Sri Sivasubramaniya Nadar College of Engineering, Chennai

(An autonomous Institution affiliated to Anna University)

Machine Learning Algorithms Laboratory

Experiment 3: Email Spam or Ham Classification using Naïve Bayes, KNN, and SVM

Subject Code & Name:	ICS1512 & Machine Learning Algorithms Laboratory
Degree & Branch:	M. Tech (Integrated) Computer Science & Engineering
Semester:	V
Academic Year:	2025-2026 (Odd)
Batch:	2023-2028

Submitted by:
Sudharshan Vijayaragavan
Reg No.: 3122237001054

1 Objective

The objective of this experiment is to classify emails as either spam or ham using three distinct classification algorithms: Naïve Bayes, K-Nearest Neighbors (KNN), and Support Vector Machine (SVM). The performance of these models will be evaluated using standard accuracy metrics, confusion matrices, ROC curves, and K-Fold cross-validation.

2 Dataset

The dataset used for this experiment is the Spambase dataset, which is publicly available on Kaggle. It contains a collection of extracted features from emails, pre-labeled as either spam or ham, making it suitable for classification tasks.

3 Implementation Steps

The following steps were performed for the implementation:

1. **Load and Preprocess the Dataset:** The dataset was loaded, and initial preprocessing steps were performed, including handling any missing values and normalizing the features using 'StandardScaler'.
2. **Exploratory Data Analysis (EDA):** An analysis of the dataset was conducted to understand the class balance and the distribution of features.
3. **Data Splitting:** The dataset was split into training and testing sets to evaluate model performance on unseen data.
4. **Model Training:** Models were trained using different variants for each algorithm:
   - **Naïve Bayes:** Gaussian, Multinomial, and Bernoulli variants were trained.
   - **K-Nearest Neighbors:** The 'k' value was varied (k=3, 5, 7), and different algorithms ('KDTree', 'BallTree') were compared.
   - **Support Vector Machine:** Linear, Polynomial, RBF, and Sigmoid kernels were evaluated.
5. **Evaluation and Visualization:** The performance of each model was evaluated using a classification report, confusion matrix, and ROC curve.
6. **K-Fold Cross-Validation:** K-Fold cross-validation with $K = 5$ was performed to get a more robust estimate of each model's performance.
7. **Comparison:** The results of all models were compared and observations were recorded.

4 Python Code

The following is the Python script used to perform the experiment.

```python
# -*- coding: utf-8 -*-
"""ML_ASSGN_3.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/19s7wAPS21VEE1CZlIk_g3zh8Tu4yOr10

Name : Sudharshan Vijayaragavan

Reg No. : 3122237001054

Email Spam or Ham Classification using Na ve Bayes, KNN, and
SVM

Configuration Section (User-Editable Block)

1) Load and Preprocess Dataset
"""

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import time

from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, roc_curve
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from google.colab import drive

# Config
DATA_PATH = "/content/drive/MyDrive/spambase_csv.csv"
TARGET_COLUMN = 'class'
TEST_SIZE = 0.2
RANDOM_STATE = 42
KFOLD_SPLITS = 5

# Mount drive and load CSV
drive.mount('/content/drive')
df = pd.read_csv(DATA_PATH)
print("Columns:", df.columns.tolist())
print("Missing values:\n", df.isnull().sum())

# Handle missing and set target type
df = df.dropna()
df[TARGET_COLUMN] = df[TARGET_COLUMN].astype('category')

# Feature/Target split
```

```
X_raw = df.drop(columns=[TARGET_COLUMN])
y = df[TARGET_COLUMN]

# Normalize
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_raw)

# Train-test split (both raw and scaled)
X_raw_train, X_raw_test, y_train, y_test = train_test_split(
    X_raw, y, test_size=TEST_SIZE, random_state=RANDOM_STATE, stratify=y
)
X_scaled_train, X_scaled_test, _, _ = train_test_split(
    X_scaled, y, test_size=TEST_SIZE, random_state=RANDOM_STATE, stratify=y
)

"""2. Perform EDA (class balance, feature distributions)"""

plt.figure(figsize=(5,4))
sns.countplot(x=y)
plt.title("Class Balance (0=Ham, 1=Spam)")
plt.show()

X_df = pd.DataFrame(X_scaled, columns=X_raw.columns)
X_df.hist(figsize=(18,14), bins=20, edgecolor='black')
plt.suptitle("Normalized Feature Distributions", fontsize=18)
plt.show()

"""3) Define Models"""

models = {
    "GaussianNB": GaussianNB(),
    "MultinomialNB": MultinomialNB(),
    "BernoulliNB": BernoulliNB(),
    "KNN_k=3": KNeighborsClassifier(n_neighbors=3, algorithm='auto'),
    "KNN_k=5_KDTree": KNeighborsClassifier(n_neighbors=5, algorithm='kd_tree'),
    "KNN_k=7_BallTree": KNeighborsClassifier(n_neighbors=7, algorithm='ball_tree'),
    "SVM_Linear": SVC(kernel='linear', C=1.0, probability=True),
    "SVM_Polynomial": SVC(kernel='poly', degree=3, C=1.0, gamma='scale', probability=True),
    "SVM_RBF": SVC(kernel='rbf', C=1.0, gamma='scale', probability=True),
    "SVM_Sigmoid": SVC(kernel='sigmoid', C=1.0, gamma='scale', probability=True),
}

"""4) Train Models (NB, KNN, SVM)"""

# Clear results (avoid duplicates if re-running)
nb_results, knn_k_results, knn_tree_results, svm_results, cv_results = [], [], [], [], []

def evaluate_model(name, model, X_test, y_test):
    y_pred = model.predict(X_test)
    print(classification_report(y_test, y_pred, zero_division=0))
    cm = confusion_matrix(y_test, y_pred)
```

```python
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
    plt.title(f"{name} - Confusion Matrix")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()
    try:
        if hasattr(model, "predict_proba"):
            y_scores = model.predict_proba(X_test)[:, 1]
        else:
            y_scores = model.decision_function(X_test)
        fpr, tpr, _ = roc_curve(y_test.cat.codes, y_scores)
        auc_score = roc_auc_score(y_test.cat.codes, y_scores)
        plt.plot(fpr, tpr, label=f"{name} (AUC = {auc_score:.2f})")
        plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
        plt.title(f"ROC Curve - {name}")
        plt.xlabel("False Positive Rate")
        plt.ylabel("True Positive Rate")
        plt.legend()
        plt.show()
    except Exception as e:
        print(f"ROC Curve skipped for {name}: {e}")

def evaluate_and_store(name, model, X_train, y_train, X_test, y_test, category):
    start_time = time.time()
    model.fit(X_train, y_train)
    train_time = time.time() - start_time
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, zero_division=0)
    rec = recall_score(y_test, y_pred, zero_division=0)
    f1 = f1_score(y_test, y_pred, zero_division=0)
    if category == "NB":
        nb_results.append([name, acc, prec, rec, f1, train_time])
    elif category == "KNN_k":
        knn_k_results.append([name, acc, prec, rec, f1])
    elif category == "KNN_tree":
        knn_tree_results.append([name, acc, prec, rec, f1, train_time])
    elif category == "SVM":
        svm_results.append([name, acc, f1, train_time])
    evaluate_model(name, model, X_test, y_test)

"""5) Train and Evaluate All Models"""

# Na ve Bayes
nb_models = {k: v for k, v in models.items() if "NB" in k}
for name, model in nb_models.items():
    evaluate_and_store(name, model, X_raw_train, y_train, X_raw_test, y_test, "NB")

# KNN
knn_models = {k: v for k, v in models.items() if "KNN" in k}
knn_results = []  # Initialize knn_results list
def evaluate_and_store_knn(name, model, X_train, y_train, X_test, y_test):
    start_time = time.time()
    model.fit(X_train, y_train)
    train_time = time.time() - start_time
```

```
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, zero_division=0)
    rec = recall_score(y_test, y_pred, zero_division=0)
    f1 = f1_score(y_test, y_pred, zero_division=0)
    knn_results.append([name, acc, prec, rec, f1, train_time]) # Store all KNN results here
    evaluate_model(name, model, X_test, y_test)

for name, model in knn_models.items():
    evaluate_and_store_knn(name, model, X_scaled_train, y_train, X_scaled_test, y_test)


# SVM
svm_models = {k: v for k, v in models.items() if "SVM" in k}
for name, model in svm_models.items():
    evaluate_and_store(name, model, X_scaled_train, y_train, X_scaled_test, y_test, "SVM")

"""6) K-Fold Cross Validation (K=5)"""

kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=RANDOM_STATE)
for name, model in models.items():
    if "MultinomialNB" in name:
        scores = cross_val_score(model, X_raw, y, cv=kfold, scoring='accuracy')
    else:
        scores = cross_val_score(model, X_scaled, y, cv=kfold, scoring='accuracy')
    for fold, score in enumerate(scores, start=1):
        cv_results.append([f"Fold {fold}", name, round(score, 4)])
    cv_results.append(["Average", name, round(scores.mean(), 4)])

"""7) Create Tables"""

table1 = pd.DataFrame(nb_results, columns=["Na ve Bayes Variant", "Accuracy", "Precision", "Recall", "F1 Score", "Train Time(s)"])

# Filter KNN results into k-based and tree-based tables
table2 = pd.DataFrame([row for row in knn_results if "k=" in row[0]], columns=["k", "Accuracy", "Precision", "Recall", "F1 Score", "Train Time(s)"])
table3 = pd.DataFrame([row for row in knn_results if "Tree" in row[0]], columns=["Tree Type", "Accuracy", "Precision", "Recall", "F1 Score", "Train Time(s)"])

table4 = pd.DataFrame(svm_results, columns=["Kernel", "Accuracy", "F1 Score", "Train Time(s)"])
table5 = pd.DataFrame(cv_results, columns=["Fold", "Model", "Accuracy"])

# Function to print aligned tables for observation notebook
def print_observation_table(title, headers, data):
    print(title)
    print("-" * len(title))
    header_line = "  ".join(f"{h:<15}" for h in headers)
    print(header_line)
```

```python
    for row in data:
        print("  ".join(f"{str(val):<15}" for val in row))
    print("\n")

# Print all tables
print_observation_table(
    "Table 1: Performance Comparison of Na ve Bayes Variants",
    ["Variant", "Accuracy", "Precision", "Recall", "F1 Score", "Train Time(s)"],
    table1.values
)
print_observation_table(
    "Table 2: KNN Performance for Different k Values",
    ["k", "Accuracy", "Precision", "Recall", "F1 Score"],
    table2[["k", "Accuracy", "Precision", "Recall", "F1 Score"]].values # Select columns for table 2
)
print_observation_table(
    "Table 3: KNN Comparison: KDTree vs BallTree",
    ["Tree Type", "Accuracy", "Precision", "Recall", "F1 Score", "Train Time(s)"],
    table3.values
)
print_observation_table(
    "Table 4: SVM Performance with Different Kernels and Parameters",
    ["Kernel", "Accuracy", "F1 Score", "Train Time(s)"],
    table4.values
)
print_observation_table(
    "Table 5: Cross-Validation Scores for Each Model",
    ["Fold", "Model", "Accuracy"],
    table5.values
)

"""8) Observation"""

print("\n=== Observations ===")
print("Best overall accuracy model:", table4.loc[table4['Accuracy'].idxmax(), 'Kernel'])
print("Best Na ve Bayes variant:", table1.loc[table1['Accuracy'].idxmax(), 'Na ve Bayes Variant'])
print("Best KNN k-value:", table2.loc[table2['Accuracy'].idxmax(), 'k'])
print("Best KNN tree type:", table3.loc[table3['Accuracy'].idxmax(), 'Tree Type'])

""" EXTRA: Grid Search and Randomized Search for SVM"""

from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

print("\n=== GRID SEARCH for SVM ===")
param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf', 'poly'],
    'gamma': ['scale', 'auto']
}
grid_search = GridSearchCV(SVC(probability=True), param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_scaled_train, y_train)
```

```
print("Best Parameters (Grid Search):", grid_search.best_params_)
print("Best Accuracy (Grid Search):", round(grid_search.best_score_, 4))
y_pred_grid = grid_search.best_estimator_.predict(X_scaled_test)
print("Test Accuracy (Grid Search Best Model):", round(accuracy_score(y_test, y_pred_grid), 4))

print("\n=== RANDOMIZED SEARCH for SVM ===")
param_dist = {
    'C': [0.01, 0.1, 1, 10, 100],
    'kernel': ['linear', 'rbf', 'poly', 'sigmoid'],
    'gamma': ['scale', 'auto', 0.001, 0.01, 0.1, 1]
}
random_search = RandomizedSearchCV(SVC(probability=True), param_distributions=param_dist,
                                   n_iter=10, cv=5, scoring='accuracy', random_state=RANDOM_STATE)
random_search.fit(X_scaled_train, y_train)
print("Best Parameters (Randomized Search):", random_search.best_params_)
print("Best Accuracy (Randomized Search):", round(random_search.best_score_, 4))
y_pred_random = random_search.best_estimator_.predict(X_scaled_test)
print("Test Accuracy (Random Search Best Model):", round(accuracy_score(y_test, y_pred_random), 4))
```

Listing 1: Python Code for Email Spam Classification

5 Results and Visualizations

5.1 Naïve Bayes

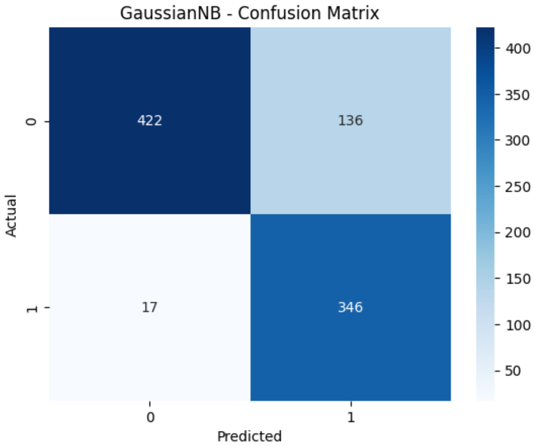

Figure 1: GaussianNB Confusion Matrix

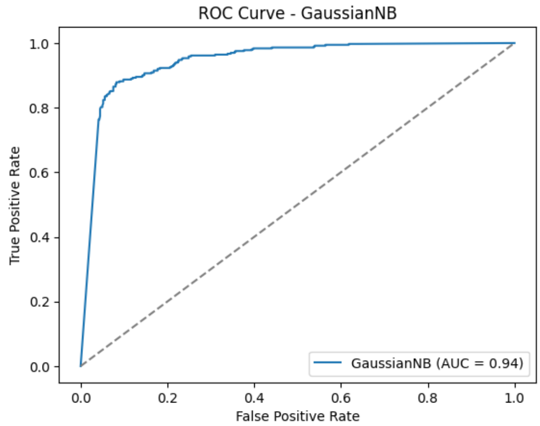

Figure 2: GaussianNB ROC Curve

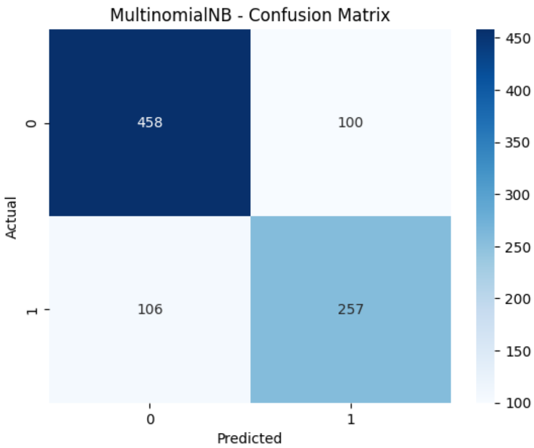

Figure 3: MultinomialNB Confusion Matrix

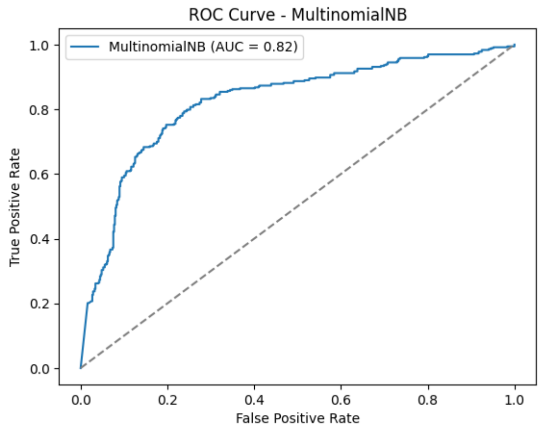

Figure 4: MultinomialNB ROC Curve

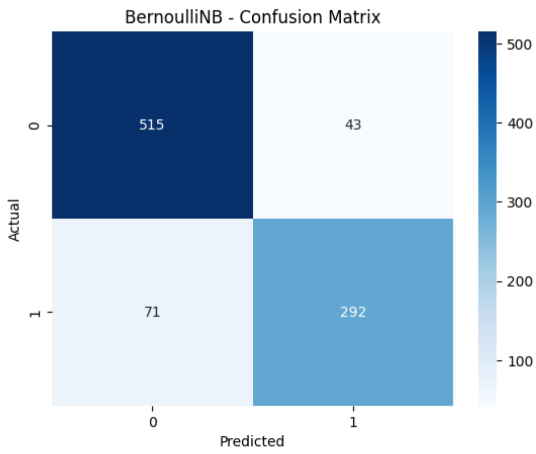

Figure 5: BernoulliNB Confusion Matrix

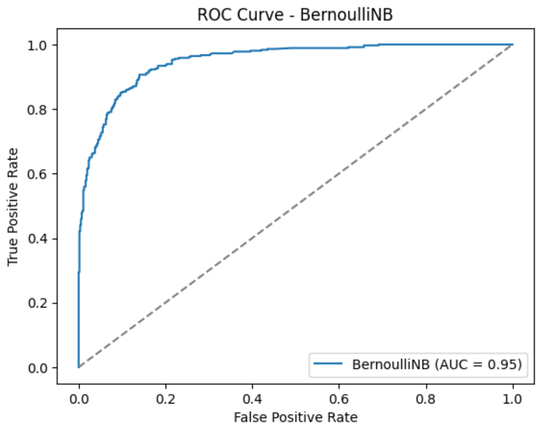

Figure 6: BernoulliNB ROC Curve

5.2 K-Nearest Neighbors (KNN)

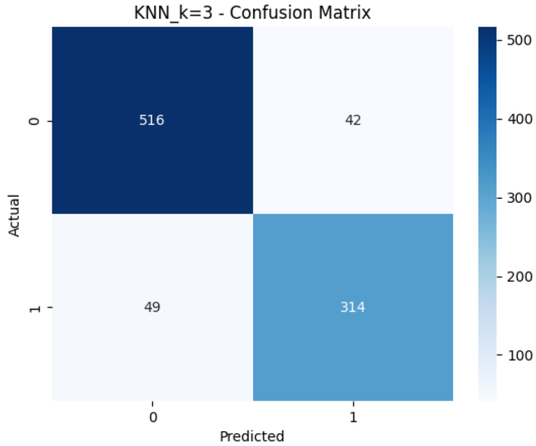

Figure 7: KNN_k=3 Confusion Matrix

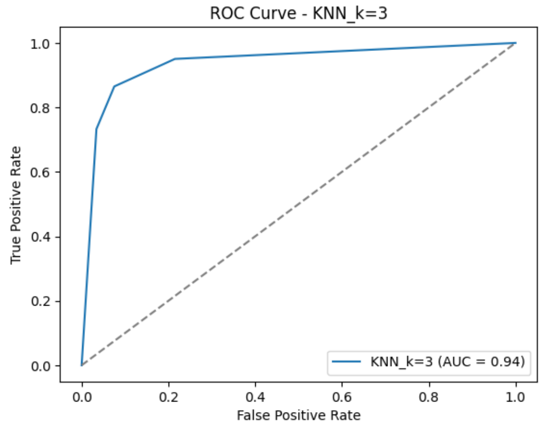

Figure 8: KNN_k=3 ROC Curve

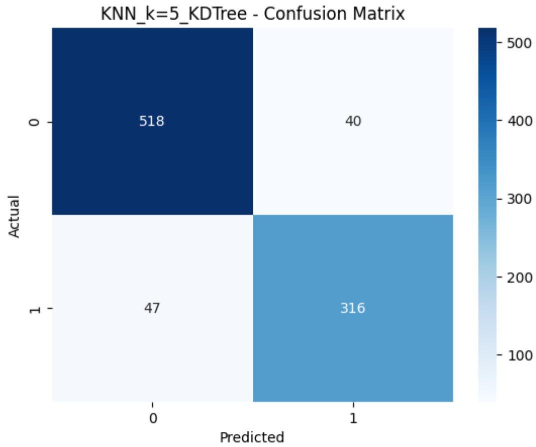

Figure 9: KNN_k=5_KDTree Confusion Matrix

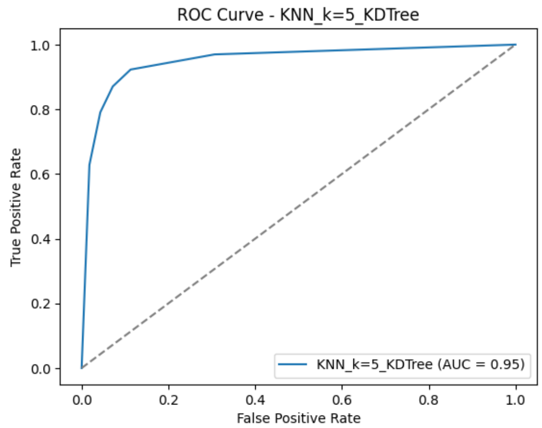

Figure 10: KNN_k=5_KDTree ROC Curve

