Fondamenti di Analisi dei Dati

from data analysis to predictive techniques

Prof. Antonino Furnari (antonino.furnari@unict.it)

Corso di Studi in Informatica
Dip. di Matematica e Informatica
Università di Catania

Università di Catania

Generative Classifiers and Naive Bayes

Generative classification models and the pragmatic power of conditional independence assumptions in machine learning.

Two Approaches to Classification

Discriminative Models

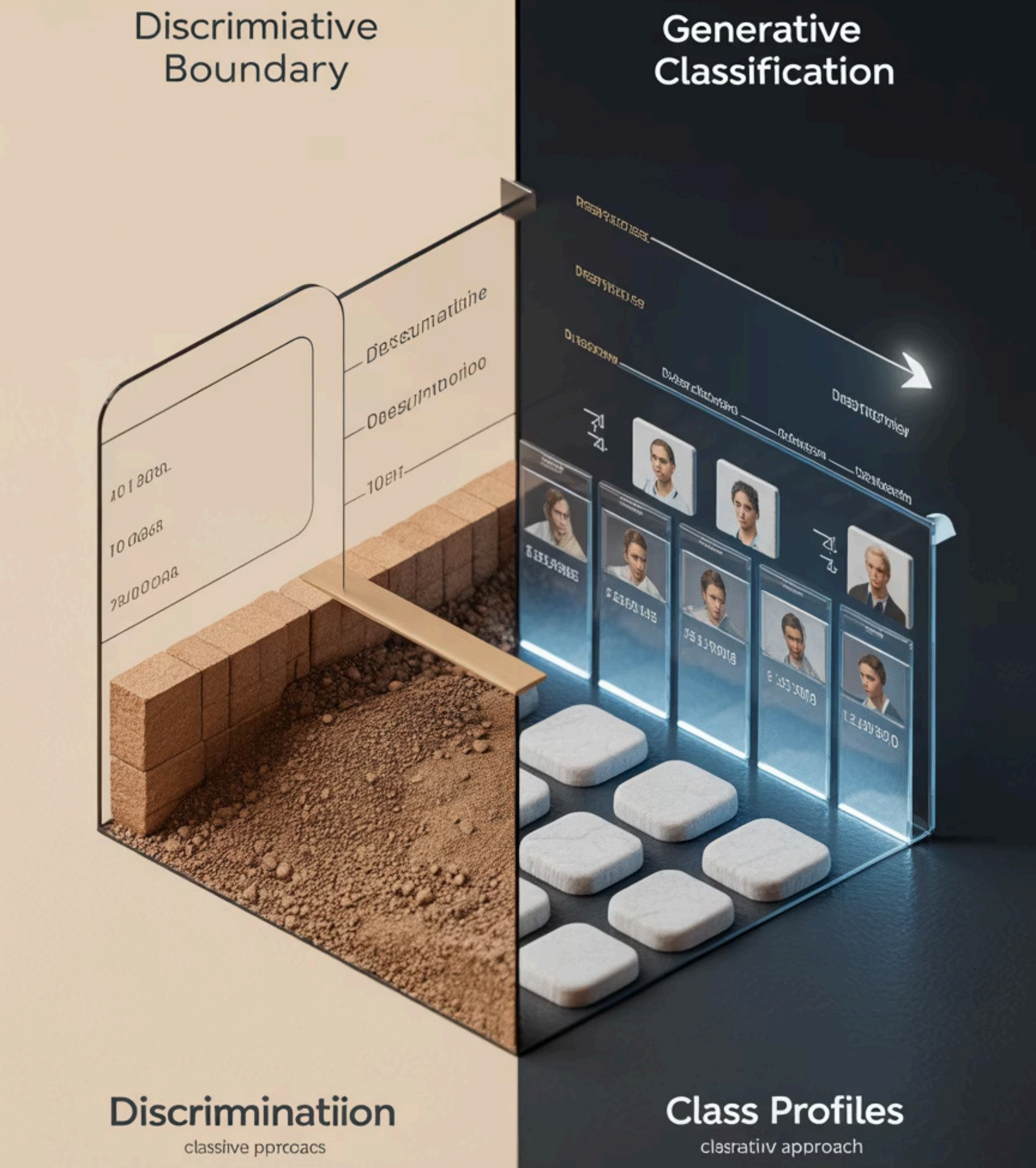
Models like Logistic Regression focus on learning the **decision boundary** between classes. They directly model $P(Y = k|X = \mathbf{x})$.

Goal: Find the line (or surface) that best separates classes.

Generative Models

Models like Naive Bayes learn the **"story" or "profile"** of each class independently. They model the joint probability $P(X, Y)$.

Goal: Understand what each class looks like, then classify based on similarity.



The Generative Approach

Recall that:

$$P(X, Y) = P(X|Y)P(Y)$$

Generative models build a complete probabilistic picture by modeling two components separately:

The Likelihood

$P(X = \mathbf{x}|Y = k)$

"What does a typical X look like for this class?"

The Prior

$P(Y = k)$

"How common is this class in general?"

Once these components are learned, we can use them together for classification through Bayes' theorem.

Maximum A Posteriori (MAP)

The MAP principle finds the most probable class after observing the data:

$$h(\mathbf{x}) = \arg\max_k P(Y = k|X = \mathbf{x})$$

Using Bayes' theorem:

$$h(\mathbf{x}) = \arg\max_k \frac{P(X = \mathbf{x}|Y = k)P(Y = k)}{P(X = \mathbf{x})}$$

Since $P(X = \mathbf{x})$ is independent of class k and X is fixed, we can simplify to:

$$h(\mathbf{x}) = \arg\max_k \underbrace{P(X|Y = k)}_{\text{Likelihood}}\underbrace{P(Y = k)}_{\text{Prior}}$$

□ This elegant formulation shows that classification reduces to computing two quantities: how likely the data is for each class (likelihood) and how common each class is (prior).

