Naive Bayes and Kernel Naive Bayes Classification

Cheng Peng

West Chester University

Contents

1 Introduction

Naive Bayes is a probabilistic machine learning algorithm based on Bayes' Theorem, commonly used for classification tasks. It assumes that the **features are independent of each other given the class label** - a condition known as **naive** independence. In mathematical statistics, we introduce Bayes Theorem to assess the relationship between two events.

Let A and B are two events, Bayes' Theorem states:

$$P(C|A) = \frac{P(A|C)P(C)}{P(A)}$$

Here:

- $P(C|A)$ is the **posterior probability** of event C given event A.
- $P(A|C)$ is the likelihood of event A given event C.
- $P(C)$ is the prior probability of event C.
- $P(A)$ is the probability of event A.

Steps in Naive Bayes

1. Compute prior probabilities for each event.
2. Calculate the likelihood of each feature for each event.
3. Compute the posterior probabilities for each event.
4. Assign the event label with the highest posterior probability.

2 What is Naive Bayes?

Naive Bayes is a probabilistic classification algorithm based on **Bayes' Theorem**, with a key assumption that **features are conditionally independent given the class label**. Despite its simplicity, it performs well in many real-world applications like spam detection, sentiment analysis, and medical diagnosis.

The key components of Naive Bayes are:

1. Bayes' Theorem

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

- $P(Y|X)$: Posterior probability (probability of class Y given features X.
- $P(X|Y)$: Likelihood (probability of features X given class Y
- $P(Y)$: Prior probability (probability of class Y.
- $P(X)$: Marginal probability (normalizing constant, often ignored in classification)

2. Naive Assumption

The crucial assumption of **naive Bayes** is that each feature contributes independently to the probability. That is, features are conditionally independent of the response Y.

$$P(X|Y) = P([(X_1 = x_1) \cap (X_2 = x_2) \cap \cdots \cap (X_k = x_k)]|Y)$$

$$= P[X_1 = x_1|Y] \times P[X_2 = x_2|Y] \times \cdots \times P[X_k = x_k|Y] = \prod_{j=1}^{k} P[X_j = x_j|Y]$$

3. Training Process

Naive Bayes is fast to train because it just needs to compute:

- The prior probabilities $P(C)$
- The conditional probabilities $P(x_i|C)$

These are usually estimated using frequency counts from the training data.

4.Prediction Rule

The model predicts the class $Y = C$ with the highest posterior probability. Since the denominator of Bayes THeorem is given, the predicted value of Y is given by

$$\hat{Y} = \arg\max_{Y} P(Y) \times \prod_{i=1}^{k} P(X_i = x_i|Y)$$

$\hat{Y}$ is the class label such as $\hat{Y} = C_k$.

Since *Naive Bayes* estimates class probabilities. It relies on counting frequencies (for discrete data) or Gaussian distributions (for continuous features), but the output is always a class label, not a continuous value (i.e., the class label corresponding to the highest probability). Therefore, *Naive Bayes* cannot do regression with continuous response.

3 Types of Naive Bayes Classifiers

There are three different types of naive Bayes algorithms

- **Gaussian Naive Bayes**: For continuous data (assumes normal distribution).
- **Multinomial Naive Bayes**: Often used for document classification (e.g. text data).
- **Bernoulli Naive Bayes**: For binary features (e.g. word present or not in text).

"In the next few subsections, we briefly discuss the different types of Naive Bayes with some toy examples to illustrate the concepts.

3.1 Gaussian Naive Bayes

Gaussian Naive Bayes (GNB) is a classification algorithm based on Bayes' theorem, assuming feature independence and that continuous features follow a Gaussian (normal) distribution. This makes it particularly effective when the data approximately satisfies these assumptions.

Bayes' theorem for classification is expressed as:

$$P(C|X) = \frac{P(X|C)P(C)}{P(X)}$$

Where:

- $P(C|X)$ is the posterior probability of class C given feature X.
- $P(X|C)$ is the likelihood of feature X given class C, modeled as a Gaussian distribution.
- $P(C)$ is the prior probability of class C
- $P(X)$ is the evidence (normalization factor).

For Gaussian Naive Bayes, the likelihood $P(X_i = x|Y = y)$ for a feature X_i is modeled by

$$P(X_i = x|Y = y) = \frac{1}{\sqrt{2\pi}\sigma_{iy}} \exp\left[-\frac{(x-\mu_{iy})^2}{2\sigma_{iy}^2}\right]$$

Where μ_{iy} is the mean of the feature X_i for class $Y = y$ and σ_{iy}^2 is the variance of the feature X_i for class $Y = y$.

Next, we will discuss two cases: continuous features only and the mixture of both continuous and categorical features.

3.1.1 All Continuous Features

The key steps for implementing Gaussian Naive Bayes are summarized below.

- **Estimate Class Priors** $P(Y = y)$ - we can estimate the prior probability (class probability) using the relative frequencies of each class label.
- **Estimate Gaussian Parameters (mean μ and variance σ^2)** for each feature per class. We can simply use the sample mean and sample variance of each feature variable within each class label of Y.
- **Compute Likelihood** $P(X_i|Y = y)$ using the Gaussian PDF with the sample mean and sample variance.
- **Apply Naive Bayes** (assume feature independence) to predict the class.

An Example with a Toy Data Set

Suppose we have a binary classification problem with two features X_1 and X_2. A small toy data set is given by

X_1	X_2	Y (Class)
1.0	1.0	0
1.5	1.2	0
2.0	1.8	0
3.0	2.5	1
3.5	3.0	1
4.0	3.5	1

New sample to classify:

$x = (X_1 = 2.2, X_2 = 2.0)$

Step 1: Compute Class Priors

Total samples: 6. We use relative frequency to estimate the class probabilities as: * $P(Y = 0) = 0.5$ * $P(Y = 1) = 0.5$

Step 2: Estimate Gaussian Parameters per Class

For each class Y, we compute Mean (μ) and variance (σ^2) for each feature: We summariz

Samples	Class (Y)	Feature	Mean (μ)	Variance (σ^2)
[1.0,1.5,2.0]	0	X_1	1.5	0.1667
[1.0,1.2,1.8]	0	X_2	1.333	0.1157
[3.0,3.5,4.0]	1	X_1	3.5	0.1667
[2.5,3.0,3.5]	1	X_2	3.0	0.1667

Step 3: Compute Likelihoods Using Gaussian PDF

The Gaussian probability density function (PDF) is

$$P(X_i = x_i|Y = y) = \frac{1}{\sqrt{2\pi\sigma_{iy}^2}} \exp\left[-\frac{(x_i - \mu_{iy})^2}{2\sigma_{iy}^2}\right].$$

We use the above density to calculate the likelihood of observing the new sample: $\mathbf{x} = (2.2, 2.0)$:

$$P(X_1 = 2.2|Y = 0) = \frac{1}{\sqrt{2\pi(0.1667)}} \exp\left[-\frac{(2.2 - 1.5)^2}{2 \times 0.1667}\right] \approx 0.2246.$$
$$P(X_1 = 2.0|Y = 0) = \frac{1}{\sqrt{2\pi(0.1157)}} \exp\left[-\frac{(2.0 - 1.333)^2}{2 \times 0.1157}\right] \approx 0.1711.$$

Under conditional independence (i.e., feature independence), the total likelihood for $Y = 0$ is given by

$$P(\mathbf{x}|Y = 0) = P(X_1 = 2.2|Y = 0) \times P(X_1 = 2.0|Y = 0) = 0.2246 \times 0.1711 \approx 0.0384.$$

$$P(X_1 = 2.2|Y = 1) = \frac{1}{\sqrt{2\pi(0.1667)}} \exp\left[-\frac{(2.2 - 3.5)^2}{2 \times 0.1667}\right] \approx 0.0061.$$

$$P(X_1 = 2.0|Y = 1) = \frac{1}{\sqrt{2\pi(0.1667)}} \exp\left[-\frac{(2.2 - 3.0)^2}{2 \times 0.1667}\right] \approx 0.0487.$$

the total likelihood for $Y = 1$ is given by

$$P(\mathbf{x}|Y = 1) = P(X_1 = 2.2|Y = 1) \times P(X_1 = 2.0|Y = 1) = 0.0061 \times 0.0487 \approx 0.0003.$$

Step 4: Compute Posterior Probabilities

Using Bayes' Theorem (ignoring denominator since it cancels out):

$$P(Y = 0|\mathbf{x}) = \frac{P(Y = 0 \cap \curvearrowleft)}{P(\mathbf{x})} \propto P(Y = 0) \times P(\mathbf{x}|Y = 0) = 0.5 \times 0.0384 = 0.0192.$$

$$P(Y = 1|\mathbf{x}) = \frac{P(Y = 1 \cap \curvearrowleft)}{P(\mathbf{x})} \propto P(Y = 1) \times P(\mathbf{x}|Y = 1) = 0.5 \times 0.0003 = 0.00015.$$

Normalizing the posteriors, we have

$$P(Y = 0|\mathbf{x}) = \frac{0.0192}{0.0192 + 0.00015} \approx 0.992.$$

$$P(Y = 1|\mathbf{x}) = \frac{0.00015}{0.0192 + 0.00015} \approx 0.008.$$

Step 5: Prediction

Since $P(Y = 0|\mathbf{x}) > P(Y = 1|\mathbf{x})$, we classify $\mathbf{x} = (2.2, 2.0)$ as class $\mathbf{Y = 0}$.

Manufacturer (Cat)	Mileage (Cont)	Reliable (Class)

3.1.2 Mixed Features

Since **naive Bayes** assumes conditional independence of features, we can use the same steps in the above case of two continuous features. Next, we use the following toy data to explain the steps of **Naive Bayes**.

