Project n^{o}3: Random Fourier Features Based Kernel Density Estimation in a Naive Bayes Classifier Advanced Machine Learning (MDS)

Federico Clerici Raffaele D'Agostino Davide Volpi

Date: 16/01/2026

Abstract

This report addresses the computational bottleneck in Kernel Density Estimation (KDE) for Naive Bayes classifiers, where exact kernel summations impose $O(N)$ inference cost per test point, limiting scalability of non-parametric density modeling for complex distributions. Leveraging Naive Bayes from Part I and Random Fourier Features (RFF) from Part III of the course, we approximate shift-invariant Gaussian kernels via Bochner's theorem, constructing explicit feature maps. This reformulates class-conditional densities as dot products with precomputed kernel mean embeddings, reducing inference complexity to $O(D)$ independent of training size N. Extensive experiments on controlled synthetic datasets reveal RFF-KDE recovers 97% of exact KDE's AUC with 10x speedup. While incurring higher training/memory costs, RFF-KDE excels in medium-to-hard settings with sufficient RFF rank ($D \geq 295$) where parametric Gaussian NB fails, establishing a scalable bridge between non-parametric flexibility and linear-time efficiency.

Note: all the notation and abbreviations that will be used can be found in Table 3.

1 Introduction

Generative classifiers such as Naive Bayes rely on accurate estimation of class-conditional densities to achieve strong predictive performance. While non-parametric methods like Kernel Density Estimation (KDE) provide the flexibility to model complex data distributions without restrictive parametric assumptions, their computational cost scales linearly with the number of training samples ($O(N)$). As a result, exact KDE becomes impractical for large-scale or time-sensitive inference tasks.

This work addresses this scalability limitation by introducing an efficient approximation of the KDE-based Naive Bayes classifier using *Random Fourier Features* (RFF). Building on Bochner's Theorem, RFF enables the approximation of shift-invariant kernels, such as the Gaussian, through randomized finite-dimensional feature mappings. This transformation converts costly kernel evaluations into simple dot products in a lower-dimensional space, reducing inference complexity from $O(N)$ to $O(D)$, where $D < N$.

We develop and evaluate an RFF-KDE Naive Bayes classifier that replaces exact kernel computations with this randomized feature approximation. Our analysis investigates the trade-offs between approximation fidelity, predictive performance (AUC), and computational efficiency across both synthetic and real-world datasets. The results demonstrate that the proposed approach significantly decouples inference time from dataset size, offering a scalable and effective alternative to conventional non-parametric generative models.

2 Problem Statement

We consider a supervised classification task with dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$. A Naive Bayes classifier assigns class c by maximizing the posterior $p(c) \prod_j p(x^{(j)}|c)$. Standard KDE estimates these marginals as:

$$\widehat{p}(\mathbf{x}|c) = \frac{1}{N_c} \sum_{i:y_i=c} k(\mathbf{x}, \mathbf{x}_i) \tag{1}$$

The fundamental problem is that evaluating this sum has $O(N)$ complexity. Every query requires traversing the entire training set, creating a latency bottleneck that grows linearly with data volume.

Our goal is to reduce this inference complexity by approximating the kernel with a finite-dimensional feature map $\mathbf{z} : \mathbb{R}^d \to \mathbb{R}^D$ such that $k(\mathbf{x}, \mathbf{y}) \approx \mathbf{z}(\mathbf{x})^\top \mathbf{z}(\mathbf{y})$. This allows us to precompute a single mean vector for each class:

$$\widehat{p}(\mathbf{x}|c) \approx \mathbf{z}(\mathbf{x})^\top \left(\frac{1}{N_c} \sum_{i:y_i=c} \mathbf{z}(\mathbf{x}_i) \right) = \mathbf{z}(\mathbf{x})^\top \boldsymbol{\mu}_c \tag{2}$$

We aim to determine if this approximation significantly reduces inference time while maintaining competitive classification performance compared to the exact baseline.

3 Related Work

Naive Bayes classifiers traditionally rely on parametric density assumptions (e.g., Gaussian distributions), which impose restrictive modeling constraints on class-conditional distributions. Kernel Density Estimation (KDE) Naive Bayes addresses this by estimating class-conditional densities nonparametrically via kernel smoothing, achieving superior performance on complex distributions at the cost of $O(n^2)$ evaluation complexity per test point [7]. For large-scale datasets, this quadratic complexity in training set size becomes prohibitive, motivating approximation strategies that preserve nonparametric flexibility while reducing computational burden.

The computational bottleneck of exact kernel methods is addressed by random Fourier features (RFF). Rahimi & Recht [12] introduced RFF to approximate shift-invariant kernels via Monte Carlo sampling of the Fourier spectrum, reducing complexity from $O(n^2)$ to $O(nD)$ where D is the number of RFFs. Sutherland & Schneider [15] proved that RFF preserves kernel distances with approximation error decreasing as $O(D^{-1/2})$, establishing statistical consistency. Recent work has extended RFF guarantees to kernel-based density estimation and two-sample testing [4, 9], demonstrating that approximation quality improves polynomially with feature dimension while maintaining statistical performance.

Our work applies RFF-approximated kernel mean embeddings to Naive Bayes classification, providing a computationally efficient alternative to KDE-based approaches. We empirically investigate whether KDE with RFF approximation preserves the classification performance of exact KDE Naive Bayes while achieving the computational efficiency necessary for large-scale applications. Our approach reduces complexity to $O(D)$ after precomputing RFF representations. We systematically evaluate this trade-off across datasets with varying distributional complexity, comparing Gaussian Naive Bayes (parametric baseline), KDE Naive Bayes (exact nonparametric), and KDE Naive Bayes with RFF (scalable nonparametric) to quantify performance retention under computational constraints.

4 Data and Preprocessing

4.1 Data description

4.1.1 Synthetic Data Generation

To assess the robustness of the classifiers in different settings, we generated synthetic binary classification datasets with $d = 10$ features using a custom generation pipeline. Task complexity is controlled via a `difficulty` parameter that simultaneously modulates three factors: feature correlation (ρ), class mean separation, and the proportion of non-Gaussian features. In this way we define different levels of violation of independence and parametric assumptions.

Each dataset is generated from a correlated ($\rho \sim U(\rho_{\min}, \rho_{\max})$) multivariate Gaussian base with class-dependent mean shifts on informative features. Selected features are then transformed to non-Gaussian distributions with class-specific shapes, and Gaussian noise ($\sigma = 0.1$) is injected to simulate measurement error.

We defined three difficulty levels (*Easy*, *Medium*, *Hard*) as summarized in Table 1.

Table 1: Configuration of Synthetic Datasets by difficulty

Difficulty	**Correlation (ρ)**	**Separation Factor**	**Informative Feat.**	**Non-Gaussian Feat.**
Easy	$0.0 - 0.2$	4.0	10 (100%)	0 (All Gaussian)
Medium	$0.3 - 0.5$	2.0	5 (50%)	8
Hard	$0.6 - 0.8$	0.0 (Overlapping)	5 (50%)	10 (All non Gaussian)

4.2 Exploratory data analysis

In this section, we analyze the structural properties of both the synthetic and real-world datasets.

We visualize the pairwise relationships and marginal distributions of the generated features across the three difficulty settings.

Figure 1 illustrates the *Easy* configuration. As expected from the generation parameters (see Table 1), the two classes are linearly separable with distinct means and minimal overlap. A simple parametric model would likely achieve near-perfect performance here.

In the *Medium* configuration (Figure 2), the class centroids move closer, and the introduction of noise features begins to obscure the decision boundary.

The *Hard* configuration (Figure 3) represents the most critical test case for our models. As shown in the diagonal density plots, the distributions of the two classes are nearly perfectly overlapping in terms of location (means are identical). The only discriminative information lies in the shape of the distributions (e.g., bimodality vs. unimodality) rather than their centers.

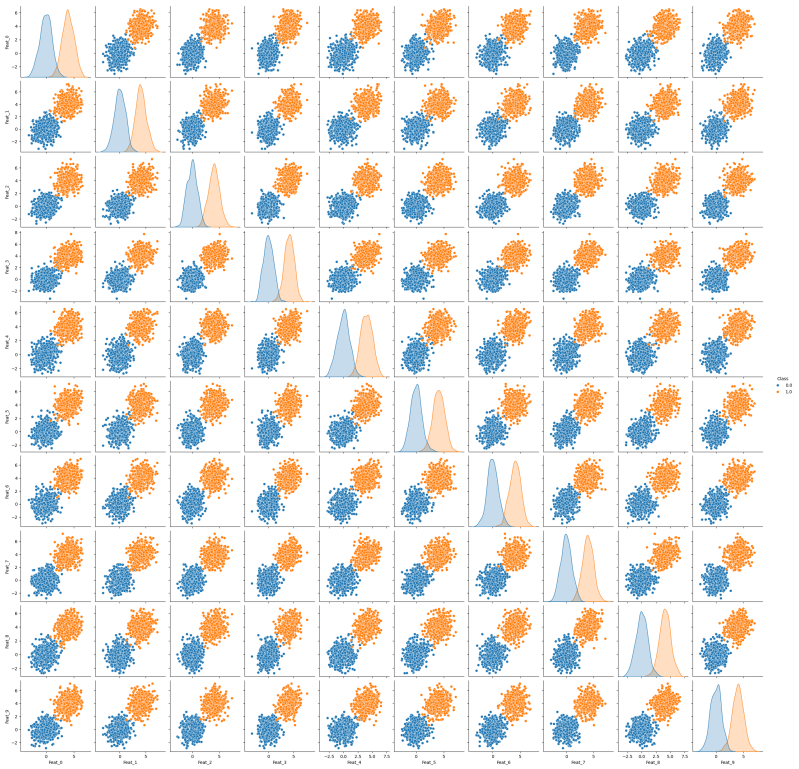

Figure 1: Pair-plot of variables from synthetic dataset (*Easy*). Classes are clearly separated by location.

