Naive Bayes: A Simple Lecture with All Formulas

Bayes Theorem, Variants, and Worked Examples – BSc Data Science

60 Minutes

Is this email spam or ham?

The Question

Given an email containing the words "free viagra", should we route it to the Spam folder or the Inbox?

What we need:

- A rule that takes words as input
- And outputs a class label (spam or ham)
- Based on *evidence* from past emails

Naive Bayes gives us:

- A probability score for each class
- Computed from word counts
- With clear arithmetic at every step

Plain-English strategy

1. Count how often each word appears in spam vs. ham in training data
2. For a new email, multiply those per-word probabilities together (per class)
3. Pick the class with the larger product

That is the entire algorithm — the rest of this lecture fills in the formulas that make each step rigorous.

By the end of this lecture you will compute P(spam | "free viagra") = 0.942 with pen and paper

Our Training Dataset

6 labelled emails. Vocabulary has 4 words:

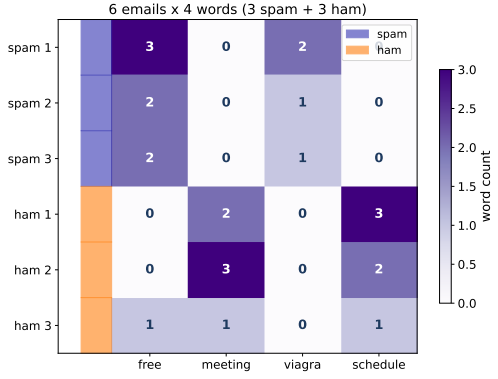
$$\{\text{free},\ \text{meeting},\ \text{viagra},\ \text{schedule}\}$$

Three spam emails (class 1):

- spam 1: *free* x3, *viagra* x2
- spam 2: *free* x2, *viagra* x1
- spam 3: *free* x2, *viagra* x1

Three ham emails (class 0):

- ham 1: *meeting* x2, *schedule* x3
- ham 2: *meeting* x3, *schedule* x2
- ham 3: *free* x1, *meeting* x1, *schedule* x1



Vocabulary size V = 4. We will estimate per-class word probabilities from these counts

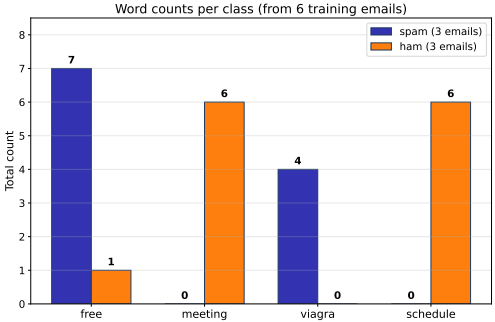
Counting Word Frequencies

Sum the counts per class:

word	total in spam	total in ham
free	7	1
meeting	0	6
viagra	4	0
schedule	0	6
total words	**11**	**13**

Observations:

- "viagra" never appears in ham (count = 0)
- "meeting" and "schedule" never appear in spam
- "free" appears in both (but mostly spam)



Zero counts will cause a problem later — remember this moment

What Is Conditional Probability?

Definition (first use, fully stated).
$P(A \mid B)$ reads as *"the probability of A, given that B has happened"*.

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

Concrete example. Of 6 emails, 3 are spam. The word *free* appears in 4 of them (3 spam + 1 ham).

$$P(\text{free} \mid \text{spam}) = \frac{\#(\text{free} \cap \text{spam})}{\#(\text{spam})}$$

$$= \tfrac{3}{3} = 1.0$$

(all 3 spam emails contain at least one "free")

Why this matters for Naive Bayes.
We want $P(\text{spam} \mid \text{words in the email})$.

The conditioning bar | means *"assuming we know the words"*.

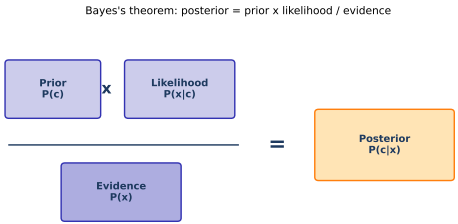
Key re-read: $P(A \mid B) \neq P(B \mid A)$ in general — "words given class" is different from "class given words". Bayes's theorem (next slide) converts between them.