Manufacturer (Cat)	Mileage (Cont)	Reliable (Class)
A	50	Yes
A	60	No
B	80	No
B	70	Yes
C	90	No
C	40	Yes

Sample for prediction: Manufacture = B and Mileage = 75.

Step 1: Compute Class Priors

Total samples: 6. We use relative frequency to estimate the class probabilities as:

- $P(Reliable = Yes) = 3/6 = 0.5$
- $P(Reliable = 1No) = 3/6 = 0.5$

Step 2: Estimate Parameters per Feature Type

Categorical Feature: Modeled using relative frequency approximation:

Class	Manufacturer A	Manufacturer B	Manufacturer C
Yes	1/3	1/3	1/3
No	1/3	1/3	1/3

To avoid zero probabilities, we can employ the Laplace smoothing by adding 1 to each count.

Continuous Normal Features: Modeled using Gaussian Naive Bayes (mean and variance).

Class	Sample	Mean (μ)	Variance (σ^2)
Yes	[50, 70, 40]	53.33	155.56
No	[60, 80, 90]	76.67	155.56

Step 3: Compute Likelihoods for New Sample

Recall New sample: `Manufacturer = B, Mileage = 75`

- **Categorical likelihood**
  - P(Manufacturer = B | Reliable = Yes)= 1/3
  - P(Manufacturer = B | Reliable = No) = 1/3
- **Continuous likelihood** (Gaussian PDF)

$$P(\text{Milage} = 75|\text{ Yes}) = \frac{1}{\sqrt{2\pi \times 155.56}} \exp\left[-\frac{(75-53.33)^2}{2 \times 155.53} = \right] = 0.0071$$

$$P(\text{Milage} = 75|\text{ No}) = \frac{1}{\sqrt{2\pi \times 155.56}} \exp\left[-\frac{(75-76.67)^2}{2 \times 155.53} = \right] = 0.03172$$

Step 4: Combine Likelihoods & Predict

$$P(Yes|B, 75) \propto P(Yes) \times P(B|Yes) \times P(75|Yes) = 0.5 \times (1/3) \times 0.03172 \approx 0.001183333$$

$$P(No|B, 75) \propto P(No) \times P(B|No) \times P(75|No) = 0.5 \times (1/3) \times 0.0071 \approx 0.005286667$$

Prediction: Since $P(No|B, 75) > P(Yes|B, 75)$, classify as **No** (unreliable).

3.1.3 Multi-category Response

The calculation for multicategory Naive Bayes (K > 2 classes) follows the same steps as the two-category (binary) case but extends to all classes independently. The key difference is the number of posterior probabilities computed (K instead of 2).

The implementation steps are also identical:

- Estimate class priors $P(Y = y_k)$ for each class y_k.
- Compute feature likelihoods $P(X_i|Y = y_k)$:
  - For continuous features: Gaussian distribution (mean/variance per class).
  - For categorical features: Frequency counts (with Laplace smoothing).
- Calculate posterior probabilities for each class: $P(Y = y_k|X) \propto P(Y = y_k) \times \prod_{i=1}^{n} P(X_i|Y = y_k)$
- Predict the class with the highest posterior:

$$\hat{Y} = \arg\max_{Y} P(Y) \times \prod_{i=1}^{k} P(X_i = x_i|Y).$$

Limitations of Naive Bayes

- Small datasets: With many classes, some $P(X_i|Y = y_k)$ estimates may be unreliable.
- Correlated features: Independence assumption harms performance more for K > 2.
- Non-Gaussian features: Requires kernel density estimation or other variants.

Multicategory Naive Bayes generalizes trivially from the binary case by treating each class as a separate "one-vs-rest" problem. The core math remains unchanged—only the number of comparisons increases. This makes it a versatile tool for classification tasks with any number of classes.

3.2 Multinomial Naive Bayes

Multinomial Naive Bayes is commonly used for text classification or discrete data where features represent counts or frequencies.

Example 1: Text Classification (Spam vs. Ham)

Assume the Training Data is given below:

Email	Label
"free money now"	Spam
"hello world"	Ham

Email	Label
“free lottery”	Spam
“hello friend”	Ham

Test Email: Predict “free world”: Spam or Ham?

Step 1: Preprocess & Compute Word Frequencies

Vocabulary: `["free", "money", "now", "hello", "world", "lottery", "friend"]`

Count words per class and obtain the following data table.

Word	Spam Count	Ham Count
free	2	0
money	1	0
now	1	0
hello	0	2
world	0	1
lottery	1	0
friend	0	1

- Total words in Spam = 4
- Total words in Ham = 4

Step 2: Compute Probabilities (with Laplace Smoothing, α =1)

Note that P(Spam) = P(Ham) = 2/4 = 0.5. Based on conditional (feature) independence, we have

$$P(\text{free world}|\text{Spam}) = P(\text{free}|\text{Spam}) \times P(\text{ world}|\text{Spam}),$$
$$= \frac{2+1}{4+7} \times \frac{0+1}{4+7} = \frac{3}{11} \times \frac{1}{11} = \frac{3}{121}.$$

$$P(\text{free world}|\text{Ham}) = P(\text{free}|\text{Ham}) \times P(\text{ world}|\text{Ham}).$$

$$= \frac{0+1}{4+7} \times \frac{1+1}{4+7} = \frac{1}{11} \times \frac{2}{11} = \frac{2}{121}.$$

Therefore, the joint probabilities for prediction are given, respectively, by:

$$P(\text{Free World } \cap \text{ Spam}) = P(\text{Spam}) \times P(\text{Free World } | \text{ Spam}) = 0.5 \times \frac{3}{121} = \frac{3}{242}.$$

and

$$P(\text{Free World } \cap \text{ Ham}) = P(\text{Ham}) \times P(\text{Free World } | \text{ Ham}) = 0.5 \times \frac{2}{121} = \frac{2}{242}.$$

Prediction: since $P(\text{Free World } \cap \text{ Spam}) = 3/242 > P(\text{Free World } \cap \text{ Ham}) = 2/242$, **Free World** is classified as a **Spam** email.

Remark 1: The above decision is the same as the prediction rule for the general **Naive Bayes**:

$$\hat{Y} = \arg\max_{Y} P(Y) \times \prod_{i=1}^{k} P(X_i = x_i | Y)$$

$$= \max\{P(\text{Free World } \cap \text{ Spam}), P(\text{Free World } \cap \text{ Ham})\} = \max\{\frac{3}{242}, \frac{2}{242}\} \rightarrow \text{ Spam.}$$

3.3 Binomial Naive Bayes

Bernoulli Naive Bayes (BNB), sometimes called *B**inomial Naive Bayes**, is a variant of the Naive Bayes algorithm designed for binary feature data (i.e., features that take values 0 or 1). Unlike **Multinomial Naive Bayes (MNB)**, which works with word counts, BNB only considers whether a feature (e.g., a word) is present (1) or absent (0) in a document.

The key Characteristics of Bernoulli Naive Bayes are

- Binary Features:
  - Each feature is treated as a binary variable (e.g., "Does this word appear in the document? Yes/No").
  - Ignores how many times a word appears (unlike MNB, which counts frequencies).
- Probability Estimation:
  - Computes the probability of a feature being present (1) or absent (0) in each class.
  - Uses Laplace smoothing to avoid zero probabilities for unseen features.
- Suitable For:
  - Short text classification (e.g., spam detection, sentiment analysis).
  - Problems where the presence/absence of a feature is more important than frequency.

A Toy Example: Problem: Classify emails as "spam" or "not spam" based on the presence/absence of certain words.

Step 1: Creating Training Data (Binary Features) based on the keywords in email subjects. Each keyword defines a variable with a value of 0 (non-presence) or 1 (presence). Assume that the keywords used for defining binary features are: `free, money, urgent, meeting`

Email	"free"	"money"	"urgent"	"meeting"	Class
1	1	1	0	0	Spam
2	1	0	1	0	Spam
3	0	0	0	1	Not Spam
4	0	0	1	1	Not Spam
5	1	1	1	0	Spam

Step 1: Calculate Class Priors

- Total emails = 5
- Spam emails = 3
- Not spam emails = 2
- P(Spam) = 3/5 = 0.6
- P(Not Spam) = 2/5 = 0.4

Step 2: Calculate Feature Probabilities

For each feature (word) and each class, we calculate:

- P(feature=1 | class)

- P(feature=0 | class) = 1 - P(feature=1 | class)

We'll use Laplace smoothing (**add 1 to numerator, 2 to denominator since binary features**) to avoid zero probabilities.

(1) Spam Case

Feature	# 1s in Spam	P(feature=1 | Spam)
free	3	(3+1)/(3+2) = 4/5 = 0.8
money	2	(2+1)/(3+2) = 3/5 = 0.6
urgent	2	(2+1)/(3+2) = 3/5 = 0.6
meeting	0	(0+1)/(3+2) = 1/5 = 0.2

(2) Non-Spam Case

Feature	# 1s in Not Spam	P(feature=1 | Not Spam)
free	0	(0+1)/(2+2) = 1/4 = 0.25
money	0	(0+1)/(2+2) = 1/4 = 0.25
urgent	1	(1+1)/(2+2) = 2/4 = 0.5
meeting	2	(2+1)/(2+2) = 3/4 = 0.75

Suppose the new email is: free=1, money=0, urgent=0, meeting=1

Using the Naive Bayes prediction formula

$$P(C|x) \propto P(C) \prod_{i=1}^{n} P(x_i|C)^{x_i} [1 - P(x_i|C)]^{1-x_i}$$

Recall that the new email subject line keywords are :

- x_1 = free = 1
- x_2 = money = 0
- x_3 = urgent = 0
- x_4 = meeting = 1

With the above values of the binary features, we calculate the posterior probabilities below:

$$P(\text{Spam}|\text{x}) \propto 0.6 \times 0.8^1 \times (1 - 0.6)^1 \times (1 - 0.6)^1 \times 0.2^1 = 0.01536.$$

$$P(\text{Not Spam}|\text{x}) \propto 0.4 \times 0.25^1 \times (1 - 0.25)^1 \times (1 - 0.5)^1 \times 0.75^1 = 0.028125.$$

Prediction - Since P(Not Spam | x) > P(Spam | x), the new email is classified as: **Not Spam**.