Estimating the Prior P(Y)

The prior probability represents how common each class is. We have several approaches to estimate it:

Dataset Proportions

Count examples in your training data. If 800 emails are non-spam and 200 are spam, then P(spam) = 0.2 and P(non-spam) = 0.8.

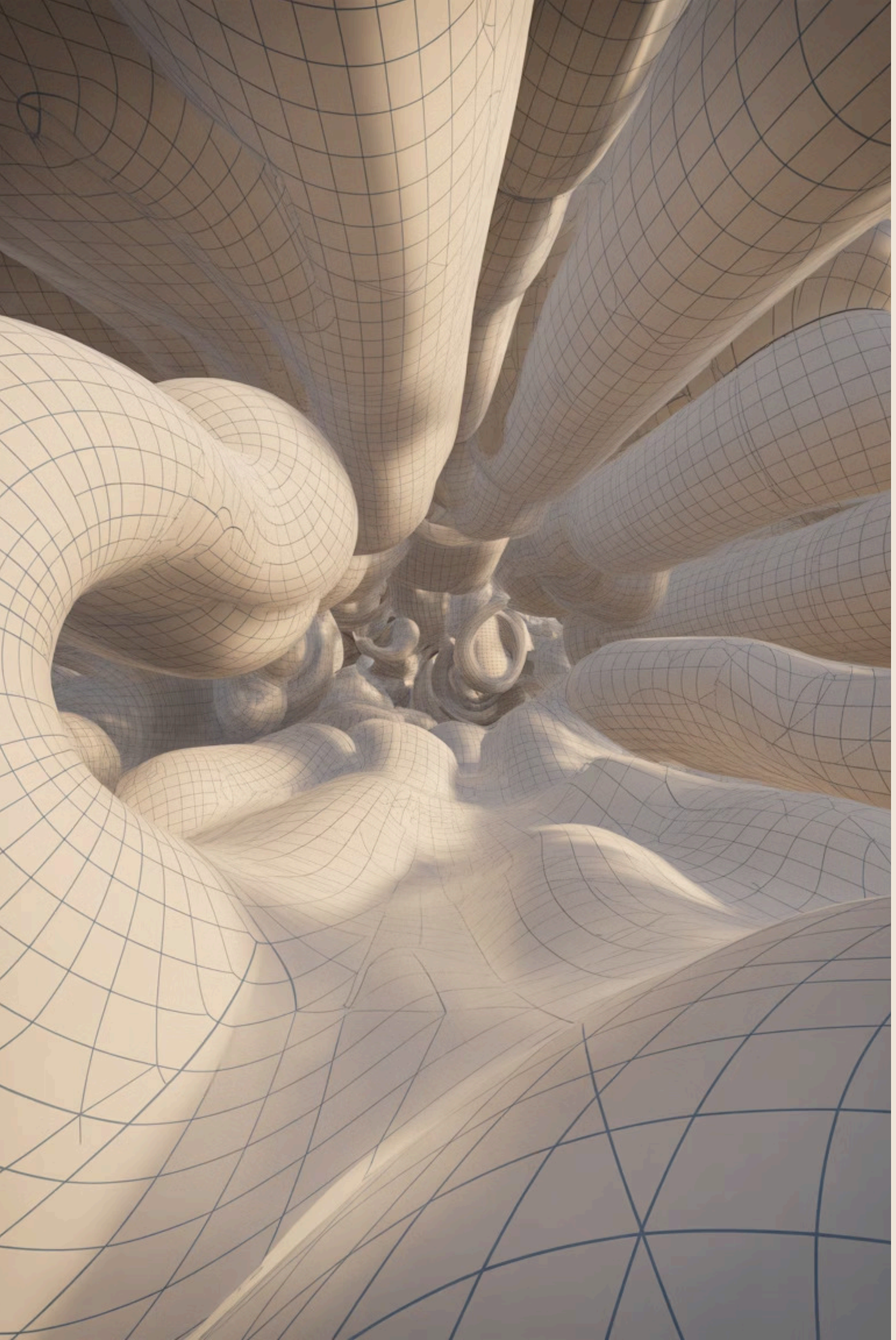
Real-World Statistics

Study actual proportions in the real world through surveys or domain research to set more accurate priors.

Uniform Assumption

When lacking information, assume all classes are equally probable: $P(Y = k) = \frac{1}{m}$ where m is the number of classes.

The prior represents your **degree of belief** about class frequencies before seeing the data.



The Likelihood Challenge

While estimating the prior is straightforward, computing the likelihood $P(X|Y=k)$ is the central challenge of generative modeling. The approach depends on your feature types and dimensionality.

For M different classes, we group observations by class and estimate $P(X_k)$ for each group:

$$P\left(X=x|Y=k\right)=P(X_k)$$

The assumptions we make here determine which generative model we get.

> The entire challenge of generative modeling lies in estimating the likelihood. This is where we encounter (again) the Curse of Dimensionality.

Guiding Example: Spam Detector

Spam detection is a classic and highly illustrative example of generative classification. Our objective is to classify an incoming email as either "spam" or "ham" (not spam) based on its characteristics.