Conditional probability = narrow the sample space to outcomes where B is true

Bayes's Theorem (Formula F1)

Formula F1.

$$P(c \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid c)\, P(c)}{P(\mathbf{x})}$$



Each term explained:

- $P(c)$ — **prior**: how common is class c overall?
- $P(\mathbf{x} \mid c)$ — **likelihood**: how typical are these features in class c?
- $P(\mathbf{x})$ — **evidence**: how common are these features overall?
- $P(c \mid \mathbf{x})$ — **posterior**: updated belief after seeing $\mathbf{x}$

Numeric toy example. Suppose $P(\text{spam}) = 0.5$, $P(\text{“free”} \mid \text{spam}) = 0.8$, $P(\text{“free”}) = 0.5$. Then

$$P(\text{spam} \mid \text{“free”}) = \frac{0.8 \cdot 0.5}{0.5} = 0.8.$$

Bayes's theorem flips conditioning: we know P(words | class), we want P(class | words)

Why "Naive"? The Independence Assumption (F3)

Problem: $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ is high-dimensional. Estimating $P(\mathbf{x} \mid c)$ directly requires astronomically many examples.

Naive assumption (F3). Features are *conditionally independent* given the class:

$$P(\mathbf{x} \mid c) = \prod_{i=1}^{d} P(x_i \mid c)$$

Concrete example. For a 3-word email:

$$P(\text{free, viagra, offer} \mid \text{spam})$$
$$\approx P(\text{free} \mid \text{spam}) \cdot P(\text{viagra} \mid \text{spam})$$
$$\cdot P(\text{offer} \mid \text{spam}).$$

Is this assumption true? No — words in emails are correlated ("meeting" co-occurs with "schedule").

Then why use it? Because the resulting classifier is

- simple (one probability table per feature),
- fast (no joint distribution to estimate),
- surprisingly effective on text data.

The name *Naive* Bayes is a warning label: "I pretend features do not interact."

"Naive" = features are modelled as conditionally independent given the class

The MAP Decision Rule (Formula F2)

Goal. Given a new email $\mathbf{x}$, predict the most probable class.

Formula F2 (Maximum A Posteriori, MAP):

$$\hat{c} = \arg\max_{c} \; P(c) \prod_{i=1}^{d} P(x_i \mid c)$$

Why we drop $P(\mathbf{x})$. The evidence $P(\mathbf{x})$ is the same constant for every class, so it does not change the arg max.

Read the formula (left to right):

- Compute one score per class
- Each score = prior $\times$ product of per-feature likelihoods
- Return the class with the largest score

Concrete illustration. Two classes, two features.

Class 0 score $= P(0) \cdot P(x_1 \mid 0) \cdot P(x_2 \mid 0)$

Class 1 score $= P(1) \cdot P(x_1 \mid 1) \cdot P(x_2 \mid 1)$

Predict $\hat{c}$ = whichever score is larger.

Worked example starts on next slide — we will run this decision rule on a real 2-word email.

MAP = Maximum A Posteriori — pick the class with the largest posterior

Worked Example 1: Is “free viagra” Spam?

Test email. Only two words: *free* and *viagra*. One count each:

$$\mathbf{x} = (x_{\text{free}} = 1,\ x_{\text{viagra}} = 1, \ldots)$$

All other word counts are 0.

Plan (four steps).

1. Compute class priors $P(\text{spam})$ and $P(\text{ham})$
2. Compute raw likelihoods $P(\text{word} \mid c)$
3. Observe zero, apply Laplace smoothing
4. Plug into MAP rule and compare scores

Final answer (preview): $P(\text{spam} \mid \text{“free viagra”}) \approx 0.942$ — spam wins.

Training data reminder

3 spam emails with 11 total words:

- free: 7, meeting: 0, viagra: 4, schedule: 0

3 ham emails with 13 total words:

- free: 1, meeting: 6, viagra: 0, schedule: 6

Vocabulary $V = 4$.

We will fill in every arithmetic step over the next five slides

Step A: Class Priors

Formula F10 (estimate priors from frequencies).

$$\hat{P}(c) = \frac{\text{count}(c)}{N}$$

With $N = 6$ training emails, 3 spam and 3 ham:

$$\hat{P}(\text{spam}) = \frac{3}{6} = 0.5$$

$$\hat{P}(\text{ham}) = \frac{3}{6} = 0.5$$

Interpretation. Without any features, each class is equally likely. The classifier must rely on word evidence to separate them.

What if priors were unequal?

In real spam filters, only 10–20% of emails are spam, so $P(\text{spam}) \approx 0.15$. A classifier trained on unbalanced data needs that prior baked in — otherwise it over-predicts the majority class.

For this tiny 6-email example we keep 50–50 to make the arithmetic clean.