4 Special Nature of Naive Bayes

Naive Bayes differs from many other classification algorithms because it does not rely on a loss function or optimization procedure to estimate its parameters. This is a fundamental distinction that sets it apart from models like logistic regression, support vector machines (SVMs), decision trees, and neural networks.

Naive Bayes is a generative probabilistic model. It learns the joint probability distribution of the features and the target variable using **Bayes' theorem**, under the (naive) assumption that all features are conditionally independent given the class. Rather than optimizing a function to find the best-fitting parameters, Naive Bayes estimates probabilities directly from the training data using simple frequency counts or likelihoods, sometimes with smoothing techniques like Laplace smoothing to handle zero probabilities. The model then uses these probabilities to compute the posterior probability for each class and selects the class with the highest posterior as the prediction.

In contrast, most other common classification algorithms are discriminative models that directly learn the boundary between classes. For example, logistic regression models the conditional probability $P(y|x)$ using a *sigmoid function* and finds the parameters (weights) that minimize a loss function, such as **cross-entropy loss**, typically using **gradient descent**. Similarly, **SVMs** optimize a hinge loss to find a hyperplane that best separates classes with the maximum margin. **Neural networks** also rely heavily on loss functions to back-propagate errors and update weights.

Because **Naive Bayes** bypasses this optimization step, it is extremely **fast and scalable**, even with high-dimensional data. It's especially useful for tasks like spam detection, sentiment analysis, and document classification, where **conditional independence assumptions** often hold reasonably well. However, its simplicity can be a limitation when feature dependencies are strong or when complex decision boundaries are needed—situations where optimization-based models may perform better.

Naive Bayes is transparent and interpretable. The probabilities it computes can be easily inspected and explained, making it a strong candidate in settings where model interpretability is essential. Its special blend of speed, simplicity, and probabilistic foundation makes Naive Bayes a uniquely powerful tool in the data scientist's toolbox, especially for baseline modeling and tasks involving high-dimensional sparse data.

In short, Naive Bayes learns by counting, not by optimizing—making it both elegant in theory and efficient in practice.

5 Guassina NB Case Study

In this case study, we use Gaussian Naive Bayes to predict diabetes based on the **Pima Indian Diabetes Data**.

As we know the **Pima Indian Diabetes** dataset contains the following features:

- **Pregnancies**: Number of times pregnant
- **Glucose**: Plasma glucose concentration (2 hours in oral glucose tolerance test)
- **BloodPressure**: Diastolic blood pressure (mm Hg)
- **SkinThickness**: Triceps skin fold thickness (mm)
- **Insulin**: 2-Hour serum insulin (mu U/ml)
- **BMI**: Body mass index (weight in kg/(height in m)^2)
- **DiabetesPedigreeFunction**: Diabetes pedigree function
- **Age**: Age in years
- **Outcome**: Class variable (0 - non-diabetic, 1 - diabetic)

We will divide the analysis into several steps:

1. Data Preparation

This stage requires handling missing values, as Naive Bayes relies on completely observed data to calculate empirical probabilities and evaluate likelihoods. Since Gaussian Naive Bayes assumes that continuous features are normally distributed, it is not necessary to scale the continuous features, though doing so can be helpful.

```r
# Load required libraries
# library(caret)
# library(naivebayes)
# library(pROC)

# Load the dataset
data("PimaIndiansDiabetes2", package = "mlbench")
diabetes.data <- na.omit(PimaIndiansDiabetes2) # Remove missing values

# Split data into training and testing sets (70-30 split)
set.seed(123)
N = length(diabetes.data$diabetes)
trainIndex <- sample(1:N, size = floor(0.7*N), replace = FALSE)
train.data <- diabetes.data[trainIndex, ]
test.data <- diabetes.data[-trainIndex, ]

# batch rescaling. Could also consider min-max scaling
preProc <- preProcess(train.data[, -9], method = c("center", "scale"))
train.data[, -9] <- predict(preProc, train.data[, -9])
test.data[, -9] <- predict(preProc, test.data[, -9])
```

2. Model Building

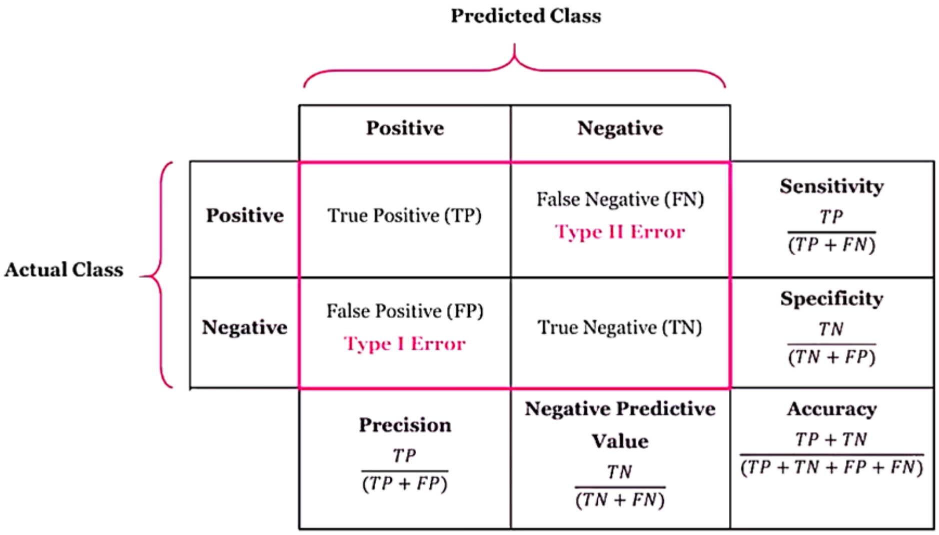
The R library **naivebayes** can perform various naive Bayes classification tasks. By default, feature variables are assumed to be normally distributed. It can also perform kernel naive Bayes to handle non-normal continuous features. In this case study, we use the default setting. Any transformations that can convert the continuous features closer to normal distribution will improve the performance of the naive Bayes classification.

```r
# Build Gaussian Naive Bayes model using naive_bayes() in library {naivebayes}
gnb.model <- naive_bayes(diabetes ~ ., data = train.data)

# View model summary: estimated means and variance of continuous feature
# gnb.model
```

3. Prediction and Performance Measures

We can see from the above toy examples that Naive Bayes calculates the posterior first and then outputs the label with a higher posterior probability. If posterior probabilities of associated binary labels are normalized (sum to 1.0), it is essentially the same as choosing the default threshold of 0.5. The method is not restricted to binary classification problems.

All metrics used in other classification algorithms can be used to assess the performance of Naive Bayes algorithms. To help read the output of `naive_bayes`, we summarize the definitions of commonly used performance metrics based on the following confusion matrix.



		Predicted Class		
		Positive	Negative	
Actual Class	Positive	True Positive (TP)	False Negative (FN) Type II Error	Sensitivity $\frac{TP}{(TP+FN)}$
	Negative	False Positive (FP) Type I Error	True Negative (TN)	Specificity $\frac{TN}{(TN+FP)}$
		Precision $\frac{TP}{(TP+FP)}$	Negative Predictive Value $\frac{TN}{(TN+FN)}$	Accuracy $\frac{TP+TN}{(TP+TN+FP+FN)}$

Term	Also Known As	Formula	Use Case
Sensitivity	Recall, TPR	TP / (TP + FN)	Detecting positives correctly
Specificity	TNR	TN / (TN + FP)	Avoiding false negatives
Accuracy	-	(TP + TN)/ Total	-
Pos Pred Value	Precision, PPV	TP / (TP + FP)	Reliability of positive predictions
Neg Pred Value	NPV	TN / (TN + FN)	Reliability of negative predictions
Prevalence	-	(TP + FN) / Total	How common is the positive class?
Detection Rate	Sensitivity	Same as Sensitivity	-
Detection Prevalence	-	(TP + FP) / Total	How often does the model predict positive?
Balanced Accuracy	-	(Sensitivity + Specificity) / 2	Handling class imbalance
F1 Score	Harmonic Mean	2 x sen x spe/(sen + spe)	good for rare event prediction

We next report the three performance metrics: Accuracy, Precision, Recall, and F1 Score for the Naive Bayes algorithm.

```r
# Make predictions on test set
predictions <- predict(gnb.model, test.data)

true.labels <- factor(test.data$diabetes)
predictions <- factor(predictions, levels = levels(true.labels))

# Confusion matrix
conf.matrix <- confusionMatrix(predictions, test.data$diabetes, positive = "pos")
#print(conf.matrix)

# Get predicted probabilities for class "pos"
```