4.3 Preprocessing steps

We applied a stratified sampling strategy to split datasets into training (70%) and testing (30%) sets, preserving class distribution across both subsets. All features were standardized to zero mean and unit variance using `sklearn`'s `StandardScaler`, defined as $x' = (x - \mu)/\sigma$, where μ and σ were computed exclusively on the training set to prevent data leakage. No missing value imputation or dimensionality reduction was required, as the synthetic datasets are complete and purely numerical.

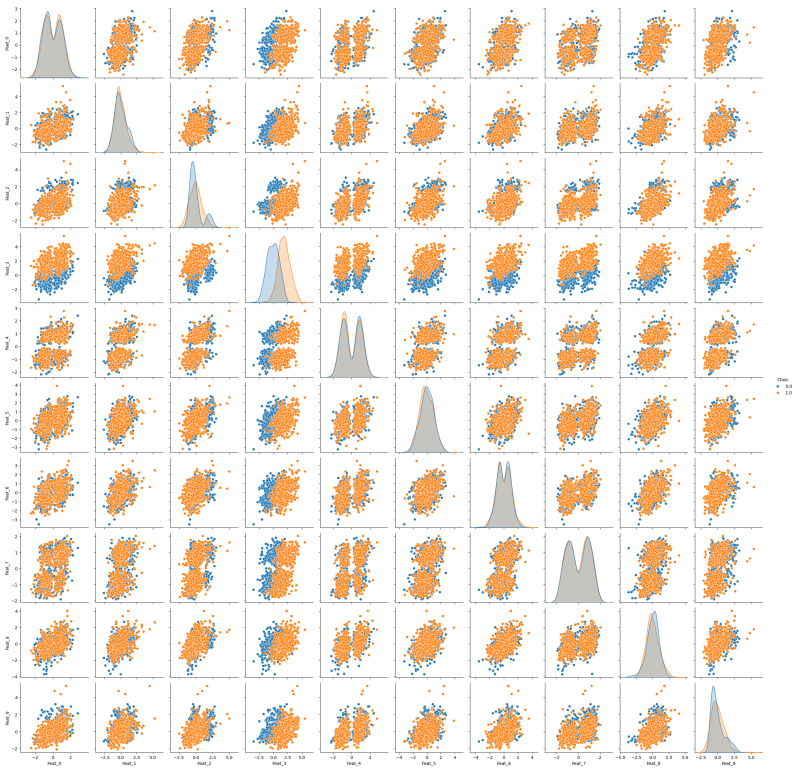

Figure 2: Pair-plot of variables from synthetic dataset (*Medium*). Margins between classes are reduced.

Figure 3: Pair-plot of variables from synthetic dataset (*Hard*). Means are overlapping; classes are distinguishable only by distribution shape.

No additional preprocessing or feature engineering was applied, as our objective was to evaluate and compare the algorithms across varying scenarios and difficulty levels without introducing bias toward any particular method. The synthetic datasets were specifically generated for this study to ensure controlled experimental conditions.

5 Methodology

This section presents the probabilistic classification framework based on Kernel Density Estimation (KDE). We formulate the Naive Bayes classifier using exact KDE to model class-conditional densities, which serves as the kernel-based performance baseline. We also include Gaussian Naive Bayes as a parametric baseline for comparison, despite not being kernel-based. We then introduce Random Fourier Features (RFF) to approximate the RBF kernel through a randomized feature map $z : \mathbb{R}^d \to \mathbb{R}^D$. This explicit low-dimensional representation enables efficient computation of KDE via inner products in $\mathbb{R}^D$, avoiding the costly kernel evaluations required by exact KDE. Computational complexity for all methods is analyzed in Section 6.

5.1 Experimental protocol

To ensure the statistical reliability of our results and strictly assess the trade-off between predictive performance and computational efficiency, we adopted the following experimental protocol.

Datasets and Tasks. To provide a comprehensive evaluation, we test the proposed method on both synthetic and real-world benchmarks. Our synthetic experiments involve datasets generated across three distinct difficulty levels (*Easy*, *Medium*, and *Hard*) designed to rigorously test the model's robustness under varying degrees of class separability and non-linearity. To validate performance in a realistic setting, we evaluate all methods on the *MAGIC Gamma Telescope* dataset (Appendix C), a physics-based classification benchmark with complex, high-dimensional feature interactions.

Baselines and Models. We benchmark the performance of our RFF-based KDE Naive Bayes against two standard reference models. The *Gaussian Naive Bayes (GNB)* serves as a parametric baseline, representing the efficient but rigid assumption of normality. Conversely, the *Exact KDE Naive Bayes (KDE-NB)* serves as the non-parametric gold standard, providing an upper bound on the achievable predictive performance through full kernel expansion.

Hyperparameters and Variations. In our analysis we first investigate the impact of the approximation rank (D) by varying the number of random features on a logarithmic scale (from $D = 1$ to 1500) to quantify how feature dimensionality correlates with approximation quality. Second, to assess scalability, we measure training and inference latency across a range of training set sizes ($N \in \{1000, \ldots, 5000\}$), allowing us to empirically verify the theoretical complexity bounds of each method. Importantly, Silverman's rule [14] has been used for bandwidth selection of both KDE and RFF as deterministic and requires no tuning, eliminating the need for cross-validation on this parameter. The RFF dimension D is varied systematically as a study variable to isolate approximation effects and characterize the performance-efficiency trade-off across the full range of approximation ranks.

Statistical significance To assess whether RFF-KDE preserves the classification performance of exact KDE, we employ the Two One-Sided Tests (TOST) procedure for statistical equivalence testing [13, 8]. We test equivalence via TOST whether $|\mathrm{AUC}_{\mathrm{KDE}} - \mathrm{AUC}_{\mathrm{RFF}}|$ falls within margins $\delta \in \{0.05, 0.02\}$, a practical threshold (5%) and strict statistical bound (2%) to test two different levels of equivalence. For each difficulty level and RFF dimension D, we conduct paired TOST across multiple runs. Since AUC differences violate normality assumptions, we employ bootstrap resampling (10,000 iterations) to obtain distribution-free p-values. We apply Holm-Bonferroni correction [6] to control familywise error rate and report Cohen's d effect sizes with 90% confidence intervals.

Evaluation Metrics and Reproducibility. Performance is evaluated using the Area Under the ROC Curve (AUC). Efficiency is measured via wall-clock time (in seconds) for both training and inference phases. To ensure statistical significance and account for randomness in both data splitting and RFF weight sampling, each experimental configuration is repeated across 20 distinct random seeds. We report the mean values and standard deviations across these runs.

Experiments are implemented in Python; exact library versions are listed in `requirements.txt`. Computations were run on ASUS Zenbook 14 (Intel i7, 11th Gen) and two MacBook Air (M1/M2).

5.2 Method 1: Gaussian Naive Bayes

5.2.1 Model formulation

The Gaussian Naive Bayes (GNB) classifier [1] serves as our parametric baseline. It assumes that features are conditionally independent given the class c and that each feature x_j follows a univariate Gaussian distribution. The class-conditional likelihood is given by:

$$p(x_j|c) = \frac{1}{\sqrt{2\pi\sigma_{cj}^2}} \exp\left(-\frac{(x_j - \mu_{cj})^2}{2\sigma_{cj}^2}\right) \tag{3}$$

where μ_{cj} and σ_{cj}^2 are the sample mean and variance computed from the training set for class c. The classification rule maximizes the posterior $p(c|\mathbf{x}) \propto p(c) \prod_{j=1}^{d} p(x_j|c)$.

5.2.2 Theoretical properties

GNB is computationally efficient and robust to overfitting due to its high bias. However, the strong assumption of normality often leads to poor approximations for complex, multimodal, or skewed data distributions.

5.3 Method 2: KDE Naive Bayes

5.3.1 Model formulation

To address the limitations of GNB, the KDE Naive Bayes replaces the parametric Gaussian assumption with a non-parametric Kernel Density Estimator [5, 11]. For a feature x_j and class c, the density is estimated as:

$$\widehat{p}(x_j|c) = \frac{1}{N_c h} \sum_{i=1}^{N_c} k\left(\frac{x_j - x_{ij}}{h}\right) \tag{4}$$

where N_c is the number of training samples in class c, h is the bandwidth, and $k(\cdot)$ is the kernel function (RBF kernel).

Classification then proceeds exactly as in standard Naive Bayes: the posterior for class c is proportional to $p(c) \prod_j \widehat{p}(x_j|c)$, maintaining the conditional independence assumption between features given the class label.

5.3.2 Theoretical properties

KDE is a consistent estimator that converges to the true marginal density as $N \to \infty$. While it provides superior predictive performance by adapting to arbitrary data shapes, it suffers from severe computational scalability issues due to the $O(N)$ inference cost per feature evaluation.

5.4 Method 3: RFF-KDE Naive Bayes

5.4.1 Model formulation

Our proposed method uses Random Fourier Features (RFF), a method proposed by [12] to approximate kernel methods like KDE, providing approximated likelihoods and decoupling inference time from dataset size.

Random Fourier Features (RFF). The Random Fourier Features (RFF) method provides an explicit, finite-dimensional approximation to shift-invariant kernels by exploiting the fundamental connection between positive definite kernels and characteristic functions established by Bochner's Theorem [2].

Theorem 5.1 (Bochner's Theorem)**.** *A continuous, shift-invariant kernel $k(\mathbf{x}, \mathbf{y}) = k(\mathbf{x} - \mathbf{y})$ is positive definite if and only if it is the Fourier transform of a non-negative finite measure. For a normalized kernel ($k(\mathbf{0}) = 1$), this measure corresponds to a probability density $p(\boldsymbol{\omega})$, yielding:*

$$k(\boldsymbol{\delta}) = \int_{\mathbb{R}^d} e^{i\boldsymbol{\omega}^\top \boldsymbol{\delta}}\, p(\boldsymbol{\omega})\, d\boldsymbol{\omega} = \mathbb{E}_{\boldsymbol{\omega}\sim p}\left[e^{i\boldsymbol{\omega}^\top \boldsymbol{\delta}}\right]$$

where $\boldsymbol{\delta} = \mathbf{x} - \mathbf{y}$ represents the displacement vector.

