

Email SPAM Classifier

Aayush Bajaj

July 10, 2026

Contents

1	bayes' rule as a classifier	1
2	the naive assumption	2
2.1	what the assumption buys: the parameter count collapses	2
3	the variants	3
4	smoothing: why zero counts are fatal	3
5	log-space implementation	4
6	a spam filter from scratch	4
7	naive bayes is a linear classifier	5
8	the calibration caveat	6
9	when naive bayes wins	6
10	see also	7

naive bayes is the classifier you get by taking bayes' rule seriously and probability theory not seriously at all. it assumes every feature is independent of every other feature given the class — an assumption that is false for essentially all real data — and yet it filters spam, routes support tickets and triages documents well enough that it has survived five decades of fancier competition. this page derives it, counts why the “naive” part is the whole point, builds a spam filter from scratch, and is honest about where it breaks (its probabilities, not its decisions).

*the older literature calls it
'idiot's bayes', which is unfair
to a model that keeps beating its
critics on small text corpora*

1 bayes' rule as a classifier

we want the most probable class for a feature vector $x = (x_1, \dots, x_d)$. bayes' rule rewrites the posterior in terms of things we can estimate (Wasserman, Larry, 2010):

$$p(y = c \mid x) = \frac{p(x \mid y = c) \pi_c}{\sum_{c'} p(x \mid y = c') \pi_{c'}}, \quad (1)$$

where $\pi_c = p(y = c)$ is the class prior. the denominator is the same for every class, so the **map decision rule** (maximum a posteriori) drops it:

$$\hat{y} = \arg \max_c \pi_c p(x | y = c) = \arg \max_c \left[\log \pi_c + \log p(x | y = c) \right]. \quad (2)$$

nothing naive so far — this is the bayes classifier, and with the true distributions it is optimal for 0–1 loss (Hastie, Trevor and Tibshirani, Robert and Friedman, Jerome, 2009). the problem is the likelihood: $p(x | y = c)$ is a joint distribution over all d features, and estimating a joint is exactly the thing small datasets cannot do.

2 the naive assumption

naive bayes assumes the features are **conditionally independent given the class**:

$$p(x_1, \dots, x_d | y = c) = \prod_{j=1}^d p(x_j | y = c). \quad (3)$$

graphically: the class is the lone parent of every feature, and there are no edges between features. knowing the class screens off everything one feature could tell you about another.

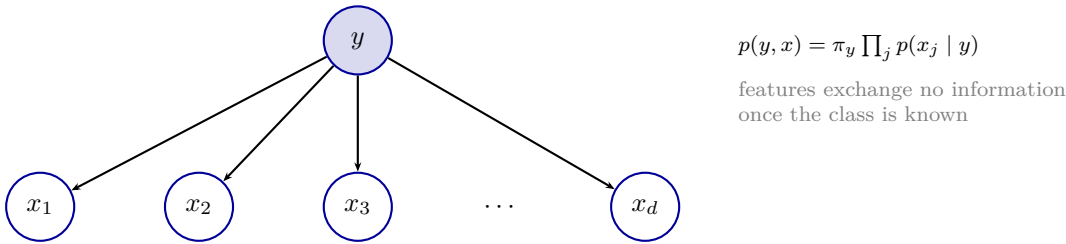


Figure 1: the naive bayes graphical model: the class y is the only parent of every feature. no feature-to-feature edges — that absence is the entire model.

2.1 what the assumption buys: the parameter count collapses

count parameters for d *binary* features and K classes:

- **full joint likelihood.** $p(x | y = c)$ is a distribution over 2^d configurations, so each class needs $2^d - 1$ free parameters: $K(2^d - 1)$ in total (plus $K - 1$ for the prior).
- **naive likelihood.** each feature needs one bernoulli parameter per class: Kd in total (plus the same $K - 1$).

for a modest spam filter with $d = 30$ binary word-presence features and $K = 2$:

$$\underbrace{2(2^{30} - 1) \approx 2.1 \times 10^9}_{\text{full joint}} \quad \text{vs} \quad \underbrace{2 \times 30 = 60}_{\text{naive}}. \quad (4)$$

exponential to linear in d . the full joint would need more training emails than have ever been sent; the naive model is estimable from a few hundred. that is the trade: naive bayes accepts (possibly large) bias in exchange for variance low enough to learn anything at all in high dimensions — the extreme low-variance end of the **bias–variance** dial.