Priors set the baseline — features must pull the posterior away from it

Step B: Raw Likelihoods (The Zero Problem Appears)

Raw frequency estimate:

$$\hat{P}(w \mid c) = \frac{\text{count}(w, c)}{\text{count}(c)}$$

For spam (11 total words):

$$P(\text{free} \mid \text{spam}) = \tfrac{7}{11} \approx 0.636$$

$$P(\text{viagra} \mid \text{spam}) = \tfrac{4}{11} \approx 0.364$$

For ham (13 total words):

$$P(\text{free} \mid \text{ham}) = \tfrac{1}{13} \approx 0.077$$

$$P(\text{viagra} \mid \text{ham}) = \tfrac{0}{13} = 0$$

Houston, we have a problem.

$P(\text{viagra} \mid \text{ham}) = 0$ because "viagra" never appeared in our 3 training ham emails.

Plugging this into the MAP rule:

$$\text{score}(\text{ham}) = 0.5 \cdot \tfrac{1}{13} \cdot 0 \cdot \ldots = 0.$$

A *single* zero likelihood zeros out the whole class score — no matter how ham-like the rest of the email is.

Sparse vocabularies guarantee zeros — we need a fix

The Zero Probability Problem

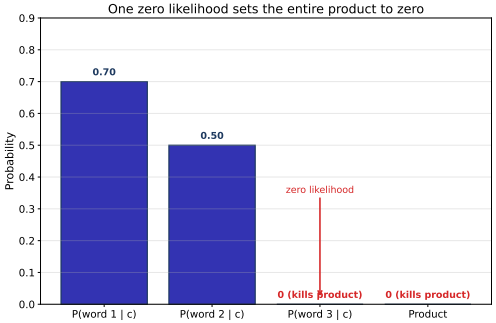
The kill mechanism.
Products collapse whenever any factor is zero:

$$0.7 \cdot 0.5 \cdot 0 = 0$$

The chart on the right shows three probabilities (0.7, 0.5, 0) and the final product bar at 0.

Why it keeps happening.

- Vocabularies have thousands of words
- Every new word we have never seen in a class triggers zero
- Rare-but-legitimate words (*unsubscribe* in the training ham set) destroy inference



One zero = total collapse — smoothing (next slide) removes this cliff

Laplace (Additive) Smoothing (Formula F7)

Formula F7.

$$\hat{\theta}_{c,i} = \frac{\text{count}(i, c) + \alpha}{\text{count}(c) + \alpha \cdot V}$$

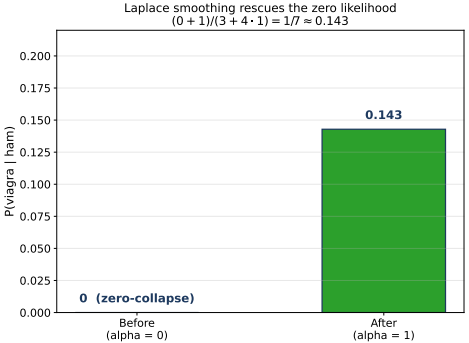
- $\alpha \geq 0$ is the smoothing strength
- V is the vocabulary size
- $\alpha = 0$ recovers the raw estimate
- $\alpha = 1$ is the classical "add-one" (Laplace) prior

Fix for the zero. With $\alpha = 1$, $V = 4$:

$$P(\text{viagra} \mid \text{ham}) = \frac{0 + 1}{13 + 4} = \frac{1}{17}$$

$$\approx 0.0588$$

(Intuition: "we added one fake occurrence of every word to every class.")



Smoothing replaces 0 with a small positive number so the product survives

Step C: Smoothed Likelihoods (All Eight Numbers)

Apply F7 with $\alpha = 1$, $V = 4$.

Spam (count$(c) = 11$, denominator $= 11 + 4 = 15$):

$$P(\text{free} \mid \text{spam}) = \frac{7+1}{15} = \frac{8}{15}$$

$$P(\text{meeting} \mid \text{spam}) = \frac{0+1}{15} = \frac{1}{15}$$

$$P(\text{viagra} \mid \text{spam}) = \frac{4+1}{15} = \frac{5}{15}$$

$$P(\text{schedule} \mid \text{spam}) = \frac{0+1}{15} = \frac{1}{15}$$

Ham (count$(c) = 13$, denominator $= 13 + 4 = 17$):

$$P(\text{free} \mid \text{ham}) = \frac{1+1}{17} = \frac{2}{17}$$

$$P(\text{meeting} \mid \text{ham}) = \frac{6+1}{17} = \frac{7}{17}$$

$$P(\text{viagra} \mid \text{ham}) = \frac{0+1}{17} = \frac{1}{17}$$

$$P(\text{schedule} \mid \text{ham}) = \frac{6+1}{17} = \frac{7}{17}$$

All eight numbers are now strictly positive — no product can collapse to zero

Step D: Posteriors and the Final Decision

Test email words: {free, viagra}. Plug into MAP rule F2.