This establishes that any shift-invariant kernel can be expressed as the expectation of complex exponentials over its spectral distribution $p(\boldsymbol{\omega})$.

For the Gaussian RBF kernel $k(\mathbf{x}, \mathbf{y}) = \exp(-\gamma\|\mathbf{x} - \mathbf{y}\|^2)$, the Fourier dual is also Gaussian: $p(\boldsymbol{\omega}) = \mathcal{N}(\mathbf{0}, 2\gamma\mathbb{I})$. Applying Euler's formula and exploiting the fact that the imaginary components cancel when averaging over the symmetric distribution, we obtain the real-valued decomposition:

$$k(\boldsymbol{\delta}) = \mathbb{E}_{\boldsymbol{\omega}}\left[\cos(\boldsymbol{\omega}^\top \boldsymbol{\delta})\right] \tag{5}$$

The RFF approximation Monte Carlo samples this expectation. Drawing D independent frequencies $\{\boldsymbol{\omega}_m\}_{m=1}^D \sim \mathcal{N}(\mathbf{0}, 2\gamma\mathbb{I})$ and random phase shifts $b_m \sim \mathcal{U}[0, 2\pi]$ (to ensure unbiased randomization), we construct the explicit feature map:

$$\mathbf{z}(\mathbf{x}) = \sqrt{\frac{2}{D}} \begin{bmatrix} \cos(\boldsymbol{\omega}_1^\top \mathbf{x} + b_1) \\ \vdots \\ \cos(\boldsymbol{\omega}_D^\top \mathbf{x} + b_D) \end{bmatrix} \in \mathbb{R}^D \tag{6}$$

The normalization factor $\sqrt{2/D}$ ensures that the expected inner product converges to the true kernel value:

$$k(\mathbf{x}, \mathbf{y}) \approx \mathbf{z}(\mathbf{x})^\top \mathbf{z}(\mathbf{y}) \tag{7}$$

with approximation error decreasing as $O(1/\sqrt{D})$ by the Central Limit Theorem.

Application to KDE. This approximation enables a fundamental reformulation of the kernel density estimator. Substituting the inner product representation into the class-conditional density:

$$\widehat{p}(\mathbf{x}|c) = \frac{1}{N_c} \sum_{i:y_i=c} k(\mathbf{x}, \mathbf{x}_i) \approx \frac{1}{N_c} \sum_{i:y_i=c} \mathbf{z}(\mathbf{x})^\top \mathbf{z}(\mathbf{x}_i) \tag{8}$$

Since $\mathbf{z}(\mathbf{x})$ is independent of the summation index, we can factor it out:

$$\widehat{p}(\mathbf{x}|c) \approx \mathbf{z}(\mathbf{x})^\top \underbrace{\left(\frac{1}{N_c} \sum_{i:y_i=c} \mathbf{z}(\mathbf{x}_i)\right)}_{\boldsymbol{\mu}_c} = \mathbf{z}(\mathbf{x})^\top \boldsymbol{\mu}_c \tag{9}$$

Here, $\boldsymbol{\mu}_c \in \mathbb{R}^D$ represents an approximated *Kernel Mean Embedding (KME)* of class c in the random Fourier feature space. Critically, this vector is computed once during training and stored, reducing inference complexity from $O(N_c)$ to $O(D)$, independent of the training set size.

Again, classification follows the one of the standard Naive Bayes framework exactly.

5.4.2 Theoretical properties and justification

The theoretical foundation of RFF lies in the uniform convergence of the random feature approximation to the exact kernel. As established by Rahimi and Recht [12], the approximation error decreases at a rate of $O(D^{-1/2})$. This guarantees that for a sufficiently large rank D, the RFF estimator statistically converges to the exact KDE estimator. Consequently, the method retains the consistency and flexibility of non-parametric density estimation while mapping the infinite-dimensional kernel problem into a finite-dimensional Euclidean space suitable for linear operations.

5.5 Comparison framework

Models are compared using mean AUC-ROC and its standard deviation across 20 different seeds. This ensures a consistent, threshold-independent basis for comparison across heterogeneous models.

Significance is assessed by analyzing the distribution of AUC-ROC scores over 10,000 bootstrap samples from the 20 seeds runs.

All models are trained and tested on identical stratified splits, using consistent tuning and evaluation protocols. Preprocessing is model-appropriate but always fitted on training data only. Probability outputs (`predict_proba`) are used for AUC-ROC computation to ensure comparability across architectures.

6 Complexity Analysis

We compare the computational efficiency of Kernel Density Estimation Naive Bayes (KDE-NB), Random Fourier Features Naive Bayes (RFF-NB), and Gaussian Naive Bayes (GNB). Table 2 summarizes the theoretical time and space complexities with respect to the dataset size N, number of features F, number of classes C, and RFF components D.

We assume a dataset split where $N_{train} \approx N$ and $N_{test} \approx N$ (asymptotically). The analysis highlights the trade-off between the non-parametric flexibility of KDE and the computational efficiency of RFF and GNB approximations. A full theoretical derivation of these computational complexities can be found in Appendix F.

Algorithm	Training Time	Inference Time	Space Complexity
KDE-NB	$\mathcal{O}(F \cdot N \log N)$	$\mathcal{O}(F \cdot N^2)$	$\mathcal{O}(F \cdot N)$
RFF-NB	$\mathcal{O}(F \cdot D \cdot N)$	$\mathcal{O}(F \cdot D \cdot N)$	$\mathcal{O}(C \cdot F \cdot D)$
Gaussian NB	$\mathcal{O}(F \cdot N)$	$\mathcal{O}(F \cdot N)$	$\mathcal{O}(C \cdot F)$

Table 2: Asymptotic complexity comparison. N denotes total dataset size, F feature dimensionality, C number of classes, and D the number of Monte Carlo samples (RFF components). Note that KDE inference scales quadratically with data size, whereas RFF and GNB scale linearly.

7 Discussion of results

7.1 Synthetic Datasets

The experimental analysis on synthetic data reveals a hierarchy of model suitability depending on the complexity of the underlying data distribution. By examining the trade-off between predictive performance (AUC) and computational efficiency across three difficulty levels, we can identify the distinct regimes where each approach excels. Appendices D and E) show the full results for the synthetic datasets.

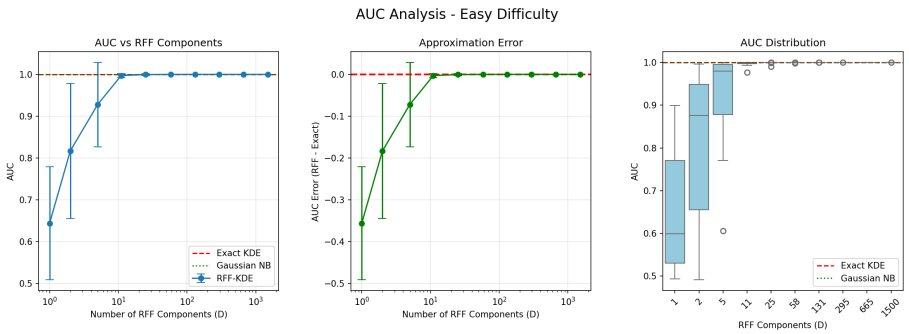

Figure 4: AUC analysis for RFF-KDE on *Easy* setting dataset (log scale for D). Shaded regions and error bars indicate ± standard deviation across multiple runs.

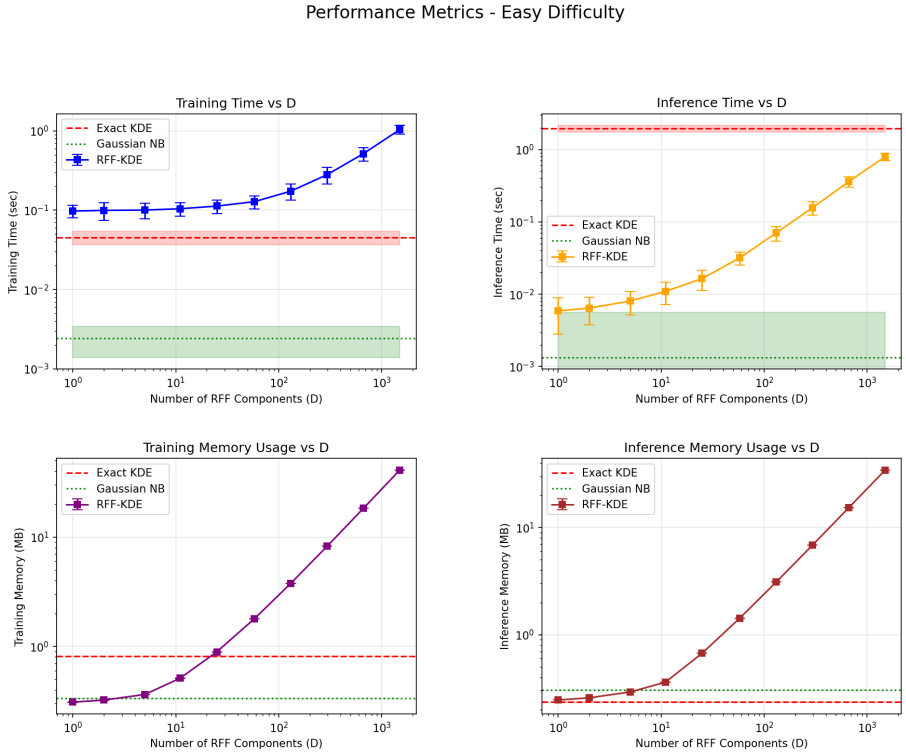

Figure 5: Computational performance analysis for RFF-KDE on *Easy* setting dataset with logarithmic scaling on both axes. Shaded regions and error bars indicate ± standard deviation across multiple runs.