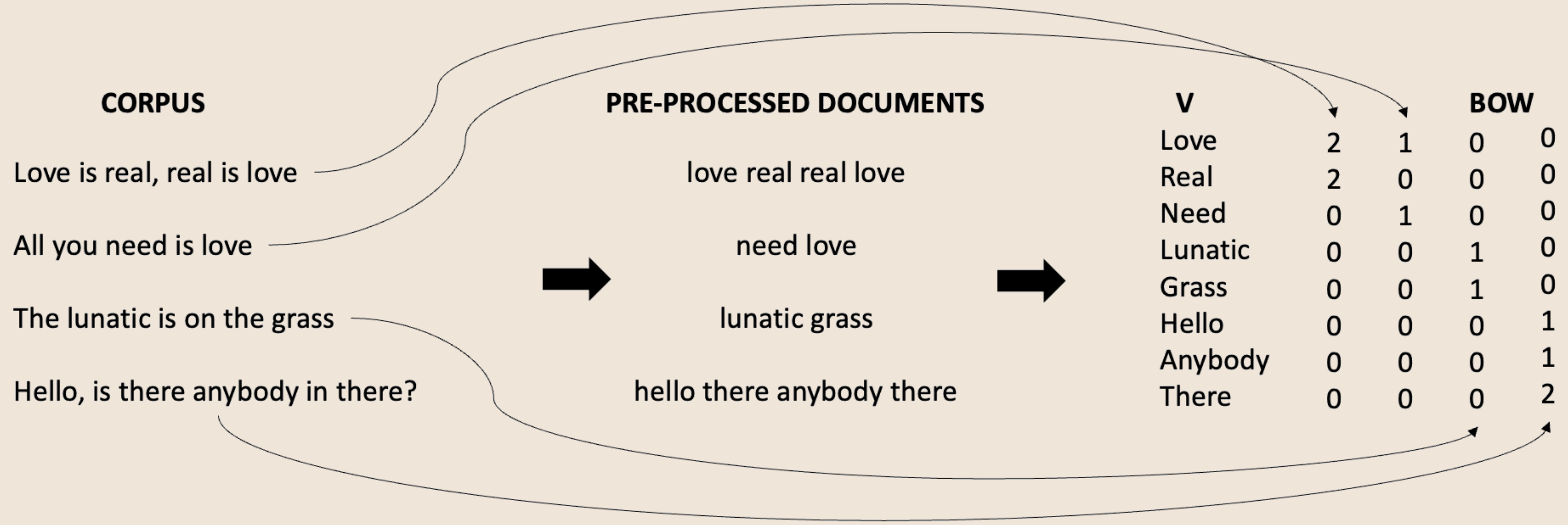
This problem is particularly interesting due to its inherent **high dimensionality**. Every email can be described by an enormous number of potential features, creating a vast feature space.

Our Features: Word Frequencies

The presence or count of every word in a potentially massive vocabulary becomes a feature. If we have a vocabulary of 10k words, we have 10k dimensions.

A feature vector can be very sparse, counting only the words actually appearing in the e-mail.

Each of these elements contributes to forming a high-dimensional vector, making the task of estimating the likelihood function $P(X|Y = k)$ complex without strategic simplifications.



CORPUS

Love is real, real is love

All you need is love

The lunatic is on the grass

Hello, is there anybody in there?

PRE-PROCESSED DOCUMENTS

love real real love

need love

lunatic grass

hello there anybody there

V			BOW	
Love	2	1	0	0
Real	2	0	0	0
Need	0	1	0	0
Lunatic	0	0	1	0
Grass	0	0	1	0
Hello	0	0	0	1
Anybody	0	0	0	1
There	0	0	0	2

The "Ideal" Model for Discrete Data (and its Limitations)

If our features are discrete (e.g., `Offer=Yes`, `Free=No`), we could try to model $P(X|Y)$ by building a **giant contingency table** for every single combination of features.

The Conceptual Approach

This "ideal" model would meticulously track the probability of every specific feature combination for each class, providing a complete picture of $P(X|Y)$.

The Problem: Curse of Dimensionality

If we have 10,000 words (features) in our vocabulary, we would need to estimate the probability for $2^{10,000}$ possible combinations. This is computationally and statistically impossible due to extreme data sparsity.

This massive number highlights why directly modelling $P(X|Y)$ for high-dimensional discrete data is unfeasible without significant simplifications.

> Note that this approach actually works in simple cases, with few features and few levels, but it does not generalise to core complex cases.

The "Ideal" Model for Continuous Data: QDA

For continuous features, we can't use contingency tables. The most flexible "ideal" approach is to model $P(X = \mathbf{x}|Y = k)$ as a **Multivariate Gaussian (Normal) Distribution**:

$$P(X = x|Y = k) = N(\mu_k, \mathbf{\Sigma}_k)$$

Following this formulation, in practice we have to compute a mean vector and a covariance matrix for each class, based on the subset of the data belonging to that class.