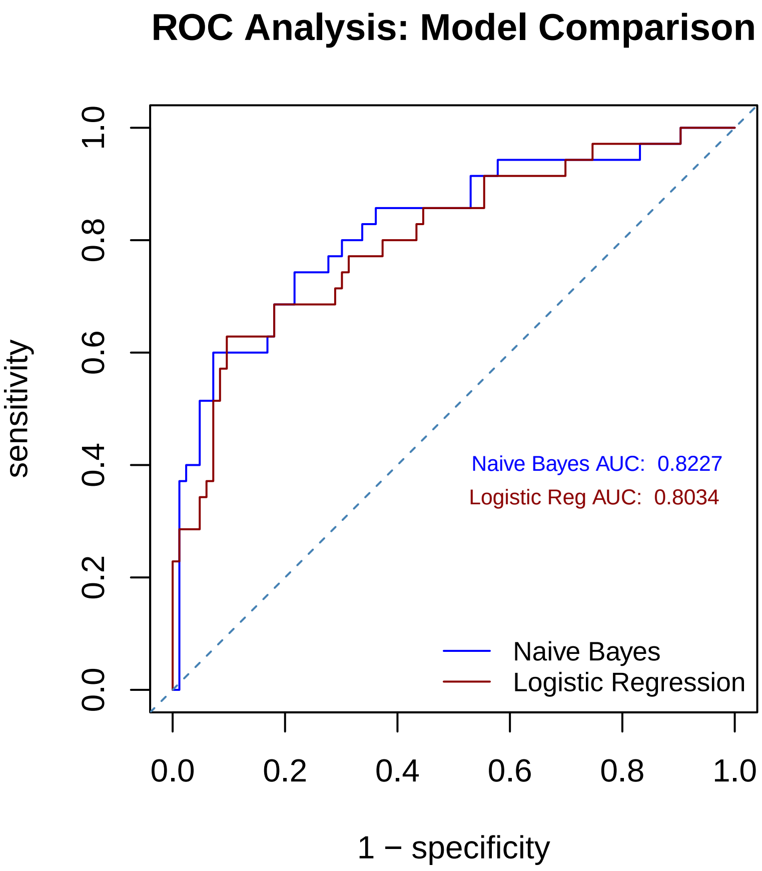
```
prob.predictions <- predict(gnb.model, test.data, type = "prob")[, "pos"]

# Performance metrics
accuracy <- conf.matrix$overall['Accuracy']
precision <- conf.matrix$byClass['Pos Pred Value']
recall <- conf.matrix$byClass['Sensitivity']
f1.score <- conf.matrix$byClass['F1']

Perf.Metric <- data.frame(Accuracy = accuracy,
           Precision = precision,
           Recall = recall,
           F1.Score = f1.score)
rownames(Perf.Metric) <- ""
pander(Perf.Metric)
```

Accuracy	Precision	Recall	F1.Score
0.7712	0.6176	0.6	0.6087

4. Global Performance - ROC Analysis

```
# ROC analysis is relevant for evaluating the model's performance across
#  different classification thresholds
roc.gnb <- roc(test.data$diabetes, prob.predictions, levels = c("neg", "pos"))
#plot(roc.gnb, main = "ROC Curve for Gaussian Naive Bayes")
gnb <- auc(roc.gnb)

# You can also use the pROC package for more detailed analysis
roc.obj <- roc(response = test.data$diabetes, predictor = prob.predictions)


par(pty="s")    # make a square plot
plot(1- roc.obj$specificities, roc.obj$sensitivities,
     type = "l", col= "blue", lty = 1,
     xlab = "1 - specificity",
     ylab = "sensitivity",
     main = "ROC Analysis: Model Comparison")

# Build logistic regression model
logit.model <- glm(diabetes ~ ., data = train.data, family = binomial())

# Make predictions
logit.probs <- predict(logit.model, test.data, type = "response")
logit.preds <- ifelse(logit.probs > 0.5, "pos", "neg")

# Confusion matrix: if using table(), need to take care of zero potential zero
# row and zero column
logit.conf.matrix <- confusionMatrix(factor(logit.preds, levels = c("neg", "pos")),
                                   test.data$diabetes, positive = "pos")
# ROC curve for logistic regression
roc.logit <- roc(test.data$diabetes, logit.probs)

lines(1- roc.logit$specificities, roc.logit$sensitivities,
      lty = 1, col = "darkred")
```

```
abline(0, 1, lty = 2, col = "steelblue")
# add legend for more than curves
legend("bottomright", legend = c("Naive Bayes", "Logistic Regression"),
       col = c("blue", "darkred"), lty = rep(1,2), cex = 0.8, bty = "n")
## text annotation
text(0.75, 0.4, paste(" Naive Bayes AUC: ",  round(auc(roc.gnb),4)),
     col="blue", cex = 0.7)
text(0.75, 0.34, paste("Logistic Reg AUC: ",  round(auc(roc.logit),4)),
     col="darkred", cex = 0.7)
```



The above ROC analysis indicates that the **Gaussian Naive Bayes** classification outperforms the classic logistic regression model. We have mentioned earlier, that **Gaussian Naive Bayes** rely on the normality assumption of the continuous feature variables. Non-normal continuous feature variables harm the performance of the **Gaussian Naive Bayes**. One way to relax normality distribution is to use the nonparametric density estimation for continuous feature variables.

6 Kernel Naive Bayes

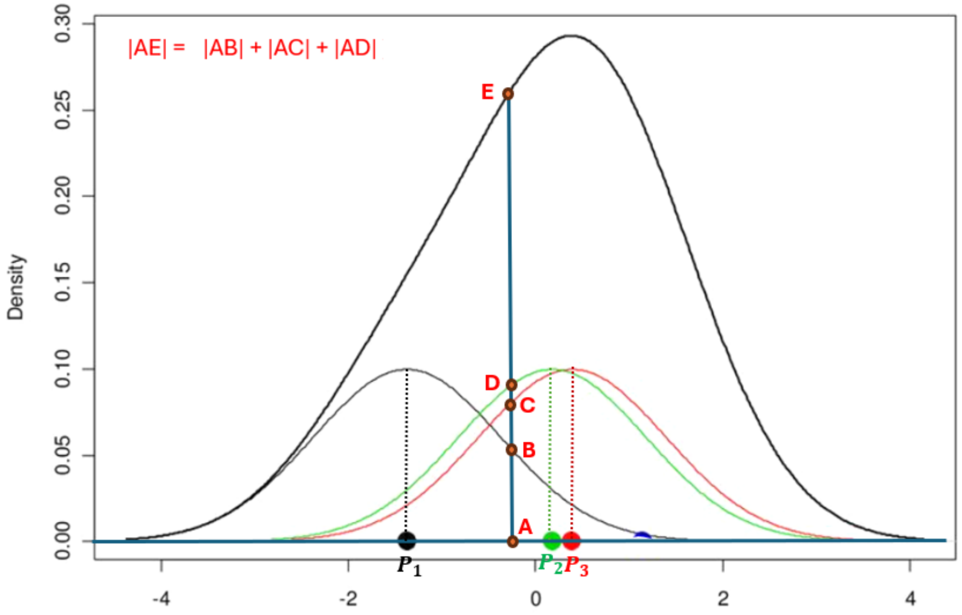
Kernel Naive Bayes (KNB) is an extension of the traditional **Naive Bayes (NB)** classifier that incorporates kernel methods to improve its performance, particularly in cases where the data distribution is not well-represented by simple parametric models (e.g., Gaussian, Multinomial, or Bernoulli). **KNB** replaces the parametric density estimation in **NB** with **non-parametric kernel density estimation (KDE)**.

6.1 Kernel Density Estimation (KDE)

If we know the distribution of a continuous random variable (e.g., a normal distribution), we can estimate the mean and variance from the sample (as done in Gaussian Naive Bayes), plot the density curve, and use it for analysis. If the parametric distribution is unknown, we can use a nonparametric approach, such as a histogram, to estimate the density instead of relying on a parametric distribution. Histograms are a simple way to visualize the distribution of data, but they have limitations:

- **Bin dependence**: The shape changes based on bin width and starting point.
- **Discontinuity**: They are not smooth.

KDE provides a smooth, continuous estimate of the **probability density function (PDF)** of a random variable. Instead of using discrete bins, **KDE** places a **smooth kernel function** (e.g., Gaussian) at each data point (P_1, P_2,and P_3 in the following figure) and sums them (|AB|, |AC|, and |AD| in the following figure) to produce a smooth density curve.



The following steps show how **KDE** works:

Step 1: Place a kernel at each data point.

Step 2: Sum all kernels to form the density estimate.

Step 3: Bandwidth (h) controls smoothness:

- Small h → Wiggly, overfitted estimate.
- Large h → Over-smoothed estimate.

The KDE formula is:

$$\hat{f}(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right)$$

Where K = kernel function (e.g., Gaussian, Epanechnikov, etc.), h = bandwidth, and X_i = data points. The following kernel density curve is based on a simulated data set with 100 values from the standard normal distribution.