In the *Easy* scenario (Figure 4 and 5), the data distribution is sufficiently simple that the parametric and independence assumptions of Gaussian Naive Bayes (GNB) hold almost perfectly. As shown in the results, GNB achieves near-optimal performance (AUC ≈ 1.0) with negligible computational cost. RFF-KDE matches GNB AUC at $D = 11$ (TOST-equivalent to Exact KDE, $\delta = 0.05$ and $\delta = 0.02$), but at higher cost. Consequently, for well-behaved, unimodal data, the additional complexity of kernel-based methods, whether exact or approximate, is unnecessary, and the standard GNB remains the most efficient choice, as expected.

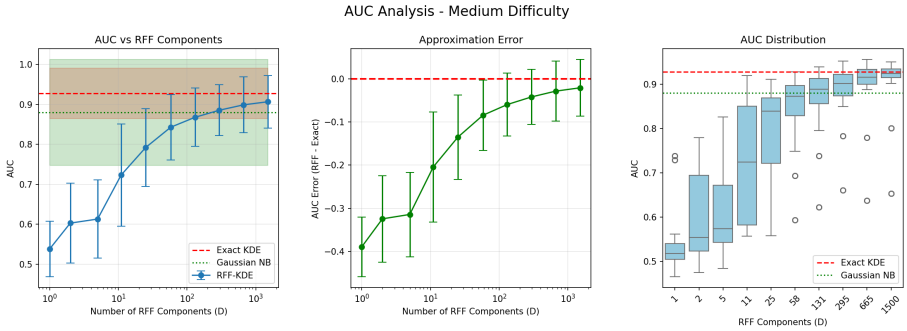

Figure 6: AUC analysis for RFF-KDE on *Medium* setting dataset (log scale for D). Shaded regions and error bars indicate ± standard deviation across multiple runs.

The situation changes in the *Medium* case (Figure 6 and 7), which represents a transition zone. Here, the strict Gaussian and independence assumptions begin to fail, causing the GNB performance to drop noticeably below the Exact KDE baseline. This gap highlights the need for the flexibility of non-parametric density estimation. At the same time, the behaviour of GNB is not completely erratic: a few features exhibit clearly separated class means, so that most positive samples lie systematically above or below most negative ones along those coordinates. These "location shifts" allow GNB to construct a reasonably well-ordered score function, which explains

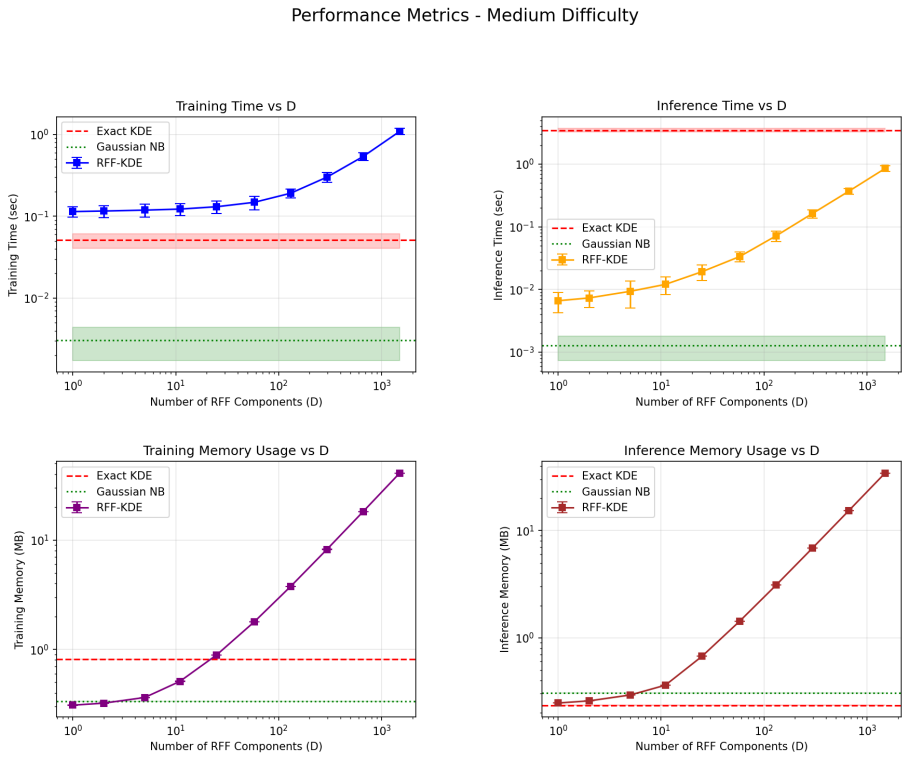

Figure 7: Computational performance analysis for RFF-KDE on *Medium* setting dataset with logarithmic scaling on both axes. Shaded regions and error bars indicate ± standard deviation across multiple runs.

why its AUC remains around 0.88 despite the evident model misspecification in both features distributions and independence. In this regime, RFF-KDE provides a valid middle ground. TOST ($\delta = 0.05$) confirms equivalence at $D = 295$, boosting AUC over GNB while cutting Exact KDE inference cost.

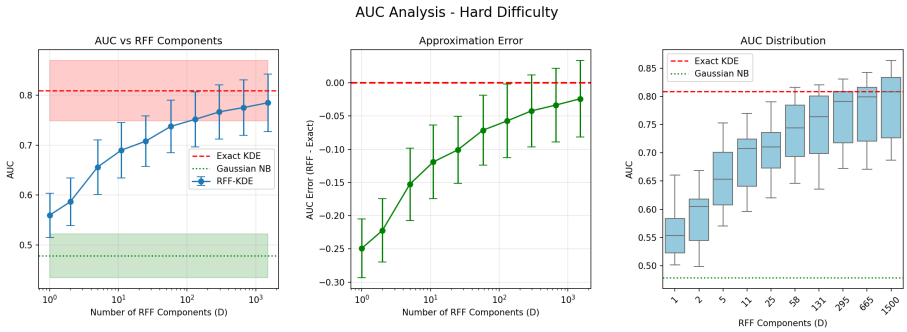

Figure 8: AUC analysis for RFF-KDE on *Hard* setting dataset (log scale for D). Shaded regions and error bars indicate ± standard deviation across multiple runs.

The experimental results for the *Hard* scenario (Figure 8 and Figure 9), clearly illustrate the trade-off between model expressiveness, computational latency, and resource utilization. As shown in Figure 8, the clear violation of independence and normality in the *Hard* scenario causes the parametric GNB to achieve near-random performance. In fact, it flatlines at an AUC of ≈ 0.5, which is equivalent to random guessing, confirming that simple parametric assumptions are insufficient for this data distribution.

The RFF-KDE model exhibits asymptotic convergence toward the Exact KDE baseline (AUC $\approx$ 0.81). While low-D approximations ($D < 100$) yield AUC < 0.75, higher D narrows the gap, reaching an $AUC_{RFF} \approx 97\%$ of AUC_{KDE} at $D = 1500$ with 10x inference speedup over Exact KDE's $O(N)$ latency (Figure 9).

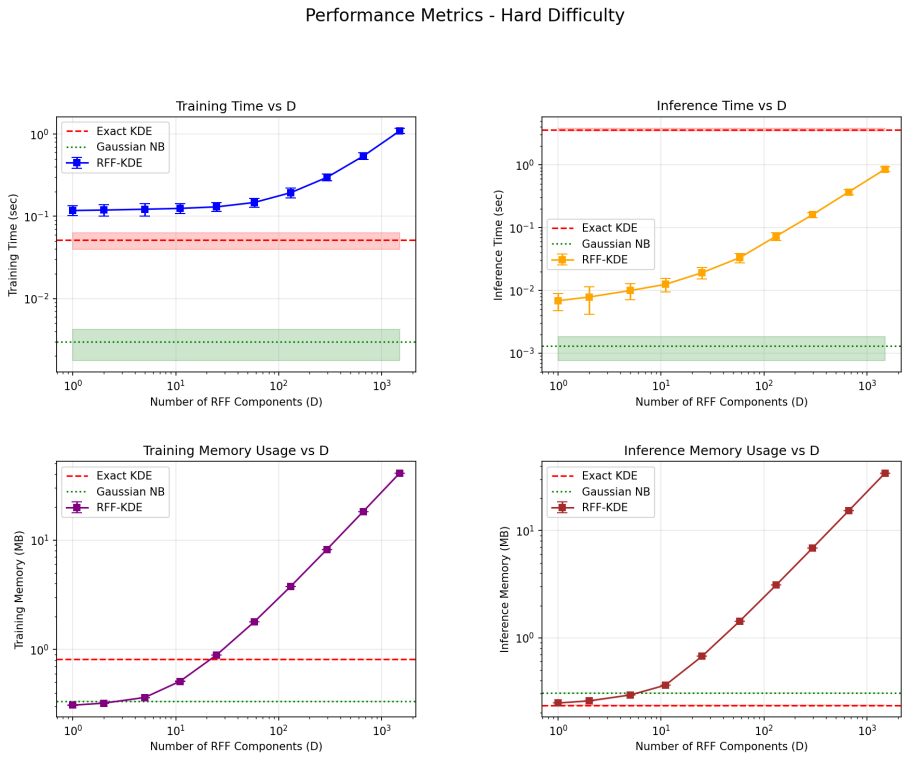

Figure 9: Computational performance analysis for RFF-KDE on *Hard* dataset with logarithmic scaling on both axes. Shaded regions and error bars indicate ± standard deviation across multiple runs.

TOST equivalence analysis validates these trade-offs. At a practical $\delta = 0.05$ margin, $D = 295$ ensures equivalence across difficulties. Stricter $\delta = 0.02$ testing confirms that in the most challenging scenarios the performances are not fully recovered with a limited number of RFF, positioning RFF as efficient approximation rather than identical substitute.

However, this acceleration is not without cost. The "Training Time" and "Memory Usage" plots in Figure 9 reveal the resources required to construct the explicit feature maps.

- Training Time: Unlike Exact KDE, which is a *lazy learner* with near-zero training cost, RFF-KDE requires projecting the data into the feature space. RFF training is slower than the Exact KDE baseline.
- Memory Footprint: The storage requirement for RFF grows with D. For high-fidelity approximations ($D \geq 1000$), the model footprint exceeds 30 MB, significantly surpassing the memory required to simply access the original dataset (Exact KDE).