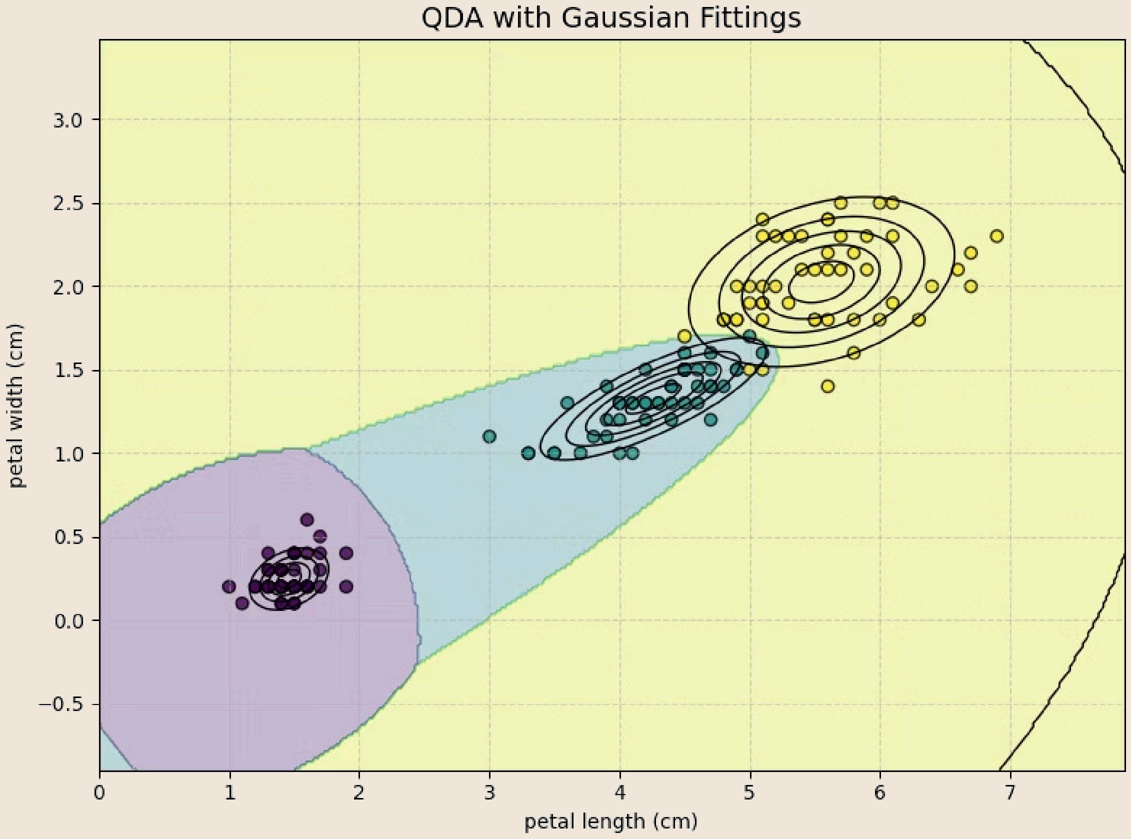
This is the foundation of **Quadratic Discriminant Analysis (QDA)**.

Assumptions

Each class k has its *own* mean vector μ_k and its *own, full* covariance matrix $\mathbf{\Sigma}_k$.

The Computational Probelm

Estimating a full covariance matrix $\mathbf{\Sigma}_k$ for d features requires $\sim \frac{d^2}{2}$ parameters per class. For 100 features, that's $\approx 5,000$ parameters, making it computationally intensive and prone to overfitting, especially with classes with limited data.



The Resulting Boundary

QDA produces a powerful, **curved (quadratic)** decision boundary between classes, reflecting its flexibility in modelling distinct class distributions.

When we have enough data, QDA can lead to good results, but it can easily overfit when we don't have enough data.

Note that each class has its own Gaussian distribution and that distributions can be very different from each other.

The First Simplification: Linear Discriminant Analysis (LDA)

The computational limitations of QDA leads us to our first compromise: **Linear Discriminant Analysis (LDA)**. This approach seeks to simplify the model for greater stability and practicality.

LDA still models $P(X = \mathbf{x}|Y = k)$ as a **Multivariate Gaussian (Normal) Distribution**:

$$P(X = x|Y = k) = N(\mu_k, \mathbf{\Sigma})$$

However, this time, while means μ_k are different, the covariance matrix $\mathbf{\Sigma}$ is assumed to be the same for all classes. This significantly reduces the number of parameters for the estimation of the covariance from $\approx K \times \frac{d^2}{2}$ to $\approx \times \frac{d^2}{2}$.

1

Assumption: Shared Covariance

We simplify the problem by assuming all classes *share* the **same covariance matrix** ($\mathbf{\Sigma}_k = \mathbf{\Sigma}$ for all k). This is a significant reduction in model complexity.