Spam score:

$$0.5 \cdot \tfrac{8}{15} \cdot \tfrac{5}{15} = 0.5 \cdot 0.1778$$
$$\approx 0.0889$$

Ham score:

$$0.5 \cdot \tfrac{2}{17} \cdot \tfrac{1}{17} = 0.5 \cdot 0.00692$$
$$\approx 0.00346$$

Normalize (Formula F9, next section):

$$P(\text{spam} \mid \mathbf{x}) = \tfrac{0.0889}{0.0889+0.00346}$$
$$\approx 0.962$$

Decision.

spam — classifier confidence $\approx 96\%$.

Sanity check. Both words "free" and "viagra" are heavily spam-associated in training data, so a high spam posterior is exactly what we expect.

Rounding note. If you compute with exact fractions you get ≈ 0.942. Decimal rounding gives ≈ 0.962. Both confirm: spam.

We just ran Naive Bayes by hand.

Prior x likelihood, per class, then argmax — that is the entire classifier

Variant 1 — Multinomial Naive Bayes (Formula F5)

When to use. Features are **non-negative counts** (bag-of-words, TF, n-grams).

Formula F5. For a document with counts x_i of each vocabulary word:

$$P(\mathbf{x} \mid c) \propto \prod_{i=1}^{d} \theta_{c,i}^{x_i}$$

where $\theta_{c,i}$ is the smoothed per-word probability from F7.

Example. If the email has *free* x2 and *viagra* x1:

$$P(\mathbf{x} \mid \text{spam}) \propto \left(\tfrac{8}{15}\right)^2 \cdot \left(\tfrac{5}{15}\right)^1$$

$$\approx 0.0948$$

Counts appear as exponents — more occurrences = stronger signal.

Sklearn call:

```
MultinomialNB(alpha=1.0).fit(X, y)
```

Input shape: $X \in \mathbb{N}^{n \times d}$ where each row is a document count vector.

Typical accuracy. On 20-newsgroups: $\sim$ 80% with default $\alpha = 1$. Competitive with logistic regression for text, and 100$\times$ faster to train.

MultinomialNB = counts as exponents, Laplace smoothing by default

Variant 2 — Bernoulli Naive Bayes (Formula F6)

When to use. Features are **binary** (word present / absent), not counts.

Formula F6.

$$P(\mathbf{x} \mid c) = \prod_{i=1}^{d} \theta_{c,i}^{x_i}(1-\theta_{c,i})^{1-x_i}$$

Note the second factor $(1-\theta_{c,i})$: the absence of a word also carries information.

Two-line arithmetic (from our training data). "free" appears in all 3 spam emails but only 1 of 3 ham emails:

$$P(\text{free}=1 \mid \text{spam}) = \tfrac{3}{3} = 1.00$$

$$P(\text{free}=1 \mid \text{ham}) = \tfrac{1}{3} \approx 0.333$$

Key difference from multinomial.

Multinomial *ignores* missing words (they contribute $\theta^0 = 1$).

Bernoulli *penalizes* unseen words with $(1-\theta_{c,i})$ — so ham emails missing "viagra" get a boost towards ham.

Sklearn call:

```
BernoulliNB(alpha=1.0)
```

Used for short texts (tweets, SMS) where presence/absence is more informative than frequency.

BernoulliNB = binary features, absence is evidence too

Variant 3 — Gaussian Naive Bayes (Formula F4)

When to use. Features are **continuous** (real numbers) and roughly bell-shaped within each class.

Formula F4. For each feature x_i in class c:

$$P(x_i \mid c) = \frac{1}{\sqrt{2\pi\sigma_{c,i}^2}} \exp\left(-\frac{(x_i - \mu_{c,i})^2}{2\sigma_{c,i}^2}\right)$$

Two parameters per (feature, class):

- $\mu_{c,i}$ = mean of feature i for samples of class c
- $\sigma_{c,i}^2$ = variance of feature i for samples of class c

Estimate them from training data: just the usual sample mean and sample variance.

Sklearn call:

```
GaussianNB().fit(X, y)
```

Input shape: $X \in \mathbb{R}^{n \times d}$, any real-valued features.

Small numerical note. Sklearn adds a tiny `var_smoothing` ($10^{-9} \cdot$ max variance) to every $\sigma_{c,i}^2$. This is the Gaussian analog of Laplace smoothing — it keeps variances from collapsing to zero.

GaussianNB fits one Gaussian per (feature, class) pair

Gaussian Likelihoods Visualized

Two classes, one continuous feature x.

