.

Optimal Parameter Selection: Balance between memory usage (m), computation cost (k), and error rate.

9.4 Mathematical Analysis

Probability Calculations: For m bins, k hash functions, n items:

Single Item, Single Hash: Collision probability $= \frac{1}{m}$

Single Item, k Hashes: Miss probability $= \left(1 - \frac{1}{m}\right)^k$

n Items, k Hashes: Miss probability $= \left(1 - \frac{1}{m}\right)^{kn}$

False Positive Rate:

$$F = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k$$

Optimal Configuration: $k^* = \frac{m \ln 2}{n}$, yielding $F^* = 0.5^{k^*}$.

DOLMA Parameters: False positive rate 10^{-15}, paragraph-level deduplication.

10 MinHash and Locality Sensitive Hashing

10.1 Jaccard Similarity Foundation

Definition: For sets A and B:

$$\text{Jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Example: $A = \{1, 2, 3, 4\}$, $B = \{1, 2, 3, 5\}$

$$\text{Jaccard}(A, B) = \frac{|\{1, 2, 3\}|}{|\{1, 2, 3, 4, 5\}|} = \frac{3}{5} = 0.6$$

Near-Duplicate Definition: Two documents are near-duplicates if Jaccard similarity exceeds threshold (typically 0.99 for high precision).

Computational Challenge: Jaccard similarity is pairwise operation, but need linear-time algorithm for web-scale processing.

10.2 MinHash Algorithm

Key Property: Hash function where collision probability equals Jaccard similarity:

$$P(\text{MinHash}(A) = \text{MinHash}(B)) = \text{Jaccard}(A, B)$$

Algorithm:

1. Hash all elements in set
2. Return minimum hash value

Intuitive Explanation: Random permutation interpretation - element appearing "first" determines MinHash value.

Mathematical Foundation

MinHash Collision Probability: Consider sets A and B with union $A \cup B$. Under random permutation, MinHash collision occurs when element from $A \cap B$ appears first. Probability equals $\frac{|A \cap B|}{|A \cup B|} = \text{Jaccard}(A, B)$.

10.3 Locality Sensitive Hashing (LSH)

Problem: Single MinHash provides probabilistic similarity but need deterministic threshold-based classification.

Goal: Transform collision probability to approximate step function around desired threshold.

Band Construction: Divide n hash functions into b bands of r hash functions each ($n = b \times r$).

Collision Criterion: Two items collide if they agree on all hash functions in at least one band.

Probability Analysis: For Jaccard similarity s:

- Band match probability: s^r
- No band matches probability: $(1 - s^r)^b$
- At least one band matches: $1 - (1 - s^r)^b$

10.4 Parameter Tuning for Threshold Sharpening

Sigmoid Curve Shaping: LSH parameters control collision probability vs. similarity curve shape.

Effect of r (band size):

- Increase r: Sharper threshold, shifted right (harder to match)
- Decrease r: Smoother curve, shifted left (easier to match)

Effect of b (number of bands):

- Increase b: Shift curve left (easier to match at high similarity)

- Decrease b: Shift curve right (harder to match overall)

Optimization Strategy: Adjust b and r to create sharp sigmoid around desired threshold.

Threshold Convergence: At optimal threshold, collision probability converges to $1-\frac{1}{e} \approx 0.63$.

10.5 Production Example

Parameters: $b = 20$ bands, $r = 450$ hash functions per band (total 9,000 hash functions).

Threshold Calculation: $t = (1/b)^{1/r} = (1/20)^{1/450} \approx 0.99$

Interpretation: Allow only 1 different word per 100 words - very strict near-duplicate detection.

Performance: At threshold, collision probability ≈ 0.64, with rapid dropoff below threshold and approach to 1.0 above threshold.

11 Advanced Deduplication Considerations

11.1 Semantic Deduplication

Embedding-Based Approaches: Use document embeddings instead of token-level similarity for paraphrase detection.

Nearest Neighbor Framework: LSH framework generalizes to any embedding space with appropriate distance metrics.

Computational Cost: Embedding generation significantly more expensive than token-based methods.

Precision Trade-offs: More semantic awareness but risk of over-aggressive removal of valuable content variations.

11.2 Quality-Aware Deduplication

Duplicate Value Consideration: High-quality content may merit multiple copies or increased sampling weight.

Count-Based Weighting: Use occurrence frequency as quality signal - rare content removed, common content preserved but down-weighted.

Logarithmic Scaling: Apply sublinear functions (square root, logarithm) to occurrence counts for balanced representation.

Mid-Training Context: Multiple epochs over high-quality data are desirable, contradicting aggressive deduplication.

12 Implementation Trade-offs and Considerations

12.1 Computational Complexity

Exact Deduplication: $O(n)$ time complexity with efficient hashing, $O(n)$ space for hash table.

MinHash LSH: $O(n \cdot k)$ time for k hash functions, $O(n \cdot k)$ space for signatures.

Bloom Filters: $O(n \cdot k)$ time, $O(m)$ space for m bins, very memory efficient.

Parallelization: All methods amenable to MapReduce-style distributed processing.

12.2 Parameter Selection Guidelines

Bloom Filter Configuration:

- Target false positive rate based on downstream impact
- Memory constraints determine bin count
- Hash function count optimized for error minimization

LSH Parameter Selection:

- Desired similarity threshold determines b and r relationship
- Computational budget constrains total hash function count
- Storage requirements scale with signature size

Quality vs. Speed Trade-offs: More sophisticated methods provide better accuracy but require significantly more computation.