To confirm the theoretical complexity introduced in Section 5, we further analyzed the computational scaling as a function of dataset size ($N \in [1000, 5000]$), with RFF dimensionality set adaptively to $D = \lceil \log(N) \rceil$. Figure 10 illustrates the divergence in inference complexity. While Exact KDE scales linearly ($O(N)$), causing inference time to surge from $\sim$ 0.1s to $\sim$ 2.7s, the RFF-KDE maintains a virtually constant, near-zero latency. This confirms that RFF effectively decouples inference cost from training set size. The trade-off is visible in the training phase, where RFF incurs a higher, though stable, offline cost ($\approx$ 0.1s) compared to the *lazy* Exact KDE baseline.

8 Conclusion

This study systematically evaluated the feasibility of Random Fourier Features (RFF) as a scalable approximation for Kernel Density Estimation in Naive Bayes binary classification.

Our results demonstrate that the utility of RFF-KDE is strictly context-dependent. In simple, unimodal scenarios (*Easy*), the added complexity of kernel methods is redundant, confirming that standard parametric models like Gaussian Naive Bayes remain the optimal choice as expected.

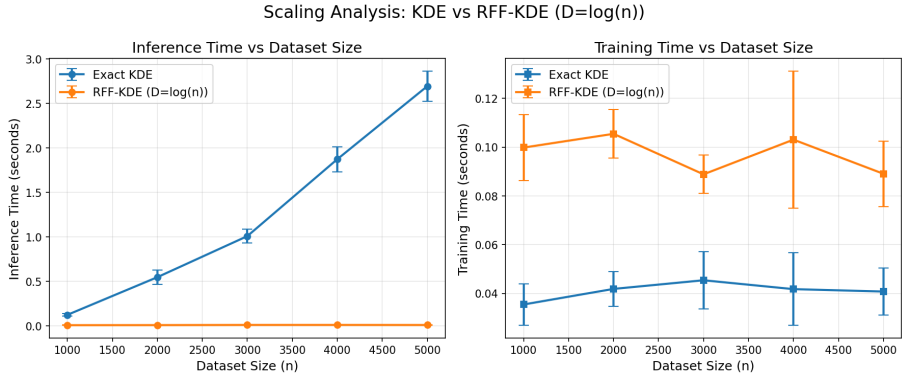

Figure 10: Computational scaling analysis for Exact KDE versus RFF-KDE with $D = \log(N)$ as a function of dataset size. Error bars indicate ± 1 standard deviation across multiple runs.

However, as data complexity increases (*Medium* to *Hard*), the parametric and independence assumptions break down, necessitating non-parametric flexibility. In these regimes, RFF-KDE proves to be a powerful enabling technology. It successfully narrow the gap between the speed of parametric models and the performance of exact kernel methods, recovering up to 97% of the non-parametric performance baseline while reducing inference latency. TOST equivalence testing confirms statistical equivalence at practical thresholds ($\delta = 0.05$ margin, $D = 11$ for *Easy* $D = 295$ for *Medium* and *Hard* scenarios), though stricter bounds ($\delta = 0.02$) reveal persistent approximation bias in complex distributions, validating RFF as efficient approximation rather than exact substitute with a limited number of RFF.

Importantly, RFF-KDE inherits the conditional independence assumption from the underlying Naive Bayes framework, which remains violated in our correlated synthetic datasets. Yet, performance stays acceptable because discriminative power often resides primarily in the marginal feature distributions rather than complex joint dependencies.

While the method incurs a higher memory footprint and upfront training cost compared to *lazy* exact methods, this investment yields a critical advantage: it decouples inference speed from dataset size. This scalability makes RFF-KDE a viable candidate for real-time, large-scale applications where traditional non-parametric density estimation would be computationally prohibitive.

This study presents several constraints that limit the generalizability of our findings. First, the RFF-KDE approach inherits the conditional independence assumption from Naive Bayes, which is violated in datasets with strong feature correlations (*Medium*, *Hard* scenarios). While marginal discriminative power partially compensates, the extent of performance degradation under severe dependency structures remains unquantified, limiting applicability to domains requiring joint density modeling. Second, experimental validation is limited to binary classification and a single real-world dataset, leaving behavior in multiclass settings ($C \gg 2$) and diverse application domains unexplored. Finally, the random sampling of Fourier frequencies introduces performance variance across initializations (particularly evident in the *Medium* scenario), raising reliability concerns for safety-critical applications. Our scalability analysis terminates at $N = 5000$; for truly large-scale problems (e.g. $N > 10^5$), the $O(F \cdot D \cdot N)$ training complexity may remain prohibitive even when D is large. Future research should investigate orthogonal RFF [16], which reduces approximation variance, and compare against Nyström sampling to determine whether the reduced variance of landmark-based methods justifies the storage overhead of retaining a training subset.

Reflections on Learning Outcomes This study provided insights into the theoretical foundations of kernel methods, particularly the operational role of Bochner's Theorem. Understanding how a shift-invariant kernel can be represented as the Fourier transform of a probability measure bridged the gap between abstract Reproducing Kernel Hilbert Spaces (RKHS) and practical implementation. By applying Random Fourier Features (RFF), we directly experienced how infinite-dimensional feature maps can be approximated via finite Monte Carlo sampling. This highlighted the fundamental trade-off in kernel estimation: balancing the expressive power of non-parametric methods with the necessity of explicit, low-dimensional approximations to make learning tractable.

9 Declarations on GenAI

The authors would like to acknowledge the use of ChatGPT [10] for assistance with spell checking, debugging, and generating well-formatted plots. The tool was used solely to improve clarity and presentation; all analysis and results were performed independently by the authors.

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 1st edition, 2006.

[2] Salomon Bochner. *Harmonic analysis and the theory of probability*. Courier Corporation, 2005.

[3] R.K. Bock. MAGIC Gamma Telescope [Dataset]. UCI Machine Learning Repository, 2004. DOI: https://doi.org/10.24432/C52C8B.

[4] Kacper P Chwialkowski, Aaditya Ramdas, Dino Sejdinovic, and Arthur Gretton. Fast two-sample testing with analytic representations of probability measures. *Advances in Neural Information Processing Systems*, 28, 2015.

[5] Richard A Davis, Keh-Shin Lii, and Dimitris N Politis. Remarks on some nonparametric estimates of a density function. In *Selected Works of Murray Rosenblatt*, pages 95–100. Springer, 2011.

[6] Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, pages 65–70, 1979.

[7] George H John and Pat Langley. Estimating continuous distributions in bayesian classifiers. *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 338–345, 1995.

[8] Daniël Lakens, Anne M Scheel, and Peder M Isager. Equivalence testing for psychological research: A tutorial. *Advances in methods and practices in psychological science*, 1(2):259–269, 2018.

[9] Zhu Li, Jean-Francois Ton, Dino Oglic, and Dino Sejdinovic. Towards a unified analysis of random fourier features. In *International conference on machine learning*, pages 3905–3914. PMLR, 2019.

[10] OpenAI. Chatgpt (january 16 version). [Large language model] https://chat.openai.com/, 2025.

[11] Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076, 1962.

[12] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20, 2007.

[13] Donald J Schuirmann. A comparison of the two one-sided tests procedure and the power approach for assessing the equivalence of average bioavailability. *Journal of pharmacokinetics and biopharmaceutics*, 15(6):657–680, 1987.

[14] Bernard W Silverman. *Density estimation for statistics and data analysis*. Routledge, 2018.

[15] Danica J Sutherland and Jeff Schneider. On the error of random fourier features. *arXiv preprint arXiv:1506.02785*, 2015.

[16] Felix Xinnan X Yu, Ananda Theertha Suresh, Krzysztof M Choromanski, Daniel N Holtmann-Rice, and Sanjiv Kumar. Orthogonal random features. *Advances in neural information processing systems*, 29, 2016.

A Notation

Table 3: Model abbreviations used throughout the text

Full text	**Abbreviation**
Area Under the Curve	AUC
Receiver Operative Curve	ROC
Random Fourier Features	RFF
Gaussian Naive Bayes	GNB
Kernel Density Estimation	KDE
Probability Density Function	PDF
Kernel Mean Embedding	KME
Naive Bayes	NB

B Algorithms

We present the pseudocode for the three classifiers used in the experiments: the exact KDE Naive Bayes, the approximated RFF Naive Bayes, and the parametric GNB.