An Illustrative Example: Consider data set $\{160, 170, 182, 186, 197\}$. The objective is to estimate the density curve based on the given data set using Gaussian kernel

$$K(u) = \frac{1}{\sqrt{2\pi}} \exp(-u^2/2)$$

and bandwidth $h = 10$. For illustration, we only calculate the density at $x = 180$. By the definition

$$\begin{aligned}\hat{f}(180) &= \frac{1}{n \times h} \sum_{i=1}^{n} K\left(\frac{180 - x_i}{h}\right) = \frac{1}{5 \times 10} \sum_{i=1}^{n} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{[(180 - x_i)/10]^2}{2}\right) \\ &= \frac{1}{5} \sum_{i=1}^{n} \frac{1}{\sqrt{2\pi} \times 10} \exp\left[-\frac{(180 - x_i)^2}{2 \times 10^2}\right]\end{aligned}$$

$$= \frac{1}{5}\left(\frac{1}{\sqrt{2\pi} \times 10} \exp\left[-\frac{(180-160)^2}{2 \times 10^2}\right] + \frac{1}{\sqrt{2\pi} \times 10} \exp\left[-\frac{(180-170)^2}{2 \times 10^2}\right] + \frac{1}{\sqrt{2\pi} \times 10} \exp\left[-\frac{(180-182)^2}{2 \times 10^2}\right]\right)$$

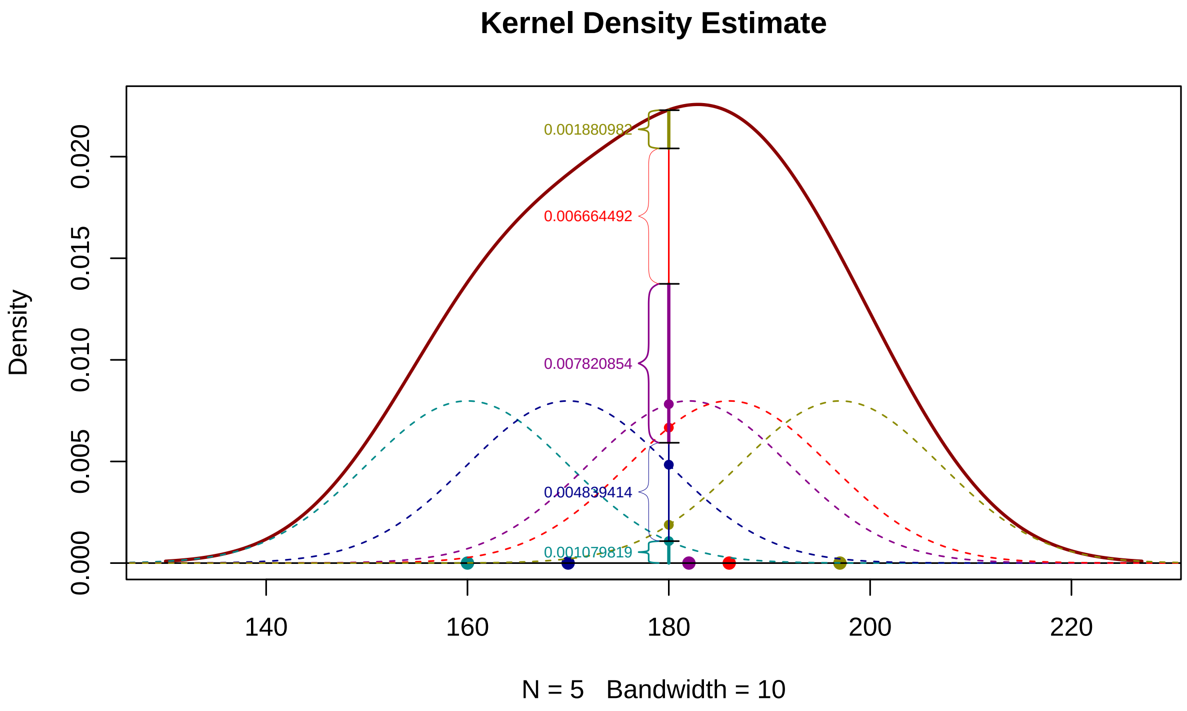
$$+\frac{1}{5}\left(\frac{1}{\sqrt{2\pi} \times 10} \exp\left[-\frac{(180-186)^2}{200}\right] + \frac{1}{\sqrt{2\pi} \times 10} \exp\left[-\frac{(180-197)^2}{200}\right]\right)$$

$$\approx 0.001079819 + 0.004839414 + 0.007820854 + 0.006664492 + 0.001880982 = 0.02228556.$$

Next, we provide a pictorial explanation of the above example in the following figure.

```r
# Generate some data
set.seed(123)
data <- c(160, 170, 182, 186, 197)
mycolor <- c("#008B8B", "#00008B", "#8B008B", "red", "#8B8B00")

# Default KDE plot
plot(density(data, bw = 10, kernel = "gaussian"), main = "Kernel Density Estimate",
     col = "darkred", lwd = 2, clim=c(120,240))
for( i in 1:5){
  points(data[i], 0, pch = 19, col = mycolor[i])
}
abline(0,0)

##
dd.180.pts = 0.2*dnorm(180, mean=data, sd = 10)

##
xx=seq(120, 240, length = 500)
for (i in 1:5){
  lines(xx, 0.2*dnorm(xx, mean=data[i], sd = 10), lty =2, col=mycolor[i])
}

##
for (i in 1:5){
  points(180, dd.180.pts[i], pch = 19, col = mycolor[i], cex = 0.7)
}

##
upper.y <- cumsum(dd.180.pts)
```

```
lower.y <- c(0, upper.y[-5])
wgt <- c(0.001079819, 0.004839414, 0.007820854, 0.006664492, 0.001880982)

##
for( i in 1:5){
   lines(rep(180,2), c(lower.y[i], upper.y[i]), col = mycolor[i], lwd = 1+i%%2, lty = 1)
}

##
for(i in 1:5){
   segments(179, upper.y[i], 181, upper.y[i])
}

##
mid.y = (lower.y + upper.y)/2
for (i in 1:5){
    brackets(179, lower.y[i], 179, upper.y[i], h=2, col = mycolor[i], lwd = i%%2)
}

##
for (i in 1:5){
   text(172, mid.y[i], paste(wgt[i]), col=mycolor[i], cex = 0.6)
 }
```

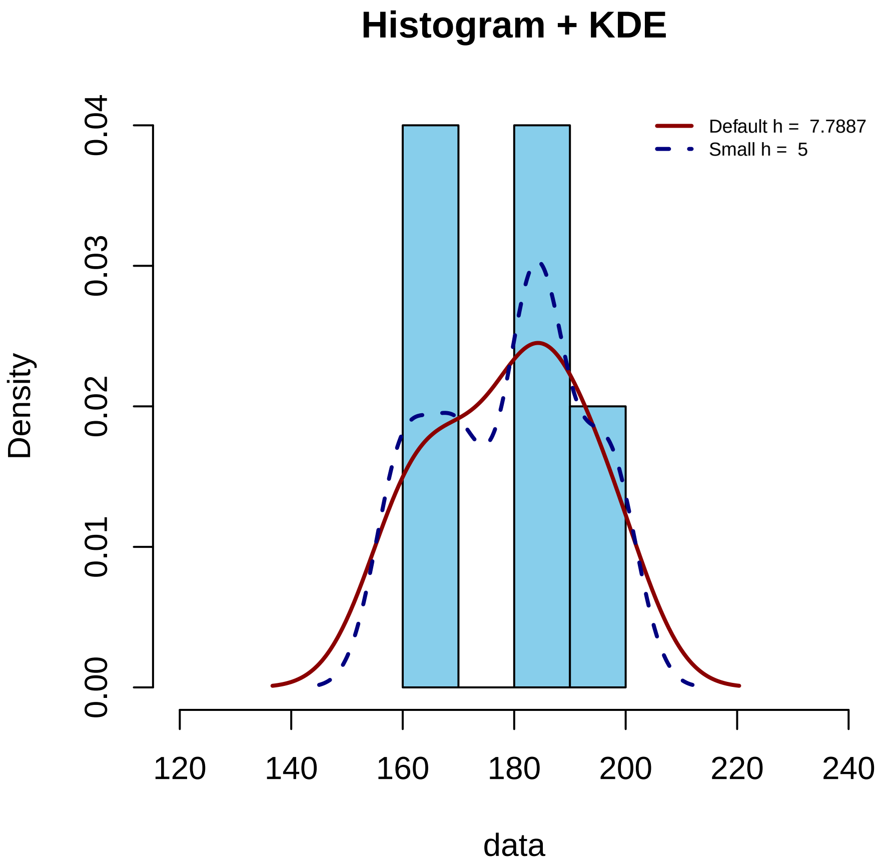


The example above is based on Gaussian kernel with a bandwidth $h = 10$ (user selected). In fact, h is a hyperparameter. It can be tuned to avoid **oversmooth** or **undersmooth**. There are also different methods for choosing default bandwidth. The default bin width displayed at the bottom of the following plot is based on the following formula (also called **Silverman's Rule of Thumb**):

$$h = 0.9 \times \min\left(\hat{\sigma}, \frac{IQR}{1.34}\right) \times n^{-1/5}$$

where $\hat{\sigma}$ = sample standard deviation, IQR = interquartile range, and n = sample size. This rule works well for **unimodal**, roughly normal distributions but may be **over-smooth** or **under-smooth** for more complex data. In the following figure, we add a kernel density curve with a smaller bin width.

```
# Compare with histogram

bin.w = 5
hist(data, freq = FALSE, col = "skyblue", main = "Histogram + KDE", xlim=c(120,240))
zz <- density(data)
binW = zz$bw
lines(zz, col = "darkred", lwd = 2)

# Adjust bandwidth (smaller h = more wiggly)
lines(density(data, bw = bin.w ), col = "navy", lwd = 2, lty = 2)

# Add legend
legend("topright", legend = c(paste("Default h = ",round(binW,4) ), paste("Small h = ",bin.w)),
       col = c("darkred", "navy"), lty = c(1, 2), lwd = 2, bty = "n", cex = 0.6)
```



For exploratory analysis, the default is often reasonable, but tuning (e.g., via cross-validation) may improve results for specific datasets.

6.2 Kernel Naive Bayes (KNB)

Kernel Naive Bayes (KNB) replaces the parametric density estimation in NB with non-parametric kernel density estimation (KDE). That is, instead of assuming a Gaussian (or other parametric) distribution, **KNB** estimates the probability density function (PDF) using kernels (e.g., Gaussian, Epanechnikov).