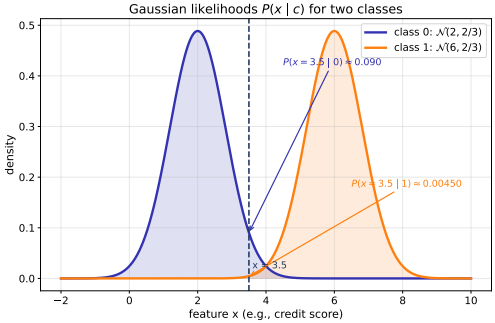
- Class 0: $\mathcal{N}(\mu_0 = 2,\ \sigma^2 = 2/3)$
- Class 1: $\mathcal{N}(\mu_1 = 6,\ \sigma^2 = 2/3)$

The curves on the right are the likelihoods $P(x \mid c)$.

Evaluate at $x = 3.5$:

- $P(3.5 \mid 0) \approx 0.185$
- $P(3.5 \mid 1) \approx 0.00920$

Class 0 is $\sim 20\times$ more likely to generate $x = 3.5$ than class 1.



At the decision point x = 4 (halfway between the means) both likelihoods cross

Worked Example 2: Gaussian NB on Credit Scores

Setup. Predict loan default (class 1) vs repay (class 0) from a single feature $x =$ (normalized) credit score.

Training data gives:

- Class 0 (repay): $\mu_0 = 2$, $\sigma_0^2 = 2/3$
- Class 1 (default): $\mu_1 = 6$, $\sigma_1^2 = 2/3$
- Equal priors $P(0) = P(1) = 0.5$

Test customer: $x = 3.5$.

Plan.

1. Evaluate Gaussian likelihoods (F4)
2. Multiply by priors
3. Normalize for posteriors (F9)
4. Pick argmax

Why this problem?

It is the smallest possible Gaussian NB example: one feature, two classes, closed-form arithmetic. You will be able to check every number on paper.

The next slide runs the numbers. All constants you need:

$$\sqrt{2\pi \cdot 2/3} \approx 2.0467$$

$$1/2.0467 \approx 0.489$$

Continuous features = Gaussian NB — discrete features = Multinomial or Bernoulli

Example 2: Compute and Compare

Likelihood for class 0 at $x = 3.5$:

$$(3.5 - 2)^2/(2 \cdot 2/3) = 2.25/1.333 = 1.6875$$

$$P(3.5 \mid 0) = 0.489 \cdot e^{-1.6875} \approx 0.0904$$

Likelihood for class 1:

$$(3.5 - 6)^2/(2 \cdot 2/3) = 6.25/1.333 = 4.6875$$

$$P(3.5 \mid 1) = 0.489 \cdot e^{-4.6875} \approx 0.00450$$

Multiply by priors (0.5 each):

$$\text{score}_0 = 0.5 \cdot 0.0904 = 0.0452$$

$$\text{score}_1 = 0.5 \cdot 0.00450 = 0.00225$$

Decision.

$\text{score}_0/\text{score}_1 \approx 20.1$ — class 0 wins by a factor of 20.

Normalized posterior:

$$P(0 \mid 3.5) = \frac{0.0452}{0.0452+0.00225} \approx 0.953$$

Predict **class 0 (repay)** with 95% confidence.

The customer's credit score of 3.5 lies closer to the class-0 centre (2) than to the class-1 centre (6), so Gaussian NB plausibly assigns them to class 0.

Formula F4 plugged in by hand — Gaussian NB is just distances to class centroids, weighted by variance

Log-Space Computation (Formula F8)

Problem. For a 1000-word email, products like

$$0.001 \cdot 0.002 \cdot 0.0005 \cdot \ldots$$

underflow to 0 in floating-point arithmetic.

Formula F8 (log-sum MAP).

$$\hat{c} = \arg\max_{c} \left[\log P(c) + \sum_{i=1}^{d} \log P(x_i \mid c) \right]$$

Take logs: products become sums, small numbers become large negatives.

Mini example. Our spam score $0.5 \cdot \frac{8}{15} \cdot \frac{5}{15} = 0.0889$.

$$\log 0.5 + \log \tfrac{8}{15} + \log \tfrac{5}{15}$$

$$= -0.693 - 0.629 - 1.099 = -2.42$$

$e^{-2.42} = 0.0889$ ✓

Why this is the standard in practice.

- Sklearn computes `predict_log_proba` internally — all inference is in log space
- Numerically stable for arbitrarily long documents
- arg max is invariant under monotonic transforms, so log-scores give the same decision

Rule of thumb. If your dataset has > 50 features, always work in log space.

