Naive Text Classification

Bag-of-words model and Naive Bayes classifier

Zbigniew Dendzik

March 2026

Lecture Plan

- Introduction to text classification
- Bag-of-words model
- TrainingDocument and WordVector classes
- Training set (TrainingSet)
- Naive Bayes classifier
- Testing and accuracy evaluation
- Extensions and comparison with LLM

Introduction to Text Classification

Goal: Automatic assignment of text documents to categories.

Application examples:

- Spam filters (spam / not spam)
- Email classification (work / personal / advertising)
- Sentiment analysis (positive / negative)
- Article categorization (sports / politics / technology)

Approaches:

- **Simple:** bag-of-words + Naive Bayes
- **Advanced:** embeddings, neural networks, LLM

Bag-of-words Model

Bag-of-words – document representation as a vector of word counts.

Main idea:

- We ignore word order
- Only frequency of occurrence matters
- Document = bag of words

Example:

- Text: "The football match ended in a draw."
- Bag-of-words: {the: 1, football: 1, match: 1, ended: 1, in: 1, a: 1, draw: 1}

Pros and cons:

+ Simple implementation
+ Fast execution
– No information about context and order

System Architecture

System components:

1. **Document representation**
   - `TrainingDocument` – document with class label
   - `WordVector` – word frequency vector
2. **Training set**
   - `TrainingSet` – collection of training documents
   - Class labels: "sports", "politics", "technology"
3. **Classifier**
   - `NaiveBayes` – statistical model
   - Training: counting probability distributions
   - Prediction: selecting most probable class

TrainingDocument Class

```
class TrainingDocument
{
    private String text;
    private String label; // "sports", "politics", etc.

    public TrainingDocument(String text, String label)
    {
        this.text = text;
        this.label = label;
    }

    public String getText()
    {
        return text;
    }

    public String getLabel()
    {
```

TrainingDocument Class – Explanation

Private fields:

- `text` – document content
- `label` – class label (category)

Public methods:

- `getText()` – returns document text
- `getLabel()` – returns class label

Application:

- Storing training examples
- Linking text with corresponding class
- Basic building block of training set

WordVector Class – Structure

```
import java.util.*;

class WordVector
{
    private Map<String, Integer> counts;

    public WordVector()
    {
        counts = new HashMap<String, Integer>();
    }

    public void fromText(String text)
    {
        // normalization and word counting
    }

    public int freq(String word)
    {
```

WordVector Class – fromText()

```
public void fromText(String text)
{
    counts.clear();
    String txt = text.toLowerCase();
    txt = txt.replaceAll("[^a-z0-9 ]", " ");
    String[] words = txt.split("\\s+");

    for(String w : words)
    {
        if(w.length() == 0)
            continue;
        Integer c = counts.get(w);
        if(c == null) c = 0;
        counts.put(w, c + 1);
    }
}
```

Text Normalization

Text processing steps:

1. **Convert to lowercase**
   - `toLowerCase()` – "Football" $\rightarrow$ "football"
2. **Remove special characters**
   - `replaceAll("[`$^{a} - z0 - 9$`]","")` – – *keepsonlyletters*, *digitsandspaces*
   - `Commas, periods, exclamation marks → spaces`
3. **Split into words**
   - `split("`
     `textbackslash`
     `textbackslash s+")` – splits by whitespace
4. **Count occurrences**
   - Each word $\rightarrow$ increment counter in map

Methods freq() and words()

```
public int freq(String word)
{
    Integer c = counts.get(word.toLowerCase());
    if(c == null) return 0;
    return c;
}

public Set<String> words()
{
    return counts.keySet();
}
```

Application:

- `freq(word)` – frequency of specific word
- `words()` – set of all unique words
- Used during classification and training

TrainingSet Class

```
import java.util.*;

class TrainingSet
{
    private List<TrainingDocument> docs;

    public TrainingSet()
    {
        docs = new ArrayList<TrainingDocument>();
    }

    public void add(TrainingDocument d)
    {
        docs.add(d);
    }

    public List<TrainingDocument> getDocuments()
    {
```

TrainingSet – labels() Method

```
public Set<String> labels()
{
    Set<String> s = new HashSet<String>();
    for(TrainingDocument d : docs)
        s.add(d.getLabel());
    return s;
}
```

Functionalities:

- `add()` – adding training documents
- `getDocuments()` – retrieving all documents
- `labels()` – set of unique class labels

Application:

- Storing training examples
- Iterating through documents during learning

Sample Training Set (1)

```
public static TrainingSet buildSample()
{
    TrainingSet set = new TrainingSet();

    set.add(new TrainingDocument(
        "The football match ended in a draw.",
        "sports"));

    set.add(new TrainingDocument(
        "The new coach of the national team was chosen.",
        "sports"));

    set.add(new TrainingDocument(
        "The parliament debate was about a new law.",
        "politics"));
```

Sample Training Set (2)

```
    set.add(new TrainingDocument(
        "The president gave a speech about the economy.",
        "politics"));

    set.add(new TrainingDocument(
        "The new smartphone offers a fast processor and 5G."
            ,
        "technology"));

    set.add(new TrainingDocument(
        "The company announced a new laptop.",
        "technology"));

    return set;
}
```

Naive Bayes – Theoretical Foundations

Naive Bayes classifier assumptions:

1. **Each class has word probability distribution**
   - P(word — class) – probability of word in given class
2. **Naive independence assumption**
   - Words in document are conditionally independent
   - P(words — class) = P(word_1 — class) · P(word_2 — class) · ...
3. **Bayes' formula**
   - P(class — document) P(class) · P(document — class)
   - We choose class with maximum probability

Naive Bayes – Implementation

Learning process:

1. **For each class:**
   - Count number of documents: P(class)
   - Count occurrences of each word
   - Compute P(word — class)
2. **Laplace smoothing:**
   - Add 1 to word counters (avoid zero probability)
   - P(word — class) = (word_count + 1) / (word_sum + vocabulary_size)
3. **Logarithms:**
   - log P(class — document) = log P(class) + log P(word — class)
   - Avoid numerical problems (very small numbers)

NaiveBayes Class – Fields

```
import java.util.*;

class NaiveBayes
{
    private TrainingSet set;
    private Map<String, Integer> docsPerLabel;
    private Map<String, Map<String, Integer>>
        wordCountsPerLabel;
    private Map<String, Integer> totalWordsPerLabel;
    private Set<String> vocabulary;

    public NaiveBayes(TrainingSet set)
    {
        this.set = set;
        docsPerLabel = new HashMap<String, Integer>();
        wordCountsPerLabel = new HashMap<String, Map<String,
            Integer>>();
        totalWordsPerLabel = new HashMap<String, Integer>();
```

Data Structures in NaiveBayes

Class fields:

docsPerLabel

- Map: class → number of documents in that class
- Example: {"sports": 2, "politics": 2, "technology": 2}

wordCountsPerLabel

- Map: class → (word → occurrence count)
- Example: {"sports": {"football": 5, "match": 3}}

totalWordsPerLabel

- Map: class → total number of words in that class

vocabulary

- Set of all unique words in entire training set

train() Method – Part 1

```
public void train()
{
    for(TrainingDocument d : set.getDocuments())
    {
        String label = d.getLabel();
        Integer nd = docsPerLabel.get(label);
        if(nd == null) nd = 0;
        docsPerLabel.put(label, nd + 1);

        Map<String, Integer> map =
            wordCountsPerLabel.get(label);
        if(map == null)
        {
            map = new HashMap<String, Integer>();
            wordCountsPerLabel.put(label, map);
        }
```

train() Method – Part 2

```
        WordVector v = new WordVector();
        v.fromText(d.getText());

        for(String w : v.words())
        {
            vocabulary.add(w);
            int c = v.freq(w);
            Integer prev = map.get(w);
            if(prev == null) prev = 0;
            map.put(w, prev + c);

            Integer sum = totalWordsPerLabel.get(label);
            if(sum == null) sum = 0;
            totalWordsPerLabel.put(label, sum + c);
        }
    }
}
```

Training Process – Step by Step

For each training document:

1. **Update document counter in class**
   - `docsPerLabel[label] += 1`
2. **Process document text**
   - Create WordVector from text
   - Obtain word frequency vector
3. **For each word in document:**
   - Add to vocabulary
   - Update word counter in given class
   - Update total word count in class

Result: Complete statistics for classification

classify() Method – Initialization

```
public String classify(String text)
{
    WordVector v = new WordVector();
    v.fromText(text);

    double bestLogP = Double.NEGATIVE_INFINITY;
    String bestLabel = null;

    int N = set.getDocuments().size();
    int V = vocabulary.size();

    for(String label : set.labels())
    {
        // calculations for each class...
    }

    return bestLabel;
}
```

classify() Method – Calculations

```
for(String label : set.labels())
{
    int nd = docsPerLabel.get(label);
    double logP = Math.log((double)nd / (double)N);

    Map<String, Integer> map = wordCountsPerLabel.get(label)
        ;
    int sumLabel = totalWordsPerLabel.get(label);

    for(String w : v.words())
    {
        int tf = v.freq(w);
        if(tf == 0) continue;

        Integer count = map.get(w);
        if(count == null) count = 0;

        double pWord = (count + 1.0) / (sumLabel + V);
```

classify() Method – Class Selection

```
    if(logP > bestLogP)
    {
        bestLogP = logP;
        bestLabel = label;
    }
}

return bestLabel;
```

Classification process:

- For each class we compute log probability
- Consider P(class) and P(words — class)
- Choose class with highest probability

Laplace Smoothing

Problem: Unknown words (not occurring in training)

Without smoothing:

- P(unknown_word — class) = 0
- Entire document probability = 0
- Classification impossible

With Laplace smoothing:

- P(word — class) = (count + 1) / (sum + V)
- Add 1 to each word counter
- Add V (vocabulary size) to denominator
- Minimum probability: 1/V

Result: Stable classifier without zeros

Test Program – TestBayes

```
import java.util.*;

public class TestBayes
{
    public static void main(String[] args)
    {
        TrainingSet set = SampleData.buildSample();

        NaiveBayes nb = new NaiveBayes(set);
        nb.train();

        Scanner in = new Scanner(System.in);
        while(true)
        {
            System.out.println("Enter text (empty=quit):");
            String text = in.nextLine();
            if(text.trim().isEmpty())
                break;
```

Example Usage

Scenario:

1. **Create training set**
   - 2 "sports" documents
   - 2 "politics" documents
   - 2 "technology" documents
2. **Train model**
   - `nb.train()` – compute statistics
3. **Testing**
   - Input: "The team won the championship"
   - Prediction: "sports"
   - Input: "The minister announced a new policy"
   - Prediction: "politics"

Accuracy Evaluation

Accuracy metric:

- Accuracy = (number of correct predictions) / (total number of tests)

Example:

- 10 tests
- 8 correct predictions
- Accuracy = $8/10 = 0.8 = 80\%$

Common error causes:

- Small training set
- Ambiguous words (e.g. "bank" – river or institution)
- Lack of context in bag-of-words
- Insufficient class differentiation

Extensions – Stopwords

Stopwords – frequently occurring words without classification significance.

Examples:

- English: "the", "a", "an", "is", "are", "and", "or"
- Polish: "i", "w", "na", "z", "do", "o"

Implementation:

- Store stopwords list in `Set<String>`
- In `fromText()` skip these words
- Alternatively: filter during classification

Effect:

- Reduced noise in representation
- Faster processing
- Often better accuracy

Extensions – Filtering Rare Words

Problem: Words occurring only once (hapax legomena)

Impact on model:

- Increase vocabulary size without benefit
- Cause overfitting
- Rarely appear in test data

Solution:

- After training: remove words with frequency $= 1$
- Set minimum frequency threshold (e.g. 2 or 3)
- Rebuild vocabulary and counters

Result:

- Smaller, more representative vocabulary
- Better generalization to new documents

Bag-of-words vs LLM

Aspect	Bag-of-words	LLM
Word order	Ignored	Considered
Context	None	Full understanding
Meaning	Superficial	Semantic
Complexity	Very low	Very high
Speed	Lightning fast	Slower
Training data	Dozens	Billions
Application	Simple filters	Complex NLP

Example:

- "This movie is not bad" vs "This movie is bad"
- Bag-of-words: almost identical
- LLM: understands negation, opposite meaning

When is Bag-of-words Useful?

Good applications:

1. **Very simple filters**
   - Basic spam detection
   - Ticket routing to categories
2. **Rapid prototyping**
   - Testing idea before LLM deployment
   - Baseline for comparison
3. **Limited resources**
   - No GPU
   - Real-time requirements
   - Embedded systems
4. **Small training set**
   - Dozens of examples
   - LLM requires fine-tuning on large data

Modern Approaches

Evolution of text classification methods:

1. **Vector embeddings (2013-2018)**
   - Word2Vec, GloVe
   - Word representation as vectors
   - Semantic similarity
2. **Contextual models (2018-2020)**
   - BERT, RoBERTa, DistilBERT
   - Context understanding
   - Transfer learning
3. **Large language models (2020+)**
   - GPT-3, GPT-4, Claude
   - Zero-shot and few-shot classification
   - Natural language instructions

Summary

- **Bag-of-words** – simple text representation as frequency vector
- **Naive Bayes** – fast probabilistic classifier
- **Laplace smoothing** – handling unknown words
- **Extensions** – stopwords, filtering rare words
- **Limitations** – no context and meaning
- **LLM** – modern alternative with deep understanding
- **Applications** – simple filters, prototypes, limited resources