a real vocabulary has tens of thousands of words, which makes the gap comically larger

3 the variants

the factorisation says nothing about *what* each per-feature likelihood $p(x_j \mid y)$ is. pick it to match the feature type (Hastie, Trevor and Tibshirani, Robert and Friedman, Jerome, 2009):

$$\text{bernoulli: } p(x \mid y = c) = \prod_{j=1}^d \theta_{cj}^{x_j} (1 - \theta_{cj})^{1-x_j}, \quad x_j \in \{0, 1\}$$

$$\text{multinomial: } p(x \mid y = c) = \frac{(\sum_j x_j)!}{\prod_j x_j!} \prod_{j=1}^d \theta_{cj}^{x_j}, \quad x_j \in \{0, 1, 2, \dots\}, \quad \sum_j \theta_{cj} = 1$$

$$\text{gaussian: } p(x \mid y = c) = \prod_{j=1}^d \frac{1}{\sqrt{2\pi\sigma_{cj}^2}} \exp\left(-\frac{(x_j - \mu_{cj})^2}{2\sigma_{cj}^2}\right), \quad x_j \in \mathbb{R}$$

- **bernoulli nb**: features are word *presence*. explicitly penalises the absence of class-typical words (the $(1 - x_j)$ factor), which helps on short documents.
- **multinomial nb**: features are word *counts*; the document is a bag of tokens drawn from a class-specific die. the workhorse for text.
- **gaussian nb**: continuous features get a per-class, per-feature normal. maximum likelihood fits μ_{cj}, σ_{cj}^2 by the class-conditional sample mean and variance — training is one pass of bookkeeping.

the multinomial coefficient does not depend on the class, so it cancels from every posterior comparison

mixing types is fine: the log-likelihood is a sum over features, so each feature can bring its own distribution.

4 smoothing: why zero counts are fatal

maximum likelihood for the multinomial model is just relative frequency:

$$\hat{\theta}_{cj} = \frac{N_{cj}}{\sum_{j'} N_{cj'}}, \quad (5)$$

where N_{cj} counts occurrences of word j in class- c training documents. now let one test email contain a vocabulary word never seen in spam. then $\hat{\theta}_{\text{spam},j} = 0$, and because the likelihood is a *product*, the entire spam posterior is annihilated by a single factor — one unseen word outvotes fifty incriminating ones, with infinite confidence. in log space it is a $-\infty$ no other term can repair.

lidstone smoothing adds a pseudo-count $\alpha > 0$ to every word–class pair:

$$\hat{\theta}_{cj} = \frac{N_{cj} + \alpha}{\sum_{j'} N_{cj'} + \alpha V}, \quad V = |\text{vocabulary}|, \quad (6)$$

with $\alpha = 1$ the classic **laplace smoothing**. this is not a hack: it is exactly the map estimate under a symmetric dirichlet prior $\text{Dir}(\alpha + 1)$ on θ_c — pretend you saw every word α times before looking at the data (Deisenroth, Marc Peter and Faisal, A. Aldo and Ong, Cheng Soon, 2020). small α trusts counts; large α flattens towards uniform. tune it on held-out data like any other regulariser.

5 log-space implementation

a 2,000-word email multiplies 2,000 numbers of magnitude 10^{-3} to 10^{-5} ; the product underflows double precision long before word 300. nobody computes the product — everyone computes the sum of logs:

$$\text{score}_c(x) = \log \pi_c + \sum_j x_j \log \hat{\theta}_{cj}, \quad (7)$$

- **decide:** $\hat{y} = \arg \max_c \text{score}_c(x)$ — no normalisation needed.
- **calibrate:** if you want actual posteriors, normalise with **log-sum-exp**, subtracting the max score first so the exponentials cannot overflow: $\log Z = m + \log \sum_c e^{\text{score}_c - m}$, $m = \max_c \text{score}_c$.
- **train:** everything is counting, one pass over the data, trivially parallel and trivially *online* — new labelled email arrives, increment counters, done.