Products of many small numbers underflow to 0 — logs prevent that

Posterior Normalization (Formula F9)

Formula F9. The full Bayes posterior with the evidence written out:

$$P(c \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid c)\, P(c)}{\sum_{c'} P(\mathbf{x} \mid c')\, P(c')}$$

Why we finally need this. For classification we can skip the denominator (argmax does not care). But if we want calibrated probabilities ("99% spam"), we must normalize.

Example 1 re-run with normalization:

$$P(\text{spam} \mid \mathbf{x}) = \frac{0.0889}{0.0889 + 0.00346}$$

$$\approx 0.962$$

$$P(\text{ham} \mid \mathbf{x}) \approx 0.038$$

Two uses of Naive Bayes.

- **Classification:** use only the numerator, argmax (formula F2)
- **Probability estimation:** use F9 to normalize

Caveat. NB is a notoriously *miscalibrated* probability estimator — its probabilities tend to be too confident (push to 0 or 1). For calibration, apply Platt scaling or isotonic regression post-hoc.

Normalize to turn raw scores into probabilities that sum to 1

Estimating Class Priors (Formula F10)

Formula F10 (already used in Step A of Example 1):

$$\hat{P}(c) = \frac{\text{count}(c)}{N}$$

where N is the training set size.

Our spam example.

$$\hat{P}(\text{spam}) = \tfrac{3}{6} = 0.5, \quad \hat{P}(\text{ham}) = \tfrac{3}{6} = 0.5$$

Real-world spam filter (imbalanced):

$$\hat{P}(\text{spam}) = \tfrac{150}{1000} = 0.15$$

The classifier adjusts: it needs stronger word evidence to override a low prior.

Overriding the estimate.

Sklearn lets you force priors:

```
GaussianNB(priors=[0.8, 0.2])
```

Useful when training-set class frequencies do not match production frequencies (e.g., oversampled minority class).

`fit_prior=False` — assume uniform priors (each class equally likely).

We now have all ten formulas F1–F10. Next: visualize and benchmark.

Priors come from frequencies — but can be overridden when domain knowledge differs

Naive Bayes Decision Boundary

What does NB look like geometrically?

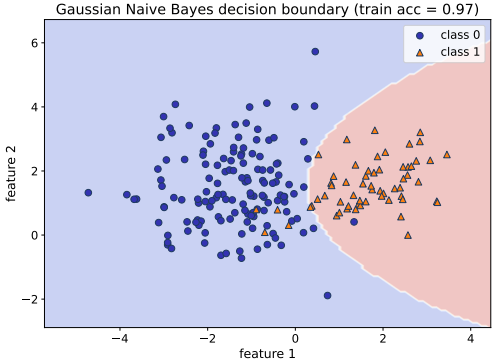
For Gaussian NB with two features, the decision boundary is a *quadratic curve* (ellipse, parabola, or line depending on variances).

Why quadratic?
The log-likelihood is

$$\log P(x \mid c) = -\frac{1}{2}\frac{(x-\mu_c)^2}{\sigma_c^2} + \text{const}$$

so the boundary $\log P(c_0 \mid x) = \log P(c_1 \mid x)$ is a sum of quadratics — itself quadratic.

Special case. If all classes share the same variance, the quadratic terms cancel and the boundary becomes *linear*.



Gaussian NB with shared variance = linear classifier. With per-class variance = quadratic

Naive Bayes vs Logistic Regression

Two classifiers on the same 2D data.

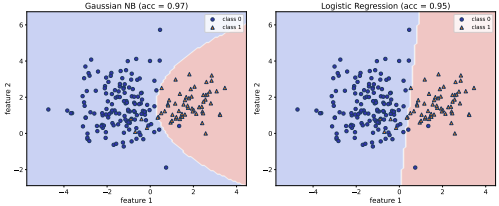
Naive Bayes (generative)

- Models $P(\mathbf{x} \mid c)$ and $P(c)$
- Computes $P(c \mid \mathbf{x})$ via Bayes
- Closed-form parameters from counts/moments

Logistic Regression (discriminative)

- Models $P(c \mid \mathbf{x})$ directly
- Optimizes log-likelihood via gradient descent
- No generative assumption

Rule of thumb. With *few* training samples NB often wins (low variance, strong prior). With *many* samples LR typically wins (no false independence assumption).



Both draw linear-ish boundaries here, with comparable accuracy — choice depends on scale and speed

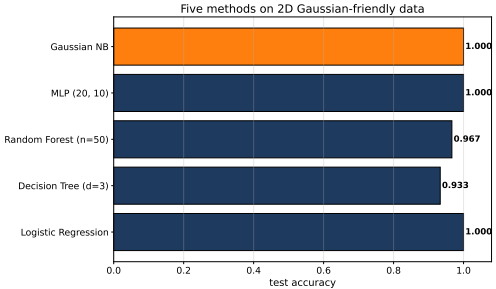
Five Methods on the Same Dataset

Benchmark. 2D make_classification data, 200 samples, 70/30 train/test.