B.1 KDE Naive Bayes

Algorithm 1 KDENaiveBayes: Training and Inference

Require: Training data $X_{train} \in \mathbb{R}^{N_{train} \times F}$, Labels $y_{train} \in \{1, \dots, C\}$
Ensure: Fitted Model $\mathcal{M}$, Predictions P
Procedure FIT(X_{train}, y_{train})
Identify unique classes $C = \text{unique}(y_{train})$
for each class $c \in C$ **do**
  Extract samples $X_c = \{x \in X_{train} \mid y = c\}$
  Calculate prior $\pi_c = |X_c|/N_{train}$
  for each feature $j \in \{1, \dots, F\}$ **do**
    Extract 1D feature vector $v = X_c[:, j]$
    Compute bandwidth $h = \text{Silverman}(v)$
    Initialize Kernel Density Estimator $KDE_{c,j}$ with kernel='gaussian', bandwidth=h
    Fit $KDE_{c,j}$ on v {Builds tree index structure}
    Store $KDE_{c,j}$ in $\mathcal{M}$
  end for
end for
return $\mathcal{M}$
Procedure PREDICTPROBA(X_{test})
$N_{test} \leftarrow$ samples in X_{test}
Initialize log-probability matrix $L \in \mathbb{R}^{N_{test} \times |C|}$ with $-\infty$
for each sample $i \in \{1, \dots, N_{test}\}$ **do**
  for each class $c \in C$ **do**
    $L[i, c] \leftarrow \log(\pi_c)$
    for each feature $j \in \{1, \dots, F\}$ **do**
      Retrieve $KDE_{c,j}$ from $\mathcal{M}$
      $\log p \leftarrow KDE_{c,j}.\text{score\_samples}(X_{test}[i, j])$
      Clip $\log p$ to range $[-700, 700]$ {Stability check}
      $L[i, c] \leftarrow L[i, c] + \log p$
    end for
  end for
  Apply LogSumExp to normalize $L[i, :]$ into probabilities $P[i, :]$
end for
return P

B.2 RFF Naive Bayes

Algorithm 2 RFFNaiveBayes: Training and Inference

Require: Training data $X_{train} \in \mathbb{R}^{N_{train} \times F}$, Labels y_{train}, Components D
Ensure: Fitted Model $\mathcal{M}_{RFF}$, Predictions P

Procedure FIT(X_{train}, y_{train})
Identify unique classes $C = \text{unique}(y_{train})$
for each class $c \in C$ **do**
  Extract samples $X_c = \{x \in X_{train} \mid y = c\}$
  Calculate prior $\pi_c = |X_c|/N_{train}$
  for each feature $j \in \{1, \ldots, F\}$ **do**
    Extract 1D feature vector $v = X_c[:, j]$
    Compute bandwidth $h = \text{Silverman}(v)$
    Set $\gamma = 1/(2h^2)$
    Initialize RFF Sampler $\phi_{c,j}$ with γ, D, random_state=$(seed + j)$
    Transform data: $Z = \phi_{c,j}$.fit_transform(v) $\{Z \in \mathbb{R}^{|X_c| \times D}\}$
    Compute Mean Embedding: $\mu_{c,j} = \frac{1}{|X_c|} \sum_{k=1}^{|X_c|} Z[k]$
    Store $(\phi_{c,j}, \mu_{c,j})$ in $\mathcal{M}_{RFF}$
  end for
end for
return $\mathcal{M}_{RFF}$
Procedure PREDICTPROBA(X_{test})
Initialize log-probability matrix L with $-\infty$
for each sample $i \in \{1, \ldots, N_{test}\}$ **do**
  for each class $c \in C$ **do**
    $L[i, c] \leftarrow \log(\pi_c)$
    for each feature $j \in \{1, \ldots, F\}$ **do**
      Retrieve $(\phi_{c,j}, \mu_{c,j})$ from $\mathcal{M}_{RFF}$
      Transform test point: $z_{test} = \phi_{c,j}$.transform($X_{test}[i, j]$)
      Estimate Density: $\hat{p} = z_{test} \cdot \mu_{c,j}$ {Dot Product}
      $\hat{p} \leftarrow \max(\hat{p}, 10^{-9})$ {Clip negative noise}
      $L[i, c] \leftarrow L[i, c] + \log(\hat{p})$
    end for
  end for
  Apply LogSumExp to normalize
end for
return P

B.3 Gaussian Naive Bayes

Algorithm 3 GaussianNaiveBayes: Training and Inference

Require: Training data $X_{train} \in \mathbb{R}^{N_{train} \times F}$, Labels y_{train}
Ensure: Fitted Model $\mathcal{M}_{GNB}$, Predictions P

Procedure FIT(X_{train}, y_{train})
Identify unique classes $C = \text{unique}(y_{train})$
for each class $c \in C$ **do**
  Extract samples $X_c = \{x \in X_{train} \mid y = c\}$
  Calculate prior $\pi_c = |X_c|/N_{train}$
  for each feature $j \in \{1, \ldots, F\}$ **do**
    Calculate Mean: $\mu_{c,j} = \frac{1}{|X_c|} \sum_{x \in X_c} x_j$
    Calculate Variance: $\sigma^2_{c,j} = \frac{1}{|X_c|} \sum_{x \in X_c} (x_j - \mu_{c,j})^2$
    Store $(\mu_{c,j}, \sigma^2_{c,j})$ in $\mathcal{M}_{GNB}$
  end for
end for
return $\mathcal{M}_{GNB}$
Procedure PREDICTPROBA(X_{test})
Initialize log-probability matrix L with $-\infty$
for each sample $i \in \{1, \ldots, N_{test}\}$ **do**
  for each class $c \in C$ **do**
    $L[i, c] \leftarrow \log(\pi_c)$
    for each feature $j \in \{1, \ldots, F\}$ **do**
      Retrieve $(\mu_{c,j}, \sigma^2_{c,j})$ from $\mathcal{M}_{GNB}$
      $\hat{p} = \frac{1}{\sqrt{2\pi\sigma^2_{c,j}}} \exp\left(-\frac{(X_{test}[i,j] - \mu_{c,j})^2}{2\sigma^2_{c,j}}\right)$
      $L[i, c] \leftarrow L[i, c] + \log(\hat{p})$
    end for
  end for
  Apply LogSumExp to normalize
end for
return P

C Test on Real Dataset: MAGIC Gamma Telescope

For real-world evaluation, we utilize the MAGIC Gamma Telescope dataset obtained from the UCI Machine Learning Repository [3]. This dataset simulates the registration of high-energy gamma particles in a ground-based atmospheric Cherenkov gamma telescope. We selected this dataset because it exhibits: (1) high-dimensional feature interactions (similar to *Medium*/*Hard* synthetic), (2) real class imbalance, and (3) non-Gaussian distributions, providing realistic validation beyond controlled synthetic experiments.

The task is binary classification: distinguishing between gamma rays (signal) and hadronic showers (background). The dataset contains 19,020 instances with 10 continuous features. The dataset is characterized by overlapping class distributions and class imbalance.

Table 4: Characteristics of the MAGIC Gamma Telescope Dataset

Property	**Variable**	**Value**
Number of observations	N	19,020
Number of features	F	10
Feature types	continuous	Real-valued
Number of classes	C	2 (Gamma vs. Hadron)
Class distribution	imbalanced	Gamma: $\sim 65\%$, Hadron: $\sim 35\%$
Missing values	percentage	0%

C.1 MAGIC Gamma Exploratory Data Analysis

Figure 12 displays the pair-plot for the MAGIC Gamma Telescope dataset. The data exhibits high complexity with significant class overlap across most feature combinations. The distributions are clearly non-Gaussian, often exhibiting skewness or long tails characteristic of high-energy physics measurements. No missing data are observed since data comes from simulations.

To identify the most discriminative features, we analyzed the feature-target correlations and the correlation matrix, shown in Figure 11. We observe that some features show stronger correlations with the target class (e.g., the top-right block), suggesting they contain higher information gain. However, no single feature provides a clear cut, reinforcing the need for a multivariate approach capable of capturing complex probability densities. Notably, the correlation matrix reveals substantial inter-feature dependencies, with many coefficients $|\rho| > 0.5$, strongly violating the conditional independence assumption shared by all our Naive Bayes variants (Gaussian, KDE, RFF-KDE).

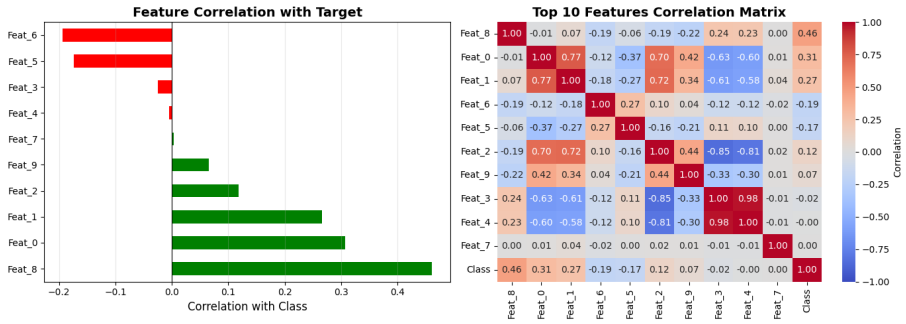

Figure 11: Feature-Target correlation of MAGIC Gamma Telescope dataset. The magnitude indicates the linear discriminative power of each feature.

C.2 Real Dataset Results

The analysis extends to the real-world dataset (Figure 13), which serves as a practical validation of the trends observed in the synthetic experiments. This dataset presents a complexity profile that effectively lies between the *Medium* and *Hard* synthetic regimes, offering a mixture of well-separated features and complex, non-linear dependencies. The MAGIC dataset was split using

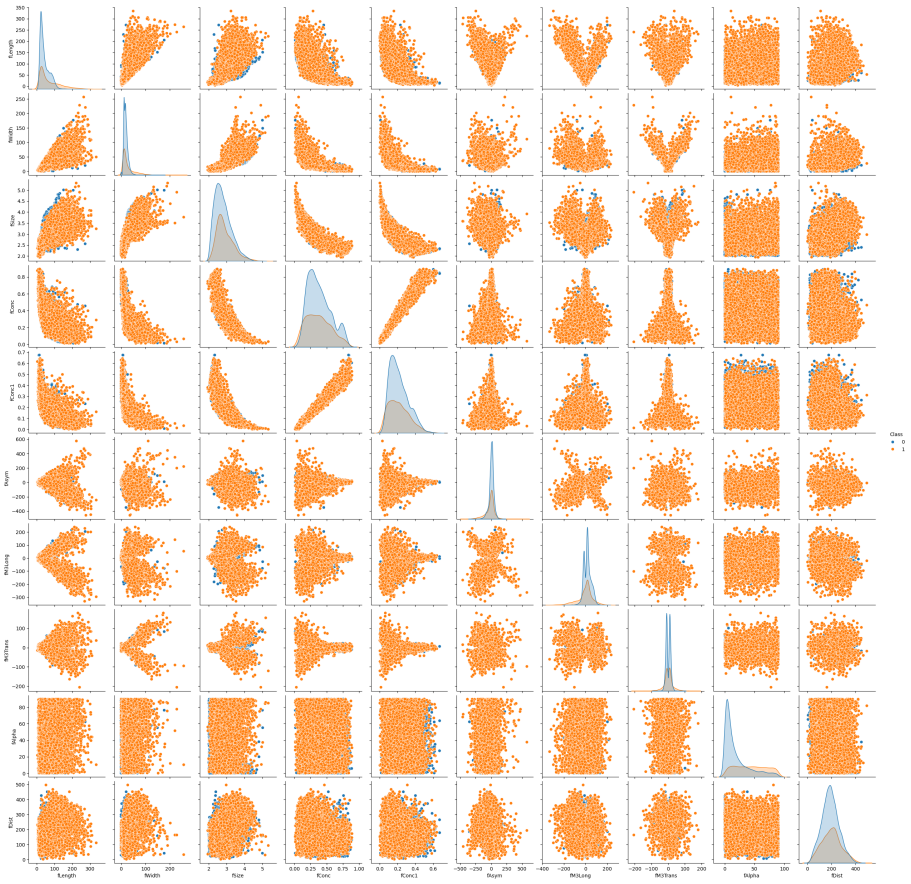

Figure 12: Pair-plot of variables from MAGIC Gamma Telescope dataset. The classes show significant overlap and non-Gaussian distributions.

