Comparison of Naïve Bayes and K-Nearest Neighbors for Spam Email Detection

Sai Geetha M
3122235001109
Department of Computer Science
SSN College Of Engineering
Email: saigeetha2310537@ssn.edu.in

***Abstract*—This work implements and compares Naïve Bayes and K-Nearest Neighbors (KNN) classifiers on the Spambase benchmark dataset for binary spam email detection. Three Naïve Bayes variants (Gaussian, Multinomial, Bernoulli) and a KNN classifier with hyperparameter tuning are evaluated using accuracy, precision, recall, F1 score, specificity and false positive rate. Extensive visualizations include exploratory data analysis, confusion matrices, receiver operating characteristic (ROC) curves and the evolution of training and validation accuracy as a function of the KNN neighborhood size. We analyze overfitting and underfitting behavior, discuss the bias–variance characteristics of both model families, and assess the statistical significance of performance differences using the Friedman test. Results show that tuned KNN with Manhattan distance and distance-based weighting outperforms all Naïve Bayes variants in terms of F1 score, while Bernoulli Naïve Bayes is the most competitive probabilistic baseline.**

***Index Terms*—Spam classification, Naïve Bayes, K-Nearest Neighbors, Hyperparameter tuning, Bias–variance, Spambase.**

I. Introduction

Spam email filtering is a classic binary classification problem where messages must be classified as spam or non-spam. Reliable spam detection requires models that are accurate while controlling both false negatives (missed spam) and false positives (legitimate email incorrectly flagged as spam). This work compares two standard supervised learning families on the Spambase dataset: Naïve Bayes and K-Nearest Neighbors (KNN).

Naïve Bayes is a probabilistic classifier that assumes conditional independence between features given the class label. It is simple, efficient for high-dimensional data and often performs surprisingly well on text-like data. Different variants (Gaussian, Multinomial, Bernoulli) correspond to different assumptions about feature distributions.

KNN is an instance-based learning algorithm that classifies a new point by looking at the labels of its nearest neighbors in feature space. KNN is strongly affected by feature scaling, by the choice of distance metric and by hyperparameters such as the number of neighbors and the weighting scheme.

In addition to implementing these models, this study investigates hyperparameter tuning, neighbor search strategies (KDTree vs. BallTree), overfitting and underfitting behavior, and bias–variance trade-offs. Evaluation is based on multiple metrics, with a particular focus on F1 score due to class imbalance and the asymmetric cost of misclassification in spam filtering.

II. Aim and Objective

The aim of this experiment is to systematically compare Naïve Bayes and KNN classifiers on a binary spam detection task and to analyze their performance and generalization behavior.

The specific objectives are:

- To implement Gaussian, Multinomial and Bernoulli Naïve Bayes classifiers and a KNN classifier for the Spambase dataset.
- To perform appropriate preprocessing, including feature scaling required for distance-based methods.
- To evaluate each model using accuracy, precision, recall, F1 score, specificity, false positive rate, and computational time.
- To visualize class distribution, feature distributions, confusion matrices, ROC curves and the relationship between KNN accuracy and the number of neighbors.
- To tune KNN hyperparameters using grid search and randomized search and compare KDTree and BallTree neighbor search algorithms.
- To study overfitting and underfitting for different K values and to interpret the bias–variance characteristics of Naïve Bayes vs. KNN.
- To apply the Friedman test to check whether performance differences between models are statistically significant.

III. Dataset Description

The experiments use the Spambase dataset, a benchmark dataset for spam detection.

A. Basic Properties

The dataset properties (from the notebook) are:

- Number of instances: 4601 emails.
- Number of attributes: 58.
- Feature types: 55 numerical frequency features (word and character frequencies, capitalization statistics) and 3 integer attributes.
- Target variable: a binary class label `class` with values 0 (non-spam) and 1 (spam).

The class distribution obtained from the data is approximately:

- Non-spam (class 0): around 60.6%.
- Spam (class 1): around 39.4%.

The mean of the class variable is approximately 0.394, which indicates moderate class imbalance. This motivates the use of F1 score, recall and specificity in addition to accuracy.

B. Feature Description

Features include:

- Word frequency attributes such as `word_freq_make`, `word_freq_address`, `word_freq_free`, etc., capturing how often certain indicative words appear.
- Character frequency attributes such as `char_freq_%21` ("!"), `char_freq_%24` ("$"), etc.
- Capitalization-related attributes: `capital_run_length_average`, `capital_run_length_longest`, `capital_run_length_total`.

These features encode both lexical properties and formatting patterns that are characteristic of spam messages, such as excessive capital letters or special characters.