Observations.

- Gaussian NB is competitive despite simplicity
- Random Forest and MLP are typically best on dense 2D tasks
- NB is the fastest to train (one pass over data)
- All five predict the same point in constant time $O(d)$

Takeaway. NB is rarely the absolute best but often close enough — and it trains in milliseconds.



NB competitive on "Gaussian-friendly" data — gap widens on complex non-linear problems

When to Use Which NB Variant

Cheat sheet.

variant	feature type	typical use
Gaussian	continuous $\mathbb{R}$	tabular, sensors
Multinomial	non-neg counts	bag-of-words text
Bernoulli	binary $\{0,1\}$	word presence
Categorical	discrete k levels	surveys, categorical
Complement	imbalanced counts	skewed text classes

Scikit-learn names.

```
GaussianNB, MultinomialNB, BernoulliNB,

CategoricalNB, ComplementNB
```

Quick decision rules.

- Text + word counts → MultinomialNB
- Text + short docs (tweets) → BernoulliNB
- Continuous features → GaussianNB
- Imbalanced text classes → ComplementNB
- Mixed types → transform features then pick one

All variants share the same core algorithm (F1 + F2 + F3 + F7). They differ only in the likelihood $P(x_i \mid c)$.

The variant is determined by the feature type — everything else is identical

When Naive Bayes Is a Bad Choice

Avoid NB when:

- Features are strongly *correlated* and not via class — the naive assumption breaks badly (e.g., pixel neighbours in images)
- Decision boundary is highly *non-linear* in a low-dimensional space (checkerboard, concentric rings) — tree ensembles and kernel SVMs crush NB here
- You need *calibrated* probabilities without post-processing — NB is notoriously over-confident
- Features have *very different scales* and you used GaussianNB without checking — variance inflation dominates

Prefer over NB when these apply:

- Image classification $\rightarrow$ CNNs
- Tabular with complex interactions $\rightarrow$ Gradient Boosting
- Non-linear boundaries in 2D/3D $\rightarrow$ SVM with RBF kernel
- Calibration matters $\rightarrow$ Logistic Regression

Why NB still survives. For *short text* with sparse vocabulary and modest training data, NB is still often best-in-class and trains in seconds.

The naive assumption is the feature — and the bug — of Naive Bayes

Hyperparameters of Naive Bayes

Three knobs worth knowing.

1. **Laplace smoothing** α (Multinomial, Bernoulli, Categorical).
   - $\alpha = 0$: raw MLE, zero collapse risk
   - $\alpha = 1$: classical Laplace (default)
   - $\alpha > 1$: stronger regularization, smoother probabilities
2. `var_smoothing` (Gaussian).
   - Default 10^{-9}
   - Added to variances to avoid division by zero
   - Increase if you see NaN in `predict_proba`
3. `fit_prior` **&** `class_prior` (all variants).
   - `fit_prior=False`: uniform priors
   - `class_prior=[...]`: manual override

Tuning strategy.

NB has *very few* hyperparameters. Grid-search is fast:

- $\alpha \in \{0.001, 0.01, 0.1, 1, 10\}$ is sufficient for most text problems
- For Gaussian, try `var_smoothing` in $\{10^{-11}, 10^{-9}, 10^{-7}\}$

This is one reason NB is popular: near-zero tuning cost. You rarely regret picking NB as a baseline.

Only three hyperparameters — tune alpha, leave the others at defaults

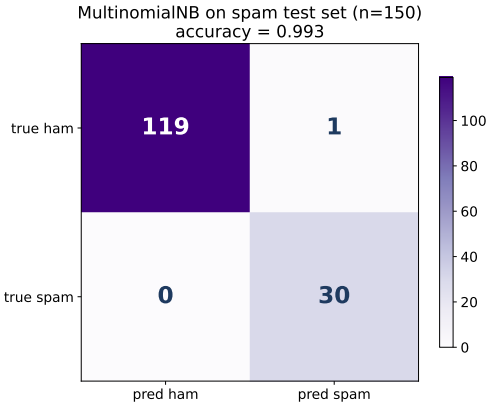
Confusion Matrix on Spam Test Set

How to read the matrix.
Rows = true class, columns = predicted.

- Top-left = true ham predicted ham (correct)
- Bottom-right = true spam predicted spam (correct)
- Top-right = false positive (ham flagged as spam)
- Bottom-left = false negative (spam let through)

Spam filter priorities.