For a feature x in class C_k, the density is estimated as

$$p(x|C_k) = \frac{1}{n_k h}\sum_{i=1}^{n_k} K\left(\frac{x - x_i^{(k)}}{h}\right)$$

where n_k = number of samples in class C_k, $K(\cdot)$ is a kernel function. h is the bandwidth (a smooth hyperparameter). Most software programs using the Gaussian kernel function (standard normal density)

$$f(u) = \frac{1}{\sqrt{2\pi}}\exp(-u^2/2).$$

Naive Bayes Classification (Prediction) with KDE is based on the following steps:

- For a new sample $\mathbf{x} = (x_1, x_2, \cdots, x_k)$, compute the posterior probability

$$P(C_k|\mathbf{x}) \propto P(C_k)\prod_{j=1}^{d} p(x_j|C_k)$$

* The class with the highest posterior is selected.

Remarks on Bandwidth Selection: (1). The choice of h (bandwidth) is crucial. Too small → overfitting; too large → oversmoothing. (2). Common methods include **Rule-of-thumb** (Silverman's rule for Gaussian kernels) and **Cross-validation** (optimize classification accuracy).

Kernel Naive Bayes is a powerful extension of Naive Bayes that improves flexibility in modeling complex data distributions. While it retains the simplicity of NB, it leverages kernel methods for better performance in non-linear and non-Gaussian settings. However, it is best suited for low-to-medium dimensional datasets due to computational constraints.

6.3 Case Study

We use the default Gaussian kernel to fit the **Kernel Naive Bayes** to Pima Indian Diabetes data and then compare its ROC curve with Gaussian Naive and Bayes and the classic logistic regression model. R library **naivebayes** will be used in this case study.

The available kernel functions in R are: `gaussian`, `epanechnikov`, `rectangular`, `triangular`, `biweight`, `cosine`, `optcosine`.

All **kernel density estimations (KDE)** require bandwidth selection because the **bandwidth (bw)** critically controls the smoothness of the estimated density. The selection can be:

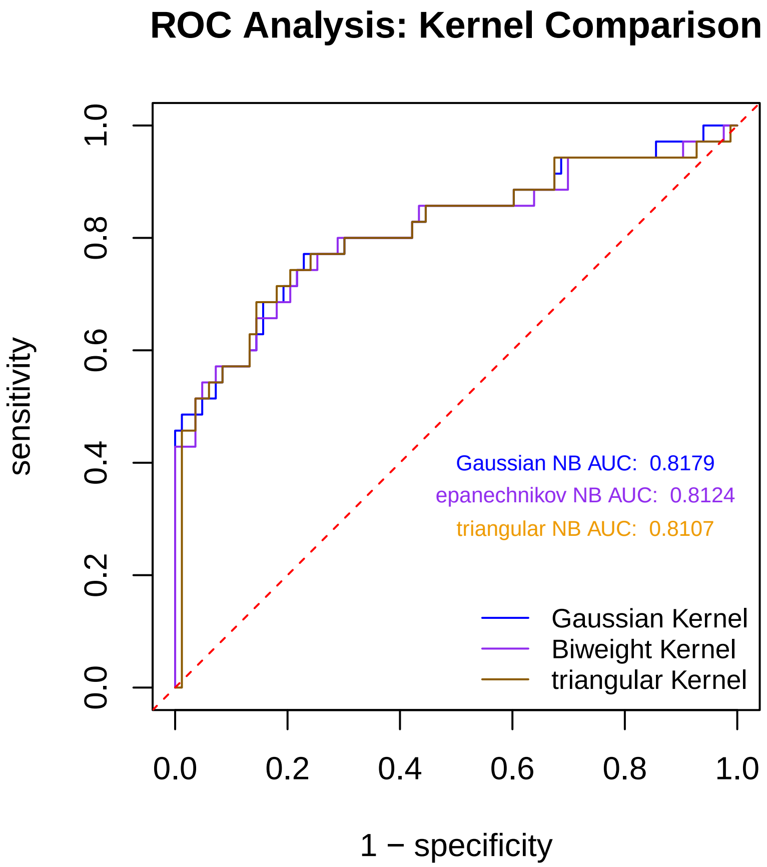
- **Manual (user-specified)** – User chooses h based on domain knowledge.
- **Rule-of-thumb (ROT)** – Heuristics like bw.nrd, bw.nrd0 (assume normality).
- **Data-driven methods** – More sophisticated approaches like:
  - Plug-in (`bw.SJ`) – Estimates the optimal h iteratively.
  - Cross-validation (`bw.ucv`, `bw.bcv`) – Minimizes estimation error.
- **Default methods** (e.g., in R's density()) use rules like bw.nrd0 if no bandwidth is specified.

The default setting in `naivs_bayes`: `kernel = Gaussian` and `bw = nrd0`

Impact of Kernel Functions

To explore the impact of kernel functions in the KNB (Kernel Naive Bayes), we developed several kernel naive Bayes algorithms using three kernels (Gaussian, Epanechnikov, and triangular) and compared their performance through ROC analysis. The results are summarized in the following ROC curves.

```r
##### NB with different kernel function
# Gaussian kernel with nrd0 (normal reference distribution) bin selector - default
knb.bw.nrd0 <- naive_bayes(diabetes ~ ., train.data, usekernel = TRUE, bw = "nrd0")
# epanechnikov kernel
knb.ep <- naive_bayes(diabetes ~ ., train.data, usekernel = TRUE, kernel = "epanechnikov")
# triangle kernel
knb.tri <- naive_bayes(diabetes ~ ., train.data, usekernel = TRUE, kernel = "triangular")

## Prediction: different kernel NB
#  Gaussian kernel - with default bw = nrd0
knb.bw.nrd0.probs <- predict(knb.bw.nrd0 , test.data, type = "prob")[, "pos"]
roc.bw.nrd0 <- roc(test.data$diabetes, knb.bw.nrd0.probs)
auc.bw.nrd0 <- round(roc.bw.nrd0$auc,4)
#  epanechnikov kernel - with default bw = nrd0
knb.ep.probs <- predict(knb.ep , test.data, type = "prob")[, "pos"]
roc.ep <- roc(test.data$diabetes, knb.ep.probs)
auc.ep <- round(roc.ep$auc,4)
#  epanechnikov kernel - with default bw = nrd0
knb.tri.probs <- predict(knb.tri , test.data, type = "prob")[, "pos"]
roc.tri <- roc(test.data$diabetes, knb.tri.probs)
auc.tri <- round(roc.tri$auc ,4)

# ROC base plot
par(pty="s")    # make a square plot
plot(1- roc.bw.nrd0$specificities, roc.bw.nrd0$sensitivities,
     type = "l", col= "blue", lty = 1,
     xlab = "1 - specificity",
     ylab = "sensitivity",
     main = "ROC Analysis: Kernel Comparison")
lines(1- roc.ep$specificities, roc.ep$sensitivities,
      lty = 1, col = "purple2")
lines(1- roc.tri$specificities, roc.tri$sensitivities,
      lty = 1, col = "orange4")
abline(0,1, lty=2, col = "red")
legend("bottomright", c("Gaussian Kernel", "Biweight Kernel", "triangular Kernel"),
       lty = rep(1,3), col = c("blue", "purple2", "orange4"), cex = 0.8, bty ="n")
text(0.73, 0.4, paste("Gaussian NB AUC: ", auc.bw.nrd0), col = "blue", cex = 0.7)
text(0.73, 0.34, paste("epanechnikov NB AUC: ", auc.ep), col = "purple2", cex = 0.7)
text(0.73, 0.28, paste("triangular NB AUC: ", auc.tri), col = "orange2", cex = 0.7)
```



Impact of Band Width

All **Kernel Naive Bayes (KNB)** models require a bandwidth parameter, which determines the smoothness of the resulting nonparametric density approximation for the feature variables. In practice, datasets often contain multiple continuous features. Ideally, we would select different optimal kernels with optimal bandwidths for density estimation across these features. This can be achieved through a hyperparameter tuning approach.

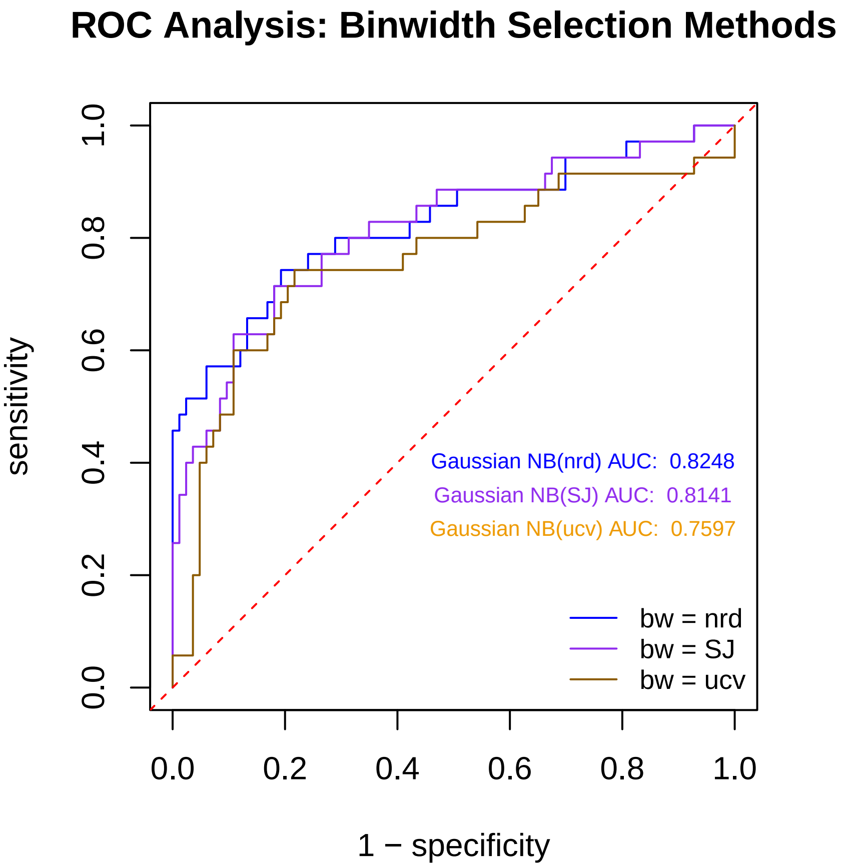
Some commonly used bandwidth selection methods are listed below:

- **Rule-of-Thumb Estimators**:
  - Silverman's Rule: $h = 1.06\sigma n^{-1/5}$ (for Gaussian kernel, `nrd0` - default)
  - Scott's Rule: $h = \sigma n^{-1/(d+4)}$ (multidimensional adjustment, `nrd`).
- Cross-Validation: Optimize bandwidth by maximizing likelihood or minimizing classification error (`ucv`).
- Plug-in Estimators: Asymptotically optimal bandwidths (e.g., Sheather-Jones method, `SJ`).