IV. Preprocessing Steps

The following preprocessing steps are implemented in the notebook:

A. Data Loading

The CSV file `spambase_csv_Kaggle.csv` is loaded into a Pandas DataFrame. Basic data inspection using `info()` and `describe()` confirms that:

- There are 4601 rows and 58 columns.
- There are no missing values; all features are fully observed.
- 55 features are `float64` and 3 (including the target) are `int64`.

B. Feature–Target Split

The feature matrix and target vector are defined as:

- X: all columns except `class`.
- y: the `class` column (0 or 1).

C. Feature Scaling

A `StandardScaler` is fitted on X and used to produce a scaled feature matrix X_{scaled}.

Scaling is essential for KNN because it is a distance-based algorithm. Without scaling, features with larger numeric ranges would dominate the Euclidean or Manhattan distance, leading to misleading similarity measures. Although Naïve Bayes does not strictly require scaling, we use the scaled features consistently to ensure comparability across models and because KNN is part of the pipeline.

D. Train–Test Split

The dataset is split as follows:

- 80% for training and 20% for testing.
- Stratified splitting on y to preserve the class distribution in both training and test sets.
- `random_state=42` for reproducibility.

This split ensures that evaluation metrics reflect generalization to unseen data and are not biased toward the majority class.

V. Implementation Details

This section explains the modeling and evaluation setup, including the rationale behind metric choices.

A. Evaluation Function and Metrics

An `evaluate` function is implemented to compute:

- Accuracy.
- Precision.
- Recall.
- F1 score.
- Specificity.
- False positive rate (FPR).

All metrics are computed from the confusion matrix. Specificity and FPR are defined as:

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}}, \quad \text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}.$$

Why F1 score is used: In spam detection, both precision and recall are important. A classifier with high accuracy but low recall could miss many spam messages, while a classifier with high recall but low precision may misclassify legitimate emails as spam. The F1 score (harmonic mean of precision and recall) balances these two quantities and is more informative than accuracy under class imbalance.

B. Naïve Bayes Models

Three Naïve Bayes variants are implemented:

- Gaussian Naïve Bayes (`GaussianNB`) for continuous, approximately Gaussian features.
- Multinomial Naïve Bayes (`MultinomialNB`) for count-like nonnegative features.
- Bernoulli Naïve Bayes (`BernoulliNB`) for binary or thresholded features.

For the Multinomial variant, the notebook uses $|X|$ to ensure nonnegative input. For each variant:

- The model is trained on the training set.
- Predictions are made on the test set.
- Training time and prediction time are measured.
- The confusion matrix is visualized using a heatmap.
- ROC curves and AUC are computed.

C. Baseline KNN

A baseline KNN model with $k = 5$ neighbors is fitted:

- Distance metric: default (Minkowski, which reduces to Euclidean).
- Weights: uniform.

The baseline KNN serves as a starting point before hyperparameter tuning and neighbor search optimization.

D. Advanced KNN and Hyperparameter Tuning

Hyperparameter tuning for KNN is performed using:

- **Grid search** over:
  - `n_neighbors`: 1 to 50.
  - `weights`: `uniform`, `distance`.
  - `metric`: `euclidean`, `manhattan`, `minkowski`.
- **Randomized search** over the same grid, with 30 sampled combinations.

Both searches use 5-fold cross-validation and F1 score as the scoring function. Using F1 in the hyperparameter search ensures that the chosen model balances precision and recall rather than optimizing raw accuracy.

The best hyperparameters obtained from the notebook are:

- Grid search best: `metric=manhattan`, `n_neighbors=6`, `weights=distance`.
- Randomized search best: `metric=minkowski`, `n_neighbors=11`, `weights=distance`.

The subsequent analysis focuses on the grid-search-optimal configuration, which achieves strong performance with a relatively small neighborhood size.

E. KDTree vs. BallTree

To study neighbor search methods, the tuned KNN configuration is evaluated with:

- Algorithm = `kd_tree`.
- Algorithm = `ball_tree`.

Both algorithms produce identical classification metrics but different training and prediction times, confirming that these methods primarily affect computational efficiency and do not change the decision rule of KNN for a fixed metric and k.

VI. Visualizations

This section lists the main visualizations. In the compiled report you should replace the figure filenames with the actual PNGs exported from the notebook.

A. Class Distribution

The class distribution plot confirms moderate imbalance, justifying the use of F1 score, recall and specificity.

B. Feature Distributions

Most word and character frequencies are highly skewed and sparse, which is typical for text-derived features.