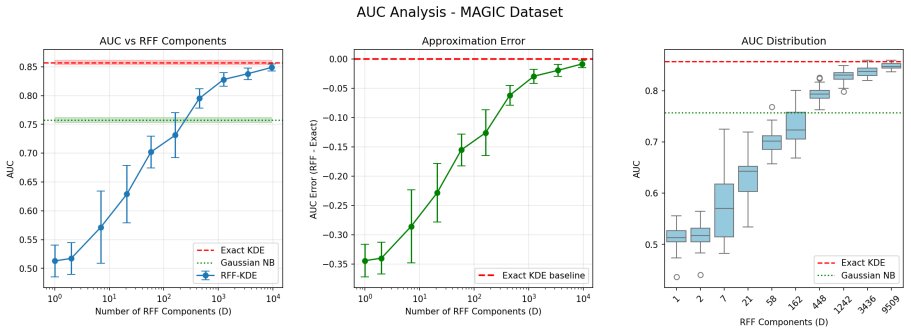

Figure 13: AUC analysis for RFF-KDE on real dataset (log scale for D). Shaded regions and error bars indicate $\pm$ standard deviation across multiple runs.

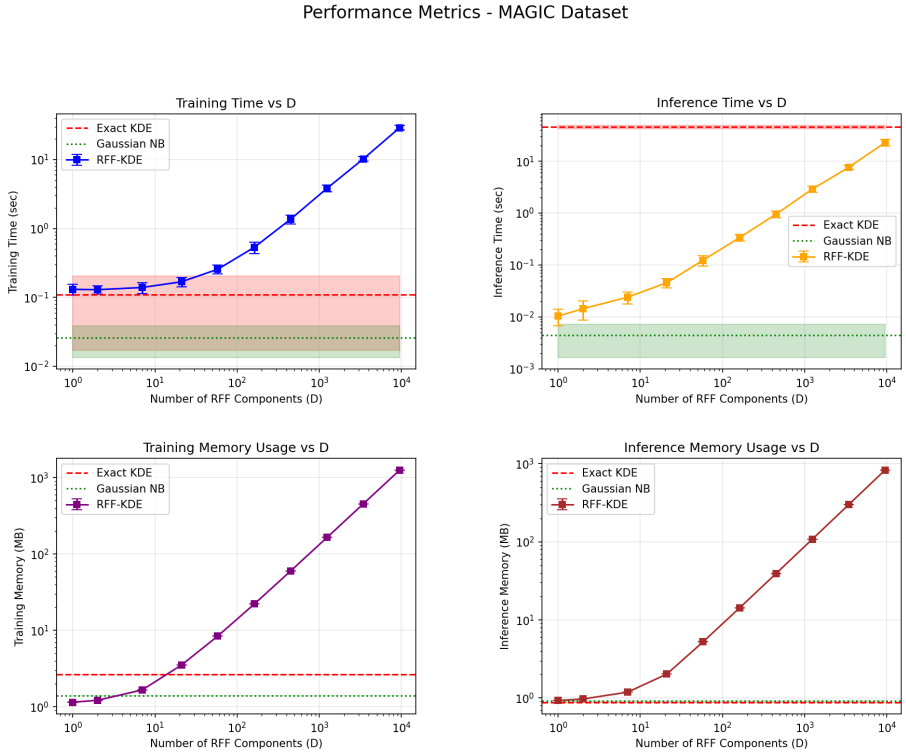

Figure 14: Computational performance analysis for RFF-KDE on real dataset with logarithmic scaling on both axes. Shaded regions and error bars indicate ± standard deviation across multiple runs.

stratified 70-30 sampling as for the synthetic datasets and mean AUC reported across 20 random seeds. Table 6 in Appendix D shows the full results for the synthetic dataset.

The "AUC vs RFF Components" plot illustrates the limitations of the parametric baseline in a real-world setting. The Gaussian NB achieves a respectable but suboptimal AUC of ≈ 0.76, significantly trailing the Exact KDE baseline of ≈ 0.86. This gap confirms that while the data is not as intractable as the *Hard* synthetic case, substantial non-Gaussian structures exist that simple models cannot capture.

Despite high correlation between features, all models maintain acceptable performance, confirming that discriminative information primarily resides in the marginal feature distributions rather than complex joint dependencies.

The RFF-KDE approximation demonstrates again a smooth, monotonic convergence towards the non-parametric gold standard. At $D \approx 100$, the model only matches the GNB baseline (AUC $\approx$ 0.73). It requires scaling to $D \approx 1000$ to fully exploit the kernel's power, reaching an AUC of ≈ 0.83, which is $\approx 96\%$ of the KDE's AUC.

The efficiency metrics (Figure 14) largely mirror the laws established in the synthetic analysis:

- Inference Speed: The linear scaling advantage of RFF remains robust. Even at the higher dimensionalities required for this dataset ($D \approx 1000$), the RFF inference time remains roughly an order of magnitude faster than the Exact KDE.

- Resource Costs: The training time and memory plots exhibit the expected growth. Notably, the trade-off remains favorable: we exchange an increase in RAM for a constant, low-latency inference capability.

D Appendix: Complete results for AUC performances

Table 5: Performance comparison across difficulty levels (AUC ± standard deviation over 20 runs)

Method	**Easy**	**Medium**	**Hard**
Gaussian NB	1.000 ± 0.000	0.880 ± 0.133	0.478 ± 0.044
Exact KDE	1.000 ± 0.000	0.927 ± 0.063	0.809 ± 0.061
RFF-KDE ($D = 1$)	0.644 ± 0.135	0.538 ± 0.069	0.560 ± 0.044
RFF-KDE ($D = 2$)	0.817 ± 0.161	0.603 ± 0.100	0.587 ± 0.048
RFF-KDE ($D = 5$)	0.928 ± 0.101	0.613 ± 0.098	0.656 ± 0.055
RFF-KDE ($D = 11$)	0.997 ± 0.005	0.723 ± 0.128	0.690 ± 0.055
RFF-KDE ($D = 25$)	0.999 ± 0.002	0.792 ± 0.098	0.708 ± 0.050
RFF-KDE ($D = 58$)	1.000 ± 0.000	0.843 ± 0.082	0.737 ± 0.052
RFF-KDE ($D = 131$)	1.000 ± 0.000	0.868 ± 0.073	0.751 ± 0.055
RFF-KDE ($D = 295$)	1.000 ± 0.000	0.885 ± 0.064	0.767 ± 0.054
RFF-KDE ($D = 665$)	1.000 ± 0.000	0.899 ± 0.070	0.775 ± 0.055
RFF-KDE ($D = 1500$)	1.000 ± 0.000	0.907 ± 0.066	0.785 ± 0.058

Table 6: Performance comparison on MAGIC Gamma Telescope dataset (AUC ± standard deviation over 20 runs)

Method	**MAGIC**
Gaussian NB	0.757 ± 0.004
Exact KDE	0.857 ± 0.004
RFF-KDE ($D = 1$)	0.513 ± 0.028
RFF-KDE ($D = 2$)	0.517 ± 0.027
RFF-KDE ($D = 7$)	0.571 ± 0.062
RFF-KDE ($D = 21$)	0.629 ± 0.050
RFF-KDE ($D = 58$)	0.702 ± 0.027
RFF-KDE ($D = 162$)	0.731 ± 0.039
RFF-KDE ($D = 448$)	0.795 ± 0.017
RFF-KDE ($D = 1242$)	0.828 ± 0.012
RFF-KDE ($D = 3436$)	0.838 ± 0.010
RFF-KDE ($D = 9509$)	0.849 ± 0.006

E TOST Equivalence Results

E.1 Margin Results $\delta = 0.02$

Table 7: TOST Results ($\delta = \pm 0.02$): Minimum D for Equivalence by Difficulty

Difficulty	Min D	Mean Δ	90% CI	Cohen's d	Adj. p
Easy	11	0.0029	[0.0024, 0.0035]	0.586	0.000
Medium	None	0.0208 ($D = 1500$)	[0.0196, 0.0220]	2.008	1.000
Hard	None	0.0241 ($D = 1500$)	[0.0230, 0.0251]	2.671	1.000

Table 8: Full TOST Results ($\delta = \pm 0.02$, Bootstrap, $n = 200$/D): Mean ΔAUC, 90% CI, Decision

Difficulty	D	Mean Δ	SE	90% CI	Cohen's d	Decision
	1	0.356	0.010	[0.340, 0.372]	2.64	×
	11	0.0029	0.0004	[0.0024, 0.0035]	0.59	✓
	25	0.0006	0.0002	[0.0003, 0.0008]	0.26	✓
	1500	0.0000	0.0000	[0.0000, 0.0000]	0.23	✓
Easy	295	0.0420	0.0016	[0.0393, 0.0447]	1.82	×
	665	0.0285	0.0010	[0.0268, 0.0302]	1.99	×
	1500	0.0208	0.0007	[0.0196, 0.0220]	2.01	×
	295	0.0422	0.0010	[0.0406, 0.0438]	3.02	×
Medium	665	0.0336	0.0009	[0.0320, 0.0351]	2.53	×
	1500	0.0241	0.0006	[0.0230, 0.0251]	2.67	×