2

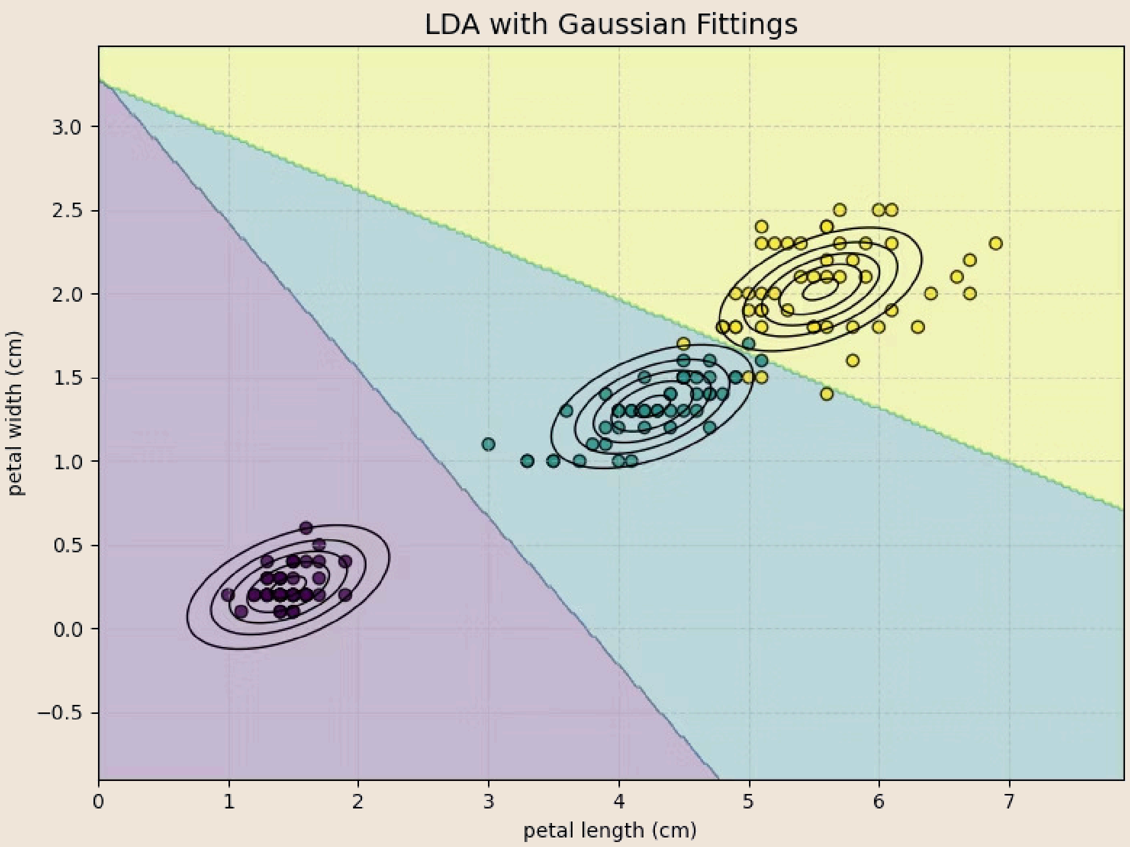
The Benefit: Increased Stability

This allows us to estimate just *one* full $\mathbf{\Sigma}$ matrix, rather than a separate one for each class, greatly enhancing the model's stability, especially with limited data (we will use all data, rather than class-grouped data).

3

The "Nightmare" (Still)

However, we *still* need to estimate a full $\sim \frac{d^2}{2}$ matrix. For very high-dimensional data, like text with $d = 10,000$ features, this remains computationally and statistically impossible.



The Resulting Boundary

This critical assumption forces the mathematical derivations to simplify, resulting in a decision boundary that is always **linear**.

While less flexible than QDA's quadratic boundaries, LDA's linear approach is more robust and less prone to overfitting when data is scarce or dimensionality is high.

Note how now each class has the same "shape" (covariance). This makes the model less flexible and leads to a linear decision boundary.

The Naive Assumption

Naive Bayes makes one powerful, simplifying assumption that solves the dimensionality problem:

> All features X_i are **conditionally independent** given the class Y = k.
>
> $$X_i \perp X_j | Y, \forall i \neq j$$

This leads to:

$$P(X_1, \ldots, X_n | Y) = P(X_1|Y)P(X_2|Y) \ldots P(X_n|Y)$$

Instead of modeling one complex joint distribution, we model $n$simple univariate distributions!

This assumption is called "naive" because it's often violated in practice. Surprisingly, Naive Bayes works remarkably well despite this violation.

Why the Assumption is "Naive"

Spam Classification Example: Consider word counts in emails. The naive assumption claims that **within non-spam emails, the count of one word doesn't influence the count of another word**.

Vacation Email

Words like "trip," "flight," and "luggage" appear together frequently.

Work Email

Words like "meeting," "report," and "deadline" co-occur often.

Clearly, word counts are **not independent**—they're correlated by topic! Yet Naive Bayes still performs excellently in practice because the assumption simplifies computation dramatically while preserving enough signal for accurate classification.

The Simplified MAP Rule

With the conditional independence assumption, our classification rule becomes beautifully simple:

$$f(\mathbf{x}) = \arg_k \max P(\mathbf{x}_1|Y=k)P(\mathbf{x}_2|Y=k)\ldots P(\mathbf{x}_n|Y=k)P(Y=k)$$

Each $P(X_i|Y=k)$ term is now easy to model since X_i is one-dimensional. We choose the distribution type based on our data:

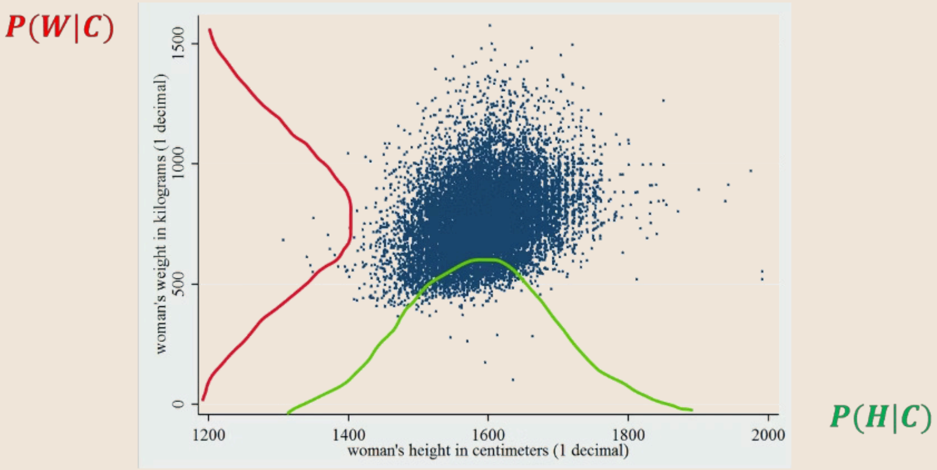
Gaussian Naive Bayes

For continuous features, model each $P(X_i|Y)$ as a univariate Gaussian distribution.