6 a spam filter from scratch

seven emails, honest arithmetic, no libraries. spam contributes 10 tokens, ham 12, and the joint vocabulary has $V = 15$ words, so e.g. the smoothed estimate for “cash” (twice in spam, absent from ham) is $\hat{\theta}_{\text{spam}, \text{cash}} = \frac{2+1}{10+15} = 0.12$ and $\hat{\theta}_{\text{ham}, \text{cash}} = \frac{0+1}{12+15} \approx 0.037$ — smoothing keeps ham merely *sceptical* of “cash”, not infinitely so.

```
1 import math
2 from collections import Counter
3
4 train = [("win cash now", "spam"),
5          ("limited offer win prize", "spam"),
6          ("cash prize inside", "spam"),
7          ("meeting notes attached", "ham"),
8          ("lunch meeting tomorrow", "ham"),
9          ("offsite notes attached", "ham"),
10         ("agenda for tomorrow", "ham")]
11
12 # --- train: count everything once -----
13 docs = Counter(c for _, c in train) # class counts
14 words = {c: Counter() for c in docs} # word counts per
15     ↪ class
16 for text, c in train:
17     words[c].update(text.split())
18 vocab = sorted(set(w for c in words for w in words[c]))
19 V = len(vocab)
20
21 def log_likelihood(w, c, alpha=1.0): # laplace-smoothed
22     return math.log((words[c][w] + alpha) /
23                     (sum(words[c].values()) + alpha * V))
24
25 def classify(text):
26     x = [w for w in text.split() if w in vocab] # drop out-of-vocab
27     ↪ words
28     scores = {c: math.log(docs[c] / len(train)) +
29              sum(log_likelihood(w, c) for w in x) for c in docs}
30     z = max(scores.values()) # log-sum-exp, stably
31     logZ = z + math.log(sum(math.exp(s - z) for s in scores.values()))
32     return scores, {c: math.exp(s - logZ) for c, s in scores.items()}
```

```

32 print(f"vocab size V = {V}, priors: spam {docs['spam']/7}, ham
    ↪ {docs['ham']/7}")
33 print(f"theta('cash'|spam) = (2+1)/(10+{V}) =
    ↪ {math.exp(log_likelihood('cash','spam')):.4f}")
34 print(f"theta('cash'|ham) = (0+1)/(12+{V}) =
    ↪ {math.exp(log_likelihood('cash','ham')):.4f}")
35 for msg in ["win cash prize", "meeting agenda tomorrow", "win cash tomorrow"]:
36     scores, post = classify(msg)
37     print(f"'{msg}': log p(spam,x) = {scores['spam']:.3f}, "
38           f"log p(ham,x) = {scores['ham']:.3f}, p(spam|x) =
    ↪ {post['spam']:.4f}")

```

vocab size $V = 15$, priors: spam $3/7$, ham $4/7$
 $\theta(\text{'cash'}|\text{spam}) = (2+1)/(10+15) = 0.1200$
 $\theta(\text{'cash'}|\text{ham}) = (0+1)/(12+15) = 0.0370$
 'win cash prize': $\log p(\text{spam},x) = -7.208$, $\log p(\text{ham},x) = -10.447$, $p(\text{spam}|x) = 0.9623$
 'meeting agenda tomorrow': $\log p(\text{spam},x) = -10.504$, $\log p(\text{ham},x) = -$
 7.557 , $p(\text{spam}|x) = 0.0499$
 'win cash tomorrow': $\log p(\text{spam},x) = -8.307$, $\log p(\text{ham},x) = -9.349$, $p(\text{spam}|x) = 0.7392$

sanity-check the first message by hand: all three words have $\hat{\theta}_{\text{spam},j} = 0.12$, so
 $\log p(\text{spam},x) = \log \frac{3}{7} + 3 \log 0.12 = -0.847 - 6.361 = -7.208$. matches. note the
 third message: "tomorrow" is ham evidence, so the posterior retreats from 0.96 to
 0.74 — the votes are additive in log space, exactly as the next section formalises.

the same pipeline at real scale — sklearn's `MultinomialNB` on an actual email
 corpus, with vectorisers and evaluation — lives in this notebook:

7 naive bayes is a linear classifier

for the binary bernoulli model, expand the log posterior odds:

$$\begin{aligned}
 \log \frac{p(y=1|x)}{p(y=0|x)} &= \log \frac{\pi_1}{\pi_0} + \sum_{j=1}^d \left[x_j \log \frac{\theta_{1j}}{\theta_{0j}} + (1-x_j) \log \frac{1-\theta_{1j}}{1-\theta_{0j}} \right] \\
 &= \underbrace{\log \frac{\pi_1}{\pi_0} + \sum_j \log \frac{1-\theta_{1j}}{1-\theta_{0j}}}_b + \sum_j \underbrace{\log \frac{\theta_{1j}(1-\theta_{0j})}{\theta_{0j}(1-\theta_{1j})}}_{w_j} x_j = w^\top x + b.
 \end{aligned}$$

the multinomial case is even quicker: $\log p(x|c)$ is $\sum_j x_j \log \theta_{cj}$ plus a class-
 independent term, so $w_j = \log(\theta_{1j}/\theta_{0j})$. either way the decision boundary $w^\top x + b =$
 0 is a hyperplane — the same hypothesis class as **logistic regression**.

the difference is purely in how the weights are fit: naive bayes estimates w *genera-*
tively (model $p(x|y)$, read the weights off the counts), logistic regression *discrimi-*
natively (optimise the conditional likelihood of $y|x$ directly). the generative route
 needs far less data to reach its (higher) asymptotic error; the discriminative route
 wins when data is plentiful.¹

*gaussian nb with a shared
 covariance per feature is
 likewise linear; with per-class
 variances the boundary picks up
 quadratic terms*

¹ng and jordan (2002), *on discriminative vs. generative classifiers: a comparison of logistic
 regression and naive bayes*, nips 14 — the classic formalisation of the two regimes.

8 the calibration caveat

naive bayes is routinely a good *classifier* and a bad *probability estimator*. correlated features are the culprit: if thirty features all restate the same evidence, the model counts that evidence thirty times, and the posterior gets driven towards 0 or 1 exponentially fast. the argmax is often unharmed — double-counting frequently pushes scores *away* from the boundary on the correct side, and 0–1 loss only asks for the right ordering (Hastie, Trevor and Tibshirani, Robert and Friedman, Jerome, 2009) — but the reported $p(\text{spam} \mid x) = 0.9999997$ is fiction.

practical consequences:

- never threshold raw nb posteriors against costs (“only delete if 99% sure”) without recalibrating — platt scaling or isotonic regression on a held-out set fixes most of it.
- ranking by posterior (triage queues, top-k retrieval) is usually fine: miscalibration is closer to a monotone distortion than a reshuffle.
- see [performance metrics](#) for measuring this (reliability diagrams, brier score) rather than assuming it.

9 when naive bayes wins

despite — really, because of — its bias, naive bayes is the right first model surprisingly often (Hastie, Trevor and Tibshirani, Robert and Friedman, Jerome, 2009):

- **small n , large d .** with thousands of features and hundreds of examples, most flexible models drown in variance; nb’s per-feature estimates stay stable. text is the canonical case: bag-of-words puts d in the tens of thousands.
- **speed and simplicity.** training is one counting pass ($O(nd)$ time, $O(Kd)$ memory), prediction is a dot product, and online updates are increments. as a production baseline it is nearly free.
- **streaming and drifting data.** counts can be decayed or windowed; the model retrains continuously without an optimiser.
- **a floor for fancier models.** if your [svm](#) or fine-tuned transformer cannot beat smoothed naive bayes on your corpus, your pipeline has a bug. (on the imdb sentiment benchmark, an nb-flavoured linear model held its own against neural methods for years.)²

its losses are just as predictable: plentiful data with strong feature interactions (vision, tabular data with cross-effects) hands the win to discriminative and non-parametric models that can spend their variance budget.

²domingos and pazzani (1997), *on the optimality of the simple bayesian classifier under zero-one loss*, machine learning 29, explain the robustness under 0–1 loss; wang and manning (2012) report the strong nb/svm text baselines.

references

Deisenroth, Marc Peter and Faisal, A. Aldo and Ong, Cheng Soon (2020). *Mathematics for Machine Learning*, Cambridge University Press. Hastie, Trevor and Tibshirani, Robert and Friedman, Jerome (2009). *The Elements of Statistical Learning*, Springer. Wasserman, Larry (2010). *All of Statistics: A Concise Course in Statistical Inference*, Springer.

10 see also

- **logistic regression** — the discriminative twin: same hyperplane, opposite estimation philosophy
- **support vector machines** — what to try when the linear boundary is right but the probabilistic story is not
- **decision trees** — the opposite bias: interactions first, independence never
- **ensembles** — where the variance-hungry models claw back the large-data regime
- **k-nearest neighbours** — the other famously simple baseline, with the mirror-image failure mode in high dimensions