For illustration, we next construct several KNB models with a Gaussian kernel, each using different bandwidth selection methods (as listed above), and compare their overall performance.

```
# Gaussian NB with different bin widths
# Gaussian kernel with Silverman's rule of thumb:
knb.bw.nrd <- naive_bayes(diabetes ~ ., train.data, usekernel = TRUE, bw = "nrd")
```

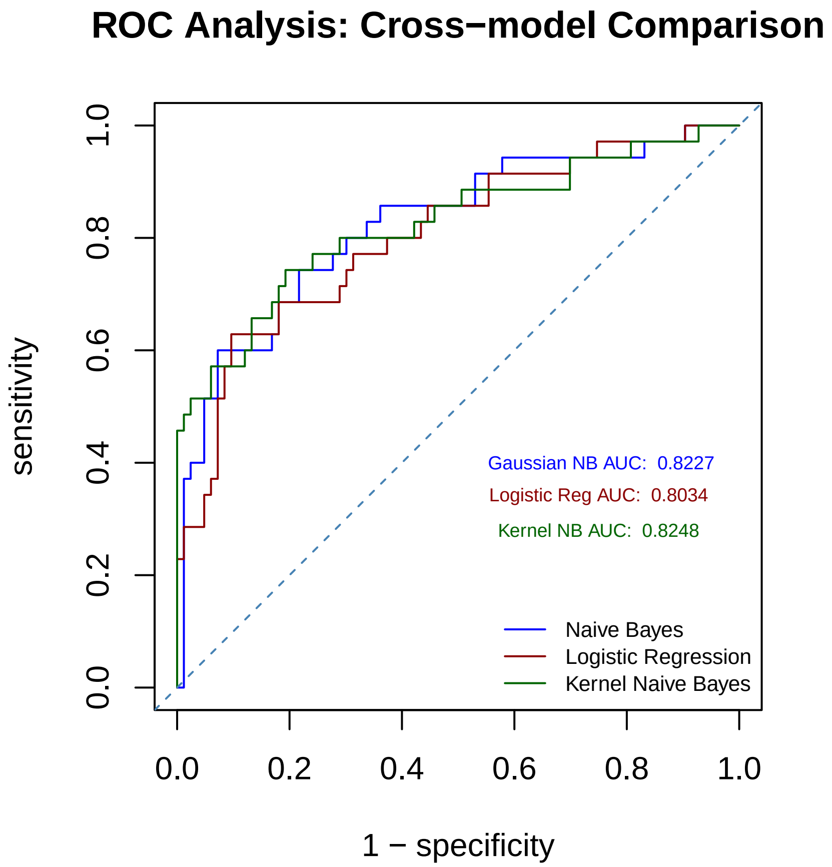
```r
# Gaussian kernel with SJ (Sheather-Jones plug-in):
knb.bw.SJ <- naive_bayes(diabetes ~ ., train.data, usekernel = TRUE, bw = "SJ")
# Gaussian kernel with ucv (unbiased cross-validation):
knb.bw.ucv <- naive_bayes(diabetes ~ ., train.data, usekernel = TRUE, bw = "ucv")

# Prediction of Gaussian NB with different bin widths
# bw = nrd0
knb.bw.nrd.probs <- predict(knb.bw.nrd , test.data, type = "prob")[, "pos"]
roc.bw.nrd <- roc(test.data$diabetes, knb.bw.nrd.probs)
auc.bw.nrd <- round(roc.bw.nrd$auc,4)
# bw = SJ
knb.bw.SJ.probs <- predict(knb.bw.SJ , test.data, type = "prob")[, "pos"]
roc.bw.SJ <- roc(test.data$diabetes, knb.bw.SJ.probs)
auc.bw.SJ <- round(roc.bw.SJ$auc,4)
# bw = ucv
knb.bw.ucv.probs <- predict(knb.bw.ucv , test.data, type = "prob")[, "pos"]
roc.bw.ucv <- roc(test.data$diabetes, knb.bw.ucv.probs)
auc.bw.ucv <- round(roc.bw.ucv$auc,4)

# ROC base plot
par(pty="s")    # make a square plot
plot(1- roc.bw.nrd$specificities, roc.bw.nrd$sensitivities,
     type = "l", col= "blue", lty = 1,
     xlab = "1 - specificity",
     ylab = "sensitivity",
     main = "ROC Analysis: Binwidth Selection Methods")
lines(1- roc.bw.SJ$specificities, roc.bw.SJ$sensitivities,
      lty = 1, col = "purple2")
lines(1- roc.bw.ucv$specificities, roc.bw.ucv$sensitivities,
      lty = 1, col = "orange4")
abline(0,1, lty=2, col = "red")
legend("bottomright", c("bw = nrd", "bw = SJ", "bw = ucv"),
       lty = rep(1,3), col = c("blue", "purple2", "orange4"), cex = 0.8, bty ="n")
text(0.73, 0.4, paste("Gaussian NB(nrd) AUC: ", auc.bw.nrd), col = "blue", cex = 0.7)
text(0.73, 0.34, paste("Gaussian NB(SJ) AUC: ", auc.bw.SJ), col = "purple2", cex = 0.7)
text(0.73, 0.28, paste("Gaussian NB(ucv) AUC: ", auc.bw.ucv ), col = "orange2", cex = 0.7)
```



The above ROC analysis shows that using the same Gaussian kernel, different bandwidth selection methods yielded varying performance metrics. The modified normal reference distribution method (`nrd`), incorporating Scott's rule for multidimensional adjustment, achieved the highest AUC.

To conclude this case study, we next construct three comparative models: (1) Gaussian Naive Bayes, (2) Kernel Naive Bayes with Gaussian kernel and modified normal reference distribution (`nrd`), and (3) classic logistic regression - then evaluate their overall performance.

```r
# Gaussian Naive Bayes
gnb.model <- naive_bayes(diabetes ~ ., data = train.data)
gnb.probs <- predict(gnb.model, test.data, type = "prob")[, "pos"]
roc.gnb <- roc(test.data$diabetes, gnb.probs)
auc.gnb <- round(roc.gnb$auc,4)

# Build logistic regression model
logit.model <- glm(diabetes ~ ., data = train.data, family = binomial())
# Make predictions
logit.probs <- predict(logit.model, test.data, type = "response")
roc.logit <- roc(test.data$diabetes, logit.probs)


# ROC base plot
par(pty="s")     # make a square plot
plot(1- roc.gnb$specificities, roc.gnb$sensitivities,
     type = "l", col= "blue", lty = 1,
     xlab = "1 - specificity",
     ylab = "sensitivity",
     main = "ROC Analysis: Cross-model Comparison")
# ROC curve for logistic regression

lines(1- roc.logit$specificities, roc.logit$sensitivities,
      lty = 1, col = "darkred")
lines(1- roc.bw.nrd$specificities, roc.bw.nrd$sensitivities,
      lty = 1, col = "darkgreen")

# add a reference line reflecting random guess
abline(0, 1, lty = 2, col = "steelblue")

#plot(roc.logit, col = "blue", add = TRUE)
legend("bottomright", legend = c("Naive Bayes", "Logistic Regression", "Kernel Naive Bayes"),
       col = c("blue", "darkred", "darkgreen"),
       lty = rep(1,3), cex = 0.7, bty = "n")
text(0.75, 0.4, paste(" Gaussian NB AUC: ",  auc.gnb),
     col="blue", cex = 0.6)
text(0.75, 0.34, paste("Logistic Reg AUC: ",  round(auc(roc.logit),4)),
     col="darkred", cex = 0.6)
text(0.75, 0.28, paste("Kernel NB AUC: ",  auc.bw.nrd),
     col="darkgreen", cex = 0.6)
```



The ROC analysis above demonstrates that **Naive Bayes** methods outperform the baseline logistic regression model. Specifically, the **kernel Naive Bayes** achieves a higher AUC than the regular **Gaussian Naive Bayes**.

7 Concluding Remarks

We have discussed both **Naive Bayes** algorithms with a primary focus on **Gaussian Native Bayes** and **Kernel Naive Bayes** in this note. The **Gaussian Naive Bayes** algorithm is essentially a parametric method while **Kernel Naive Bayes** is a nonparametric method using the kernel density estimation to estimate the distribution of numeric features. We also compare the **Naive Bayes** and the **classic logistic regression**.