Multinomial Naive Bayes

For discrete count data (like text), model using multinomial distributions.

Gaussian Naive Bayes in Action



Consider classifying biological sex based on height and weight. For each feature and each class, we estimate a simple 1D Gaussian:

- $P(H = h|C = 0) = N(x; \mu_1, \sigma_1)$ - Height distribution for class 0
- $P(W = w|C = 0) = N(x; \mu_2, \sigma_2)$ - Weight distribution for class 0
- $P(H = h|C = 1) = N(x; \mu_3, \sigma_3)$ - Height distribution for class 1
- $P(W = w|C = 1) = N(x; \mu_4, \sigma_4)$ - Weight distribution for class 1

The plot on the left shows the estimation of height and weight distributions for the female class. A similar computation is done for the male class.

Classification rule: Assign (h, w) to class 1 if:

$$P(h|C = 1)P(w|C = 1)P(C = 1) > P(h|C = 0)P(w|C = 0)P(C = 0)$$

Implications of the Naive Assumption

The conditional independence assumption is a strong one, but it offers crucial benefits and imposes specific characteristics on our model:

Diagonal Covariance Matrix

Given the class, the assumption that features X_i are independent implies that the off-diagonal elements of the covariance matrix $\mathbf{\Sigma}_k$ for each class k are zero. This results in a diagonal covariance matrix.

$$\mathbf{\Sigma}_k = \begin{pmatrix} \sigma_{k1}^2 & 0 & \dots & 0 \\ 0 & \sigma_{k2}^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_{kn}^2 \end{pmatrix}$$

Axis-Aligned Distributions

A diagonal covariance matrix means the multivariate Gaussian distributions for each class are "axis-aligned." Their elliptical contours are oriented parallel to the coordinate axes, simplifying calculations but preventing the model from capturing any rotation or correlation between features within a class.

Non-Linear Decision Boundaries

Crucially, while each class's covariance matrix is diagonal, each class can still have a **different** diagonal covariance matrix ($\mathbf{\Sigma}_k \neq \mathbf{\Sigma}_j$). This flexibility allows Gaussian Naive Bayes to produce non-linear decision boundaries, similar in concept to QDA, but with stricter constraints on the shape of the distributions.

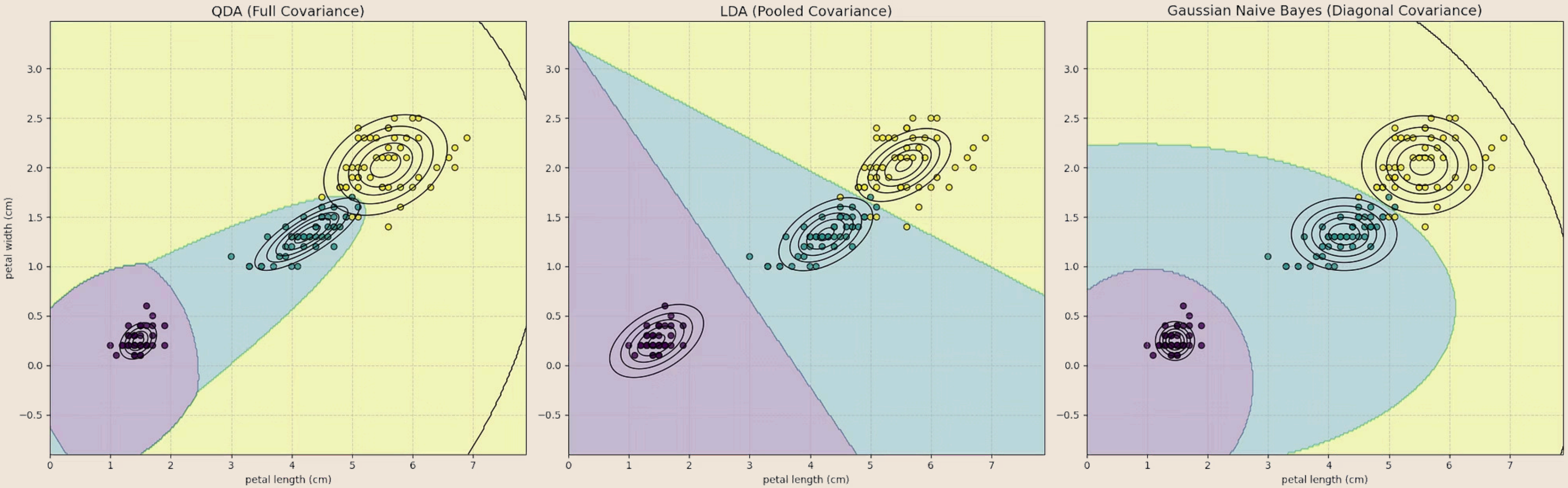
No Intra-Class Correlation

The primary limitation: Naive Bayes explicitly **cannot** model correlations between features **within** the same class. If, for instance, taller individuals in a class tend to also be heavier, Naive Bayes cannot directly capture this relationship, as it treats height and weight as independent given the class.

> Hence, Naive Bayes may not work well when there is high correlation between features.

This trade-off of simplified modeling for the inability to capture intra-class correlations is what defines Naive Bayes.