✓: EQUIV; ×: NOT equiv (non-normal data across all)

Hard

E.2 Margin Results $\delta = 0.05$

Table 9: TOST Results ($\delta = \pm 0.05$): Minimum D for Equivalence by Difficulty

Difficulty	Min D	Mean Δ	90% CI	Cohen's d	Adj. p
Easy	11	0.0029	[0.0024, 0.0035]	0.586	0.000
Medium	295	0.0420	[0.0393, 0.0447]	1.816	0.000
Hard	295	0.0422	[0.0406, 0.0438]	3.019	0.000

Table 10: Full TOST Results ($\delta = \pm 0.05$, Bootstrap, $n = 200/\text{D}$): Selected D Values

Difficulty	D	Mean Δ	SE	90% CI	Cohen's d	Decision
Easy	1	0.356	0.010	[0.340, 0.372]	2.64	×
	11	0.0029	0.0004	[0.0024, 0.0035]	0.59	✓
	25	0.0006	0.0002	[0.0003, 0.0008]	0.26	✓
	1500	0.0000	0.0000	[0.0000, 0.0000]	0.23	✓
Medium	131	0.0597	0.0023	[0.0558, 0.0635]	1.81	×
	295	0.0420	0.0016	[0.0393, 0.0447]	1.82	✓
	665	0.0285	0.0010	[0.0268, 0.0302]	1.99	✓
	1500	0.0208	0.0007	[0.0196, 0.0220]	2.01	✓
Hard	131	0.0573	0.0016	[0.0547, 0.0599]	2.61	×
	295	0.0422	0.0010	[0.0406, 0.0438]	3.02	✓
	665	0.0336	0.0009	[0.0320, 0.0351]	2.53	✓
	1500	0.0241	0.0006	[0.0230, 0.0251]	2.67	✓

✓: EQUIV; ×: NOT equiv (non-normal data across all)

E.3 Margin Comparison $\delta = 0.02$ vs $\delta = 0.05$

Table 11: Minimum D for Equivalence: ± 0.05 vs ± 0.02

Difficulty	± 0.05 Min D	Δ (± 0.05)	± 0.02 Min D	Final Rec.
Easy	11	0.0026	11	$D = 11$
Medium	295	0.040	None	$D = 295$
Hard	295	0.039	None	$D = 295$

F Appendix: Formal Derivation of Computational Complexity

This appendix details the derivation of time and space complexities for the evaluated algorithms. Let N be the total number of samples, split into training size $N_{train} = \alpha N$ and test size $N_{test} = (1-\alpha)N$, where $0 < \alpha < 1$. Let F denote the number of features and C the number of classes.

F.1 KDE Naive Bayes (KDE-NB)

F.1.1 Training Complexity

The training phase for KDE-NB requires two primary operations per feature per class:

1. **Bandwidth Selection:** Using robust rules of thumb (e.g., Silverman's rule) or cross-validation often requires sorting or calculating interquartile ranges, costing $\mathcal{O}(N_{train} \log N_{train})$.
2. **Index Construction:** To optimize density queries, tree-based structures (e.g., k-d trees or ball trees) are constructed. Building a tree on scalar data takes $\mathcal{O}(N_{train} \log N_{train})$.

Summing over F features and C classes (absorbed into the constant for asymptotic analysis w.r.t N):

$$T_{train}^{KDE} \approx \mathcal{O}(F \cdot N_{train} \log N_{train}) = \mathcal{O}(F \cdot N \log N)$$

F.1.2 Inference Complexity

For Exact KDE, the probability density function for a single query point x_q is estimated by summing the contributions of a kernel $K(\cdot)$ over all training points:

$$\hat{f}(x_q) = \frac{1}{N_{train} h} \sum_{i=1}^{N_{train}} K\left(\frac{x_q - x_i}{h}\right)$$

This summation requires $\mathcal{O}(N_{train})$ operations per query point per feature. Given N_{test} query points:

$$T_{infer}^{KDE} = N_{test} \cdot F \cdot \mathcal{O}(N_{train}) = \mathcal{O}((1-\alpha)N \cdot F \cdot \alpha N) = \mathcal{O}(F \cdot N^2)$$

This quadratic complexity $\mathcal{O}(N^2)$ makes exact KDE prohibitive for large datasets.

F.1.3 Space Complexity

KDE is a lazy learning algorithm that requires retaining the entire training dataset in memory.

$$M^{KDE} = \mathcal{O}(F \cdot N_{train}) = \mathcal{O}(F \cdot N)$$

F.2 KDE Naive Bayes with Random Fourier Features (RFF-NB)

F.2.1 Training Complexity

RFF approximates the kernel by mapping input vectors $x \in \mathbb{R}^F$ to a randomized feature space $z(x) \in \mathbb{R}^{2D}$.

1. **Projection:** Mapping all N_{train} samples involves a matrix multiplication of size $(N_{train} \times F)$ by $(F \times D)$, costing $\mathcal{O}(N_{train} \cdot F \cdot D)$.
2. **Parameter Estimation:** Computing the mean and variance of the projected features for each class takes $\mathcal{O}(N_{train} \cdot D)$.

The dominant term is the projection:

$$T_{train}^{RFF} = \mathcal{O}(N \cdot F \cdot D)$$

F.2.2 Inference Complexity

Inference requires mapping the N_{test} samples to the RFF space and then evaluating the Gaussian likelihood in that space.

1. **Projection:** $\mathcal{O}(N_{test} \cdot F \cdot D)$.
2. **Likelihood Evaluation:** $\mathcal{O}(N_{test} \cdot D \cdot C)$.

Assuming $F \approx D$ or $D > C$, the projection cost dominates:

$$T_{infer}^{RFF} = \mathcal{O}((1-\alpha)N \cdot F \cdot D) = \mathcal{O}(N \cdot F \cdot D)$$

F.2.3 Space Complexity

Unlike KDE, RFF-NB does not store training data. It stores:

- The random projection weights: $\mathcal{O}(F \cdot D)$.
- The learned means and variances for C classes in D dimensions: $\mathcal{O}(C \cdot D)$.

The total model size is independent of N:

$$M_{model}^{RFF} = \mathcal{O}(C \cdot F \cdot D)$$

However, during execution, memory is required to store the projected batch of data, scaling as $\mathcal{O}(N \cdot D)$.

F.3 Gaussian Naive Bayes (GNB)

F.3.1 Training Complexity

GNB performs a single pass over the data to compute maximum likelihood estimates (mean and variance) for each feature and class.

$$T_{train}^{GNB} = \mathcal{O}(N_{train} \cdot F \cdot C) = \mathcal{O}(F \cdot N)$$

F.3.2 Inference Complexity

Inference involves evaluating the univariate Gaussian PDF for each feature of each test sample.

$$T_{infer}^{GNB} = \mathcal{O}(N_{test} \cdot F \cdot C) = \mathcal{O}(F \cdot N)$$

F.3.3 Space Complexity

The model only stores the summary statistics (μ, σ^2) for each class and feature.

$$M_{model}^{GNB} = \mathcal{O}(C \cdot F)$$

This represents the minimal memory footprint among the compared algorithms.

G Implementation Details

The experimental framework was developed in `Python 3`, structured as a modular package to ensure extensibility and separation of concerns. The source code is organized into specific modules: `Data.py` for synthetic generation via copulas; `Classifiers.py` for the custom estimator definitions; `Stat_tests.py` for the equivalence testing logic; and `Executions.py` for orchestrating parallel experimental runs.

G.1 Software Environment

We rely on the standard scientific Python ecosystem. Specific library dependencies include:

- **Core Computing:** `numpy` and `pandas` for vectorized operations and result aggregation.
- **Machine Learning:** `scikit-learn` serves as the foundation for our custom classes. We utilize `BaseEstimator` and `ClassifierMixin` to ensure pipeline compatibility, `StandardScaler` for preprocessing, and `roc_auc_score` for evaluation.
- **Distributions:** The synthetic data generation pipeline (`Data.py`) utilizes specific distributions from `scipy.stats`, including `norm`, `uniform`, `laplace`, and `dweibull`, to model non-Gaussian features.
- **Statistical Analysis:** The Two One-Sided Tests (TOST) and subsequent Holm-Bonferroni corrections are implemented using `statsmodels`.
- **Parallelization:** To efficiently handle the multiple-seed validation, we employ `joblib` and `multiprocessing`, dynamically scaling to $\max(1, \text{cpu\_count}() - 1)$ cores.

The exact version of every package is listed inside the `requirements.txt` file.

G.2 Hyperparameter Definitions

While Section 5.1 outlines the range of the Random Fourier Features (D), we detail here the exact logarithmic grids used to ensure rigorous reproducibility.

For the synthetic experiments, the set of components $\mathcal{D}_{\text{synth}}$ is defined as 10 logarithmically spaced integers:

$$\mathcal{D}_{\text{synth}} = \{1, 2, 5, 11, 25, 58, 131, 295, 665, 1500\} \tag{10}$$

This grid was generated using $np.logspace(0, \log_{10}(1500), 10)$.

For the MAGIC Gamma dataset ($N = 19,020$), the grid extends to approximately $N/2$ to test the high-dimensional limit:

$$\mathcal{D}_{\text{real}} = \{1, 2, 7, 21, 58, 162, 448, 1242, 3436, 9509\} \tag{11}$$

G.3 Randomness Control

To guarantee that the reported mean and standard deviations are robust to initialization noise (both in data splitting and RFF weight sampling), we fixed a specific set of 20 integers as random seeds. All experiments iterate deterministically over this set $\mathcal{S}$:

$$\begin{aligned}\mathcal{S} = \{&42, 123, 456, 789, 1024, 2048, 4096, 8192, 271828, 314159,\\ &161803, 999, 5555, 7777, 12345, 54321, 13579, 24680, 87654, 11111\}\end{aligned} \tag{12}$$