C. Naïve Bayes Confusion Matrices and ROC

These plots show the trade-off between true positive and false positive rates for each Naïve Bayes variant.

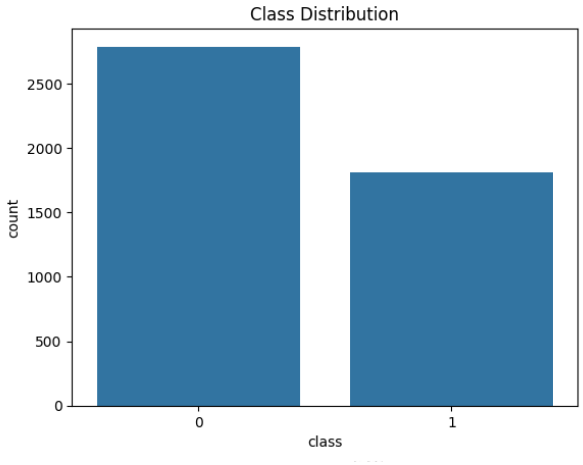

Fig. 1. Class distribution of non-spam (0) and spam (1) emails.

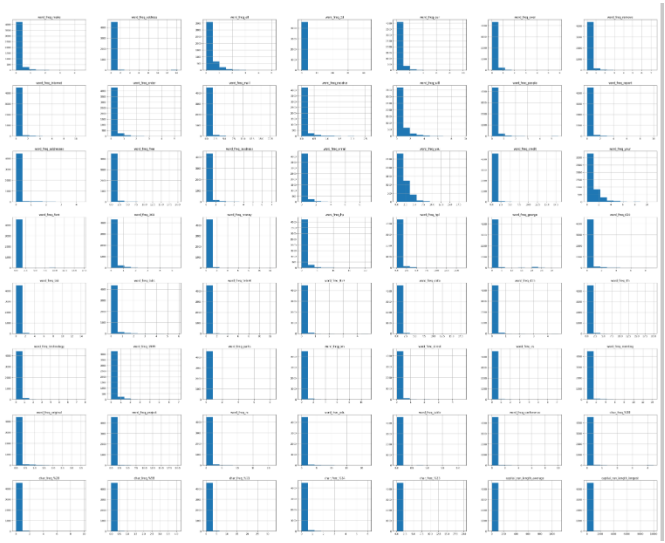

Fig. 2. Histograms of the 56 non-target features, showing skewed word and character frequency distributions.

D. KNN Confusion Matrices

E. Accuracy vs. Number of Neighbors

This plot is crucial for understanding overfitting and underfitting behavior in KNN.

VII. Performance Tables

Tables in this section summarize the numerical results obtained from the notebook.

A. Naïve Bayes Variants

Bernoulli Naïve Bayes achieves the best F1 score (0.870) and the highest accuracy (approximately 0.90), with a low false positive rate (≈ 0.059). Multinomial Naïve Bayes achieves a more conservative recall but higher specificity, while Gaussian Naïve Bayes has high recall but relatively lower precision and specificity.

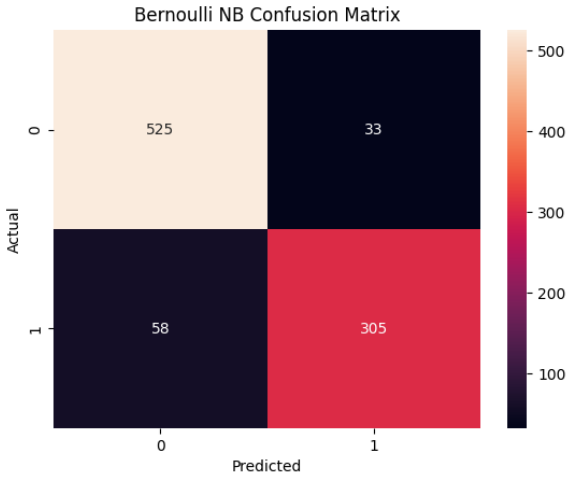

Fig. 3. Confusion matrices of best naive bayes model

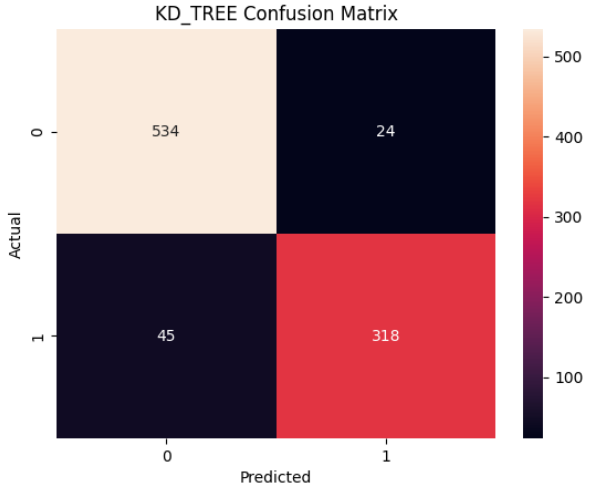

Fig. 5. Confusion matrices of KDTree (similar to BallTree.