Naive Bayes Decision Boundaries



This figure perfectly summarizes the three models:

- **QDA (Left):** Most flexible. It learns a separate, *rotated* ellipse (full $\mathbf{\Sigma}_k$) for each class. Its boundaries are quadratic.
- **LDA (Middle):** The compromise. It learns *one* pooled, *rotated* ellipse (full $\mathbf{\Sigma}$) and uses it for all classes. Its boundaries are linear.
- **GNB (Right):** Most simple. It learns a separate, *axis-aligned* ellipse (diagonal $\mathbf{\Sigma}_k$) for each class. Its boundaries are quadratic but can't capture any feature correlation (tilt).

Multinomial Naive Bayes for Text

For text classification with word counts (also called bag-of-words), features are discrete. A document is represented as:

$$\mathbf{x} = (x_1, x_2, x_3, \dots, x_d)$$

where $x_i \in \mathbb{N}$ is the count of word i in the document $\mathbf{x}$ and $d = |V|$ is the size of the vocabulary V.

Hence, the probability of a given document $\mathbf{x}$ to belong to class k can be expressed as:

$$P(\mathbf{x}|Y = k) = P(x_1, x_2, \dots, x_d|Y = k)$$

We can model with probability with a **multinomial distribution**, treating document generation as repeatedly sampling words from a vocabulary:

Remember that **the multinomial distribution models the probability of obtaining exactly** $(n_1, \dots, n_k)$ **occurrences with** $n = \sum_i n_i$ **for each of** k **possible outcomes in a sequence of** n **independent experiments which follow categorical distributions with probabilities** $p_1, \dots, p_k$.

In our case:

- **Individual experiments**: the process of generating a single word
- k **possible outcomes**: words in the vocabulary
- n **independent experiments**: the words in a document $n = \sum_i x_i$

We can assume that this process follows a categorical distribution with some words being more common than others. These experiments are *independent* because of the naïve assumption (given a class, word counts are independent).

Fitting the Multinomial Distribution to the Data

Using the analytical form of the multinomial distribution, we can write:

$$P(\mathbf{x}|Y=k) = \frac{(\sum_i x_i)!}{x_1! \dots x_d!} p_{k1}^{x_1} \cdot p_{k2}^{x_2} \cdot \dots \cdot p_{\mathrm{kn}}^{x_d}$$

In the expression above:

- $n = \sum_i x_i$ is the total number of experiments (words in the document)
- x_i is the count of word i in the current document $\mathbf{x}$
- p_{ki} is the probability of observing word i if we randomly pick a word from a document of class $Y = k$

In practice, to the use the formula above, we need to estimate the parameters:

$$p_{ki} \qquad k = 1, \dots, |V|, k \in \{1, \dots, M\}$$

where $|V|$ is the size of the vocabulary and $\{1, \dots, M\}$ are the M classes.

Once we have all these parameters, we obtain our classifier as follows:

$$h\left(x_1,\ \dots, x_n\right) = \arg_k \max P\left(Y = k\right) p_{k1}^{x_1} \cdot p_{k2}^{x_2} \cdot \dots \cdot p_{\mathrm{kn}}^{x_n}$$

Where we can ignore the big factorial term $\frac{(\sum_i x_i)!}{x_1! \dots x_n!}$ as it is independent of k.

Estimating Word Probabilities

We estimate the probability of word i for class k by counting:

$$p_{ki} = \frac{\text{Total count of word } i \text{ in all documents of class } k}{\text{Total count of *all words* in all documents of class } k}$$

Using the Iverson bracket notation from your slides (where $[y^{(j)} = k]$ is 1 if true, 0 otherwise):

$$p_{ki} = \frac{\sum_j [y^{(j)} = k] x_i^{(j)}}{\sum_{l=1}^d \left(\sum_j [y^{(j)} = k] x_l^{(j)} \right)}$$

- **Numerator:** Sums the counts of word i **only** in documents j from class k.
- **Denominator:** Sums the counts of **all** words ($l = 1$ to d) in **all** documents j from class k.

The prior probability is estimated as:

$$P(Y = k) = \frac{\sum_j [y^{(j)} = k]}{N}$$

Where N is the number of observations in the training set. Alternatively, we can assume uniform priors and set:

$$P(Y = k) = \frac{1}{M}$$

Addressing the Zero-Probability Problem

A critical issue in Multinomial Naive Bayes arises when classifying a document containing a word not present in a specific class's training data. This leads to the "Zero-Probability Problem."

The Challenge: Zero Probabilities

Consider a new email containing the word "crypto." If our training data for the 'Spam' class **never** included this word, its estimated probability $p_{k,\text{'crypto'}}$ for that class would be 0.

$$h(\mathbf{x}) \propto P(Y = \text{Spam}) \cdot \ldots \cdot p_{k,\text{'crypto'}}^{x_{\text{crypto}}} \cdot \ldots = P(Y = \text{Spam}) \cdot \ldots \cdot 0 \cdot \ldots = 0$$

Because of this single zero, the entire probability for the 'Spam' class becomes zero, effectively disabling classification for that instance and rendering the model ineffective.

The Solution: Laplace (Add-One) Smoothing

To prevent such occurrences, we use **Laplace Smoothing**. This technique adds a "pseudo-count" (typically 1) to each word's count for every class. This ensures that even unobserved words have a non-zero, albeit small, probability.

$$p_{ki} = \frac{\left(\sum_j [y^{(j)} = k] x_i^{(j)}\right) + 1}{\left(\sum_{l=1}^{d} \sum_j [y^{(j)} = k] x_l^{(j)}\right) + |V|}$$

- The `+ 1` in the numerator means we "pretend" we've seen each word at least once.
- The `+ |V|` (where $|V|$ is the vocabulary size) in the denominator corrects for these added pseudo-counts, ensuring the probabilities for each class still sum to 1.