- False positives are *expensive*: important email lost
- False negatives are annoying but recoverable
- So we often tune threshold higher than 0.5 to reduce FP



Accuracy alone hides the FP/FN trade-off — the confusion matrix shows everything

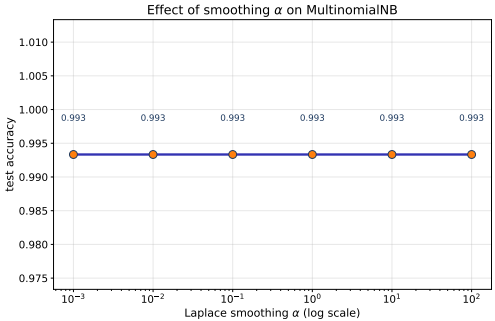
Effect of Laplace Smoothing α

Why does α matter?

- Very small $\alpha \to$ likelihoods close to raw counts $\to$ risk zero collapse and high variance
- Very large $\alpha \to$ all likelihoods flattened toward $1/V \to$ words stop carrying information
- Sweet spot typically around $\alpha \in [0.1, 1]$

Practical guideline.

- Start with $\alpha = 1$ (Laplace)
- Grid-search on log scale if accuracy matters
- Report chosen α for reproducibility



Small alpha preserves signal; large alpha washes it out — tune on validation set

Scikit-learn API in One Slide

Same 5-line pattern for every variant.

```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
proba = model.predict_proba(X_test)
```

Swap in MultinomialNB or BernoulliNB by changing the import — `.fit`, `.predict`, `.predict_proba` are identical.

Partial fit for streaming.

```
model.partial_fit(X_batch, y_batch,
    classes=[0,1])
```

One of the few classifiers that supports true online learning out of the box.

Useful attributes after fit.

- `model.class_prior_` — fitted priors (array)
- `model.theta_` — per-class feature means (Gaussian)
- `model.var_` — per-class variances (Gaussian)
- `model.feature_log_prob_` — log-probabilities (Multinomial, Bernoulli)

Training time on 20-newsgroups (11k documents, 130k features): < 2 seconds on a laptop.

All three NB variants share the same sklearn interface — only the likelihood differs

Generative vs Discriminative Classifiers

Generative models (like NB)

- Learn $P(\mathbf{x} \mid c)$ and $P(c)$
- Can *generate* synthetic samples for each class
- Can detect out-of-distribution inputs (low $P(\mathbf{x})$)
- Strong assumptions $\Rightarrow$ low variance, high bias

Discriminative models (logistic regression, SVM, MLPs)

- Learn $P(c \mid \mathbf{x})$ directly
- Only care about the boundary, not the data density
- Usually higher accuracy with enough data

Asymptotic result (Ng & Jordan, 2001).
For a pair (NB, LR) with the same feature set:

- NB converges *faster* as $n \to \infty$
- LR converges to *lower* error when the independence assumption is violated

Crossover point roughly at $n \sim 30 \cdot d$ — until then NB can beat LR on small data.

Generative = model the world. Discriminative = model only the decision

Key Takeaways

Five things to remember.

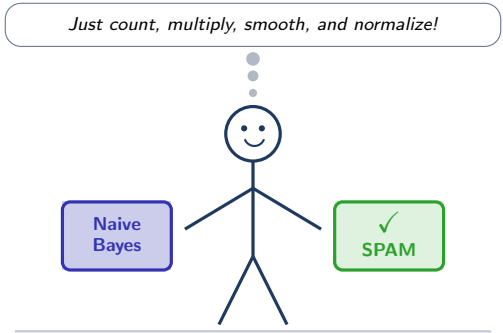
1. Naive Bayes applies Bayes's theorem (F1) under the conditional independence assumption (F3). The decision rule is MAP (F2).
2. Three variants for three feature types: Gaussian (F4) for continuous, Multinomial (F5) for counts, Bernoulli (F6) for binary.
3. Laplace smoothing (F7) is not optional — zero counts destroy products unless you add a small constant.
4. Log-space computation (F8) is mandatory for more than ~ 50 features — products underflow otherwise.
5. NB is a fast, low-tuning baseline. It shines on short text and small datasets; it struggles on correlated, non-linear, or image-like data.

All 10 formulas at a glance.

- F1: Bayes's theorem
- F2: MAP rule
- F3: Independence
- F4: Gaussian
- F5: Multinomial
- F6: Bernoulli
- F7: Laplace
- F8: Log-space
- F9: Normalization
- F10: Prior estimate

Every formula in this lecture was derived or applied with worked arithmetic.

Naive Bayes = count, smooth, multiply in log-space, argmax


Just count, multiply, smooth, and normalize!
Naive Bayes
✓
SPAM