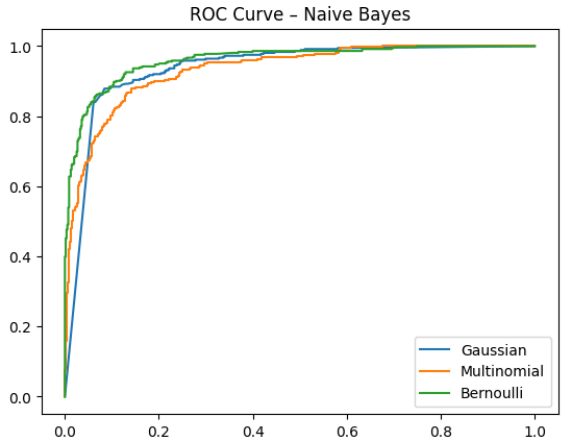

Fig. 4. ROC curves for the three Naïve Bayes variants, including AUC values.

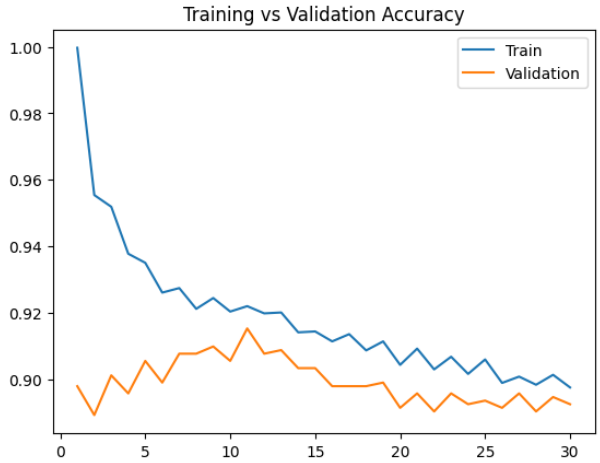

Fig. 6. Training and validation accuracy as a function of the number of neighbors k.

B. KNN with KDTree vs. BallTree

Both KDTree and BallTree produce identical classification metrics, confirming that the neighbor search structure does not change the decision rule. However, their computational profiles differ slightly: KDTree trains marginally faster, while BallTree predicts slightly faster.

C. Cross-Validated F1 Scores and Friedman Test

Using 5-fold cross-validation and F1 score, the notebook reports:

- GaussianNB mean F1: 0.812.
- BernoulliNB mean F1: 0.867.
- KNN (optimal k): 0.846.

A Friedman test comparing GaussianNB, BernoulliNB and KNN yields:

- Friedman statistic: 6.40.
- p-value: 0.0408.

Since $p < 0.05$, we reject the null hypothesis that all three models perform equivalently. This supports the conclusion that Bernoulli Naïve Bayes and tuned KNN are statistically better than Gaussian Naïve Bayes on this dataset.

VIII. Overfitting and Underfitting Analysis

Overfitting and underfitting are analyzed primarily through the KNN accuracy vs. k plot (Figure 6) and the behavior of Naïve Bayes models.

A. KNN: Effect of Number of Neighbors

For the KNN model:

- For very small k (e.g., $k = 1$), training accuracy is close to 1 while validation accuracy is noticeably lower. The model memorizes the training set and is highly sensitive to noise, which indicates overfitting and high variance.
- As k increases, training accuracy decreases gradually while validation accuracy initially increases, reaches a peak around $k = 6$, and then slowly decreases for very large k.

TABLE I
NAÏVE BAYES PERFORMANCE ON THE TEST SET

Model	Acc	Prec	Rec	F1	Spec	FPR
Gaussian	0.833	0.715	0.959	0.819	0.751	0.249
Multinomial	0.861	0.847	0.791	0.818	0.907	0.093
Bernoulli	0.901	0.902	0.840	0.870	0.941	0.059

Model	Train Time (s)	Pred Time (s)
Gaussian	0.0087	0.0038
Multinomial	0.0162	0.0008
Bernoulli	0.0167	0.0066

TABLE II
TUNED KNN (OPTIMAL k) WITH KDTREE AND BALLTREE

Algo	Acc	Prec	Rec	F1	Spec	FPR
kd_tree	0.925	0.930	0.876	0.902	0.957	0.043
ball_tree	0.925	0.930	0.876	0.902	0.957	0.043

Algo	Train Time (s)	Pred Time (s)
kd_tree	0.0216	0.4633
ball_tree	0.0308	0.4487

- For large k (e.g., above 25–30), both training and validation accuracy are lower and closer together, indicating underfitting due to excessive smoothing.