> In practice, words which were not present in the training corpus for a given class, get assigned a uniform probability of $\frac{1}{|V|}$.

Addressing Arithmetic Underflow

Our formulation involves multiplying small (< 1) probability values $p_{k1}^{x_1} \cdot p_{k2}^{x_2} \cdot \ldots \cdot p_{\text{kn}}^{x_n}$. This can lead to "arithmetic underflow." This occurs when the product becomes so tiny that computers round it to zero, rendering classification invalid.

The solution is to use **log-probabilities.** Since the logarithm is a monotonic function, maximizing `log P(X|Y)` is equivalent to maximizing `P(X|Y)`. This transforms unstable multiplications into stable sums, practically transitioning from this:

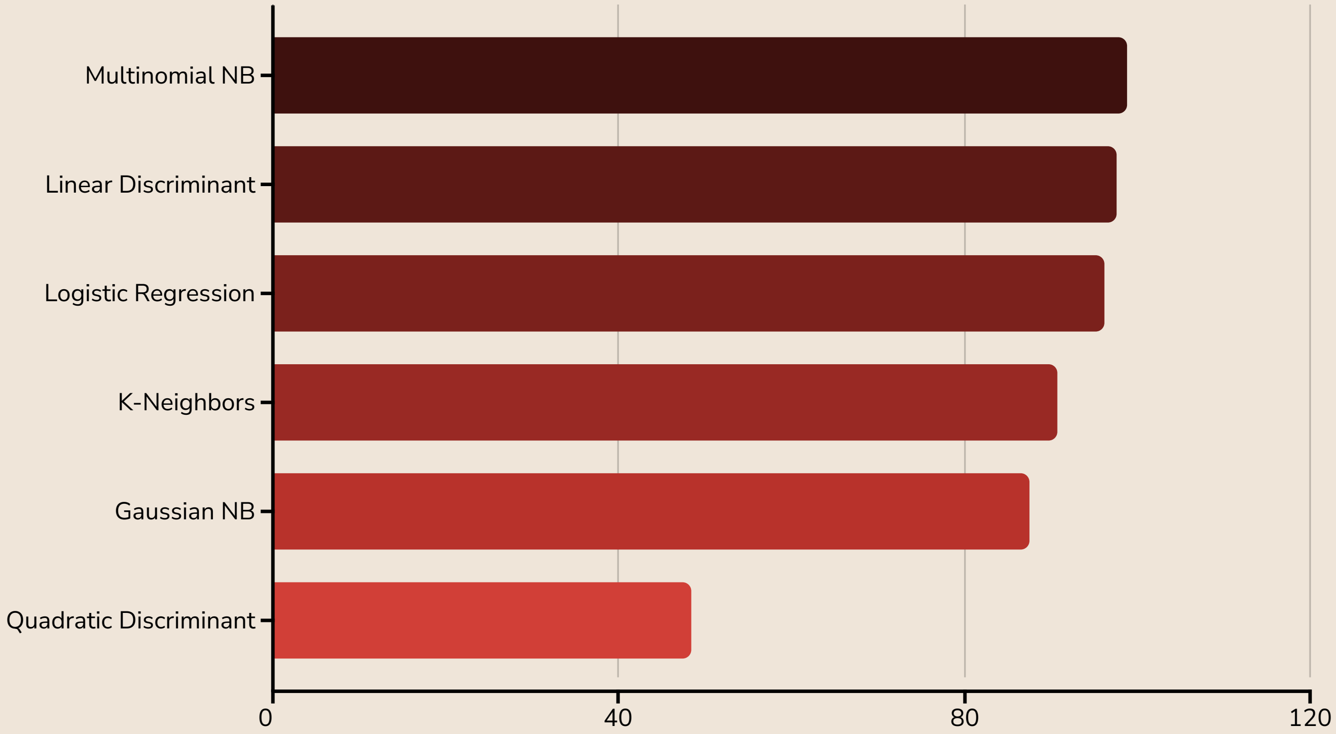
$$h\left(\mathbf{x}\right) = \arg\max_k P\left(Y = k\right) \prod_{i=1}^{d} p_{ki}^{x_i}$$

To this:

$$h(\mathbf{x}) = \arg\max_k \left(\log P(Y = k) + \sum_{i=1}^{d} x_i \cdot \log p_{ki}\right)$$

This log-transformed rule is numerically stable, extremely fast, and forms the foundation for practical Naive Bayes implementations in libraries like `scikit-learn`.

Laboratory: Comparison of Models



98.74%

Multinomial NB

Winner! Fast (0.18s) and accurate

45.11

LDA Time (sec)

Accurate but extremely slow

48.33%

QDA Accuracy

Complete failure on high-dimensional text

Conclusions and Next Steps

We Have Explored:

- Generative classifier
- The MAP principle
- The "ideal" solution for discrete data (big contingency table)
- The "ideal" solution for continuous data (QDA)
- The LDA simplification and its limits
- The Naive Assumption
- Gaussian Naive Bayes
- Multinomial Naive Bayes

> In the next lectures, we will look at closer and data representation and clustering

References

- Naïve Bayes Classifier: https://en.wikipedia.org/wiki/Naive_Bayes_classifier;
- https://scikit-learn.org/stable/modules/lda_qda.html#lda-qda
- Section 4.4 of [1]

[1] James, Gareth Gareth Michael. An introduction to statistical learning: with applications in Python, 2023.https://www.statlearning.com