7.1 Naive Bayes vs Logistic Regression

The above case study shows that Naive Bayes algorithms the classic logistic regression model. In general, **Kernel Naive Bayes** (KNB) may outperform logistic regression in certain scenarios due to several structural and practical advantages:

1. Handling of Non-Linear Decision Boundaries

- **Logistic Regression**: Assume that the log odds and features are linear correlated (unless explicitly kernelized or polynomial features are added).
- **KNB**: Through **kernel density estimation (KDE)**, it can model non-linear and multi-modal

distributions natively, capturing complex patterns without manual feature engineering.

2. **Distributional Flexibility**
   - **Logistic Regression**: Although not explicitly specified, logistic regression implicitly assumes that feature variables follow unimodal distributions without extreme skewness.
   - **KNB**: Makes no parametric assumptions about feature distributions (nonparametric). It adapts to:
     - Skewed or asymmetric data
     - Heavy-tailed distributions
     - Multiple modes (unimodal vs. multimodal)

```
#library(psych)
# Basic version
pairs.panels(train.data[, -9])
```

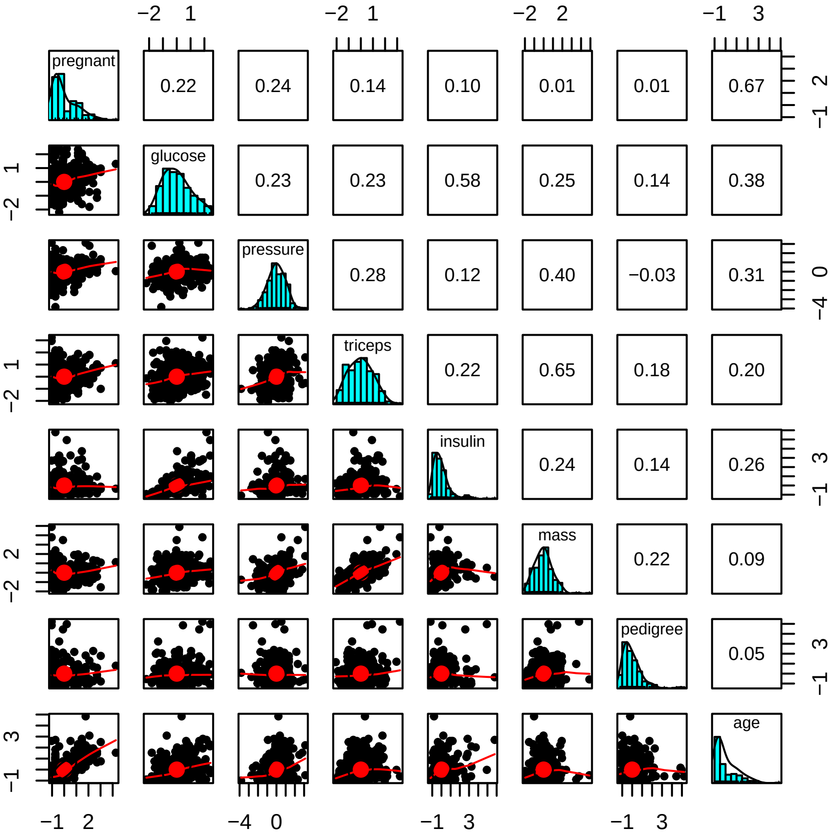


Figure 1: Age and insulin are very skewed, pregnant is bimodal. The rest of the features are slightly skewed.

3. **Robustness to Feature Independence Violations**
   - While both methods technically assume feature independence, KNB is often more robust to mild dependencies because:
   - **Kernel smoothing** implicitly accounts for local correlations in the data.
   - **Logistic regression**'s coefficients can degrade significantly with correlated features.

4. **Small Data Advantage**
    - **KNB**'s kernel-based density estimation can generalize better with limited training data by "smoothing" observations.
    - **Logistic regression** may overfit without regularization.
5. **No Optimization Required**
    - **Logistic Regression**: Requires solving a convex optimization problem (sensitive to learning rate, iterations, etc.).
    - **KNB**: Computes densities directly from data (no gradient descent or risk of convergence issues).

7.2 Hyperparameter Tuning

In Naive Bayes, hyperparameters primarily involve the selection of kernel functions and bandwidth methods. Unlike many other algorithms, Naive Bayes calculates posterior probabilities for class labels through direct computation of density or probability mass functions for individual features but not by optimizing a loss function. This approach provides two key advantages:

- **Computational Efficiency**: Training is significantly faster since it avoids iterative loss optimization.
- **Resource Allocation**: Freed computational resources can be dedicated to hyperparameter tuning for improved performance.

In the case study above, we manually selected a few kernel functions and a few different bandwidth selections based on the Gaussian kernel and found that the kernel naive Bayes with **Gaussian kernel** using **normal reference distribution** (`nrd`) performed the best. If writing a custom function, we tune the by finding the optimal combination of kernel functions and bandwidth selection methods to identify the best model.

7.3 Potential Structural Improvements

Gaussian Naive Bayes and **Kernel Naive Bayes** represent two extreme cases from a statistical perspective.

- **Gaussian Naive Bayes** is a parametric algorithm that assumes each continuous feature follows a normal distribution. The parameters of this distribution (μ and σ) can be directly estimated from the training data.
- **Kernel Naive Bayes**, conversely, is a completely distribution-free, nonparametric approach. It employs kernel density estimation to estimate a smooth density curve (i.e., a smoothed histogram) for every continuous feature variable without making any distributional assumptions.

When the assumed parametric distribution is correctly specified, the resulting estimate is statistically efficient and will improve the performance of all subsequent inferences. However, if the distribution is misspecified, the estimated distribution may produce misleading or incorrect results.

While kernel density estimation is data-driven and distribution-free, the resulting inference may not be statistically efficient. Nevertheless, it avoids the risk of fundamentally wrong inferences that can occur with misspecified parametric models.

Potential Methods for Improvements

Several semi-parametric and flexible parametric estimation methods lie between fully parametric (e.g., Gaussian, Poisson) and completely nonparametric (e.g., kernel density estimation, histograms) approaches. These can improve the performance of Naive Bayes (NB) for continuous features by better capturing the underlying data distribution without overfitting.

Semiparametric Naive Bayes (NB) algorithms combine parametric assumptions (for some aspects of the data) with nonparametric flexibility (for others) to improve performance while maintaining computational efficiency. Below are some existing semiparametric NB variants and their key characteristics.

- **Gaussian Mixture Naive Bayes (GMNB)** models each feature using a Gaussian Mixture Model (GMM) instead of a single Gaussian. It is useful for multimodal or skewed continuous features.
- **Transformed Gaussian Naive Bayes (TGNB)** applies a power transform (Box-Cox/Yeo-Johnson) to features before using Gaussian NB. It is good for non-Gaussian but unimodal features.
- **Bernstein Naive Bayes (BNB)** models densities using Bernstein polynomials, which are smooth approximations of empirical data.

7.4 A New Directions to Improve Naive Bayes(Optional)

Empirical likelihood-based Semiparametric Naive Bayes methods: This semiparametric Naive Bayes (NB) approach combines:

- **Nonparametric estimation** of the feature distributions per class (stratified by the target variable).
- **Parametric modeling of the density ratio** between each class-conditional distribution and a baseline/reference distribution.

This is a powerful hybrid approach, as it avoids strict parametric assumptions (e.g., Gaussianity) while leveraging parametric efficiency for the ratio function. Next, we outline the idea of the method, discuss its advantages, and link it to existing methods.

Semiparametric Density Ratio Estimation

Some notations:

- $p_k(x)$ - Nonparametric estimate of the density of feature X for class k.
- $p_0(x)$ - baseline (or reference) density distribution of the feature for the baseline class of the target variable.
- The **density ratio** $r_k(x) = p_k(x)/p_0(x)$ is modeled parametrically like $r_k(x) = \exp[\theta_k^T\phi(x)]$, where $\phi(x)$ is a base function such polynomial, spline).

The class-conditional density is then.

$$p_k(x) = r_k(x) \times p_0(x).$$

How It Works in Naive Bayes

- **Nonparametric step**: Estimate $p_0(x) and $p_k(x)$ using kernel density estimation (KDE), histograms, or empirical cumulative distribution functions (ECDF)s.
- **Parametric step**: Fit a parametric model (e.g., logistic, exponential family) to the ratio $r_k(x)$. For each class k, solve the following optimization

$$\min_{\theta_k} \sum_{i:y_i=k} [\log p_0(x_i) + \log r_k(x_i;\theta_k)] + \text{Regularization}$$

Using MLE, logistic regression, and gradient descent.

- **Prediction**: Use Bayes' theorem with the reconstructed $p_k(x)$ as

$$p_k(x) = P(y = k|x) \propto P(Y = k)r_k(x;\theta_k) \times p_0(x).$$

This density ratio estimation framework is a unifying approach to generalize and extend existing **Naive Bayes (NB)** algorithms into semiparametric families. Some of the advantages of the framework

- **Flexibility**: Choose $p_0(x)$ and $r_k(x)$ to match data structure.
- **Interpretability**: Ratios $r_k(x)$ reveal how classes modify the baseline.

- **Compatibility**: Extends any NB variant (discrete/continuous).

Some Theoretical Insights

- **Recovers Classic NB**: If $p_0(x)$ is conjugate to $r_k(x)$ (e.g., Gaussian + quadratic ratios).
- **Generalizes KNB**: If $r_k(x)$ is nonparametric, it reduces to kernel naive Bayes (KNB).
- **Bias-Variance Tradeoff**: Controlled by complexity of $r_k(x)$.