The best compromise between overfitting and underfitting is found around $k = 6$ with distance-based weighting and the Manhattan metric, which corresponds to the grid-search-optimal configuration.

B. Naïve Bayes Models

Naïve Bayes models are relatively stable with respect to overfitting because they rely on strong conditional independence assumptions and closed-form parameter estimates rather than flexible decision boundaries. This leads to:

- Moderate training and test performance that is relatively close, suggesting lower variance.
- Potential underfitting in cases where feature dependencies are important, since the simplistic independence assumption cannot capture complex patterns present in the data.

In this experiment, Bernoulli Naïve Bayes strikes a good balance, achieving high F1 with relatively simple assumptions, while still not matching the tuned KNN performance.

IX. BIAS–VARIANCE ANALYSIS

Bias–variance trade-offs are assessed qualitatively from model structures and empirically from cross-validation results.

A. Naïve Bayes

Naïve Bayes has:

- **High bias:** The conditional independence assumption and parametric form restrict the hypothesis space. The model cannot represent complex decision boundaries.
- **Low variance:** Parameter estimates are relatively stable across different training samples. Cross-validated F1 scores show moderate variation, with Bernoulli Naïve Bayes achieving an average F1 of approximately 0.867.

This high-bias, low-variance behavior is often desirable in high-dimensional problems with limited data, as it reduces overfitting. However, it may underfit when the true relationships between features and labels are more complex than the model can represent.

B. K-Nearest Neighbors

KNN exhibits the opposite behavior:

- **Low bias:** With small k, KNN can approximate complex, highly nonlinear decision boundaries. This reduces bias but can lead to overfitting.
- **High variance:** Predictions can change substantially with small variations in the training set, especially when k is small. This is visible in the large gap between training and validation accuracy for small k.

By increasing k and using distance-based weighting, we reduce variance and increase bias. The optimal $k = 6$ with distance weighting and Manhattan distance provides a good balance, giving a cross-validated mean F1 of around 0.846 and a test-set F1 of approximately 0.902.

C. Neighbor Search Methods

KDTree and BallTree alter the computational complexity of neighbor search but do not change the KNN decision rule for a given metric and k. The identical performance metrics in Table II confirm that:

- The bias–variance profile of KNN is determined by k, the weighting scheme and the distance metric.
- KDTree and BallTree mainly affect training and prediction time, not bias or variance.

X. OBSERVATIONS AND CONCLUSION

The main observations from this study are:

- The Spambase dataset has 4601 instances, 58 features and moderate class imbalance (around 39.4% spam).
- Preprocessing with standardization is essential for KNN and does not harm Naïve Bayes.
- Among Naïve Bayes variants, Bernoulli Naïve Bayes achieves the best overall performance with test-set F1 ≈ 0.870, combining high precision and reasonably high recall and specificity.
- A tuned KNN with Manhattan distance, distance-based weighting and $k = 6$ neighbors achieves the best performance overall, with test-set accuracy ≈ 0.925 and F1 ≈ 0.902.
- KDTree and BallTree neighbor search methods yield identical classification metrics, confirming that they are purely computational optimizations.
- The KNN accuracy vs. k curve illustrates classical overfitting for small k and underfitting for very large k, with a clear sweet spot in the intermediate range.
- Naïve Bayes models are high-bias, low-variance; they are robust but may underfit. KNN is low-bias, high-variance;

hyperparameter tuning is necessary to control overfitting and underfitting.

- The Friedman test shows that performance differences between GaussianNB, BernoulliNB and tuned KNN are statistically significant at the 5% level, supporting the conclusion that BernoulliNB and tuned KNN outperform GaussianNB.

In conclusion, for this spam detection task, a carefully tuned KNN classifier outperforms simple Naïve Bayes baselines in terms of F1 score and overall accuracy, at the cost of higher computational time. Bernoulli Naïve Bayes remains a strong baseline that is extremely fast to train and predict, making it attractive in resource-constrained settings. The analysis also highlights the importance of hyperparameter tuning, proper evaluation metrics and an explicit understanding of overfitting, underfitting and bias–variance trade-offs in practical machine learning workflows.