Sentiment Classification using Naive Bayes Classifier

Koç University COMP541 Self-Assessment Programming Exercise*

October 12, 2025

Deadline: October 19, 2025, 23:59 PM.

1 Introduction

In this project you will implement a sentiment analysis system based on the "Large Movie Review Dataset". The input to your system is a piece of text, and the output is a binary classification of the text into **positive** or **negative**.

Examples:

Input:	But perhaps you have to have grown up in the 80's to truly appreciate this movie. If you love the early 80's this is definitely a must see. Also, one of the best soundtracks ever!
Output:	Positive
Input:	Widow hires a psychopath as a handyman. Sloppy film noir thriller which doesn't make much of its tension promising set-up.
Output:	Negative

The dataset contains 50,000 reviews split evenly into 25k training and 25k test samples. The label distribution is balanced (25k positive, 25k negative). You should train on the training data and report your accuracy on the test data.

Please start by downloading and reviewing the dataset from:

https://ai.stanford.edu/~amaas/data/sentiment

2 Preprocessing

You can explore the following ideas for preprocessing; they are all optional, and not all may be helpful. Below is a recommended **baseline pipeline** to ensure consistency and reproducibility:

- **Tokenize** each review using simple whitespace and punctuation splitting (e.g., `split()` or `re.findall()`).
- **Lowercase** all tokens.
- **Remove punctuation** or map it to separate tokens if desired.
- **Build a vocabulary** using only the training data.
- **Replace unseen or rare tokens** (below a frequency threshold k) with a special `<UNK>` symbol.
- **Optionally truncate or pad** documents to a fixed length for convenience.

These steps form a minimal yet strong baseline that can achieve $\geq 80\%$ accuracy when combined with proper smoothing.

You are encouraged to experiment further. For example:

- Try **stop-word removal** or **stemming/lemmatization** and compare results.
- Explore n**-gram features** (bigrams or trigrams) to capture limited word dependencies.
- Visualize frequent words for each class to gain intuition about model behavior.

3 Model

For prediction, you will implement a **Multinomial Naive Bayes Classifier**. This classifier assumes the following generative process: to generate each (x, y) pair where $y \in Y$ is a class and $x = [x_1, x_2, \ldots, x_n]$ is a document with features (e.g., words) $x_i \in X$:

- First pick $y \in Y$ with probability q_y (categorical distribution).
- Then pick each word $x_i \in X$ with probability $q_{x_i|y}$ (more categorical distributions).

The joint probability of a document and its label is:

$$P(x, y) = q_y \prod_i q_{x_i|y}.$$

The maximum likelihood estimates for the Naive Bayes parameters are:

$$\hat{q}_y = \frac{\text{\# documents with class } y}{\text{\# documents}}, \qquad \hat{q}_{x_i|y} = \frac{\text{\# of token } x_i \text{ in class } y + \alpha}{\text{\# tokens in class } y + \alpha|V|},$$

where $\alpha > 0$ is a smoothing constant (e.g., 1 for Laplace smoothing).

Implementation notes. To avoid numerical underflow, compute and store **log-probabilities** $\log P(y)$ and $\log P(x_i|y)$, and predict according to

$$\log P(y|x) \propto \log P(y) + \sum_i \log P(x_i|y).$$

The predicted label is the one with the highest log-probability.

Common issues and suggested remedies.

- Zero probabilities $\rightarrow$ add-one (Laplace) or Dirichlet smoothing.
- Independence assumptions too strong $\rightarrow$ try n-gram features.
- Word "burstiness" (the tendency of words to repeat within a document) $\rightarrow$ consider the Dirichlet Compound Multinomial (DCM) model (Murphy 2012, Sec. 3.5.5; Madsen et al., 2005: ICML 2005 PDF).

Reference: Kevin P. Murphy, *Machine Learning: A Probabilistic Perspective*, available at https://www.cs.ubc.ca/~murphyk

4 Requirements

In order to complete this assignment, you **MUST** fulfill all of the following requirements:

- You **MUST** implement a Naive Bayes classifier: you cannot select another model.
- You **MUST** implement this assignment using Python.
- For **ONLY** the preprocessing part, you are allowed to use third-party packages that may be suitable for the task. However, if you use any functions from such libraries, you **HAVE TO** provide a comment for each function that explains what the function does and why you used it in detail.
- For other parts of the assignment, you **CANNOT** use third-party packages except NumPy.
- You **CANNOT** use the preprocessed data (`imdb.vocab`, `labeledBow.feat`, `unsupBow.feat`, etc.).

- Your implementation **MUST** run in less than 1 minute on a regular workstation.
- You **MUST** achieve a minimum of 80% accuracy on the test set.
- You **MUST NOT** use hard-coded paths for the data directory. You can achieve this in two different ways:
  - (i) Define a `DATA_DIR` variable at the beginning of your code, or
  - (ii) Place the data in the same directory as your code.
- Your code **MUST** run without any errors.
- You **MUST** upload a single file (`.py` or `.ipynb`) with comments explaining your code. We encourage using a .ipynb file for the task.

5 Extensions / Exploration (Optional)

This section is optional and intended for students who want to explore beyond the required task.

- **Feature Engineering:** extend your model with bigram or trigram features.
- **Smoothing Techniques:** compare Laplace and Dirichlet smoothing empirically.
- **Visualization:** list or plot the most informative words for each class.
- **Theoretical Comparison:** derive $P(y|x)$ and contrast Naive Bayes with Logistic Regression conceptually.
- **Efficiency:** try sparse data structures to improve speed and memory usage.

Warning

If you think this assignment is too difficult for you, or you spend more than 10 hours completing it, please consider taking this course in the next academic year.

---

[*] *Initially prepared by Deniz Yuret. This updated version adds baseline preprocessing, detailed model guidance, and an optional exploration section (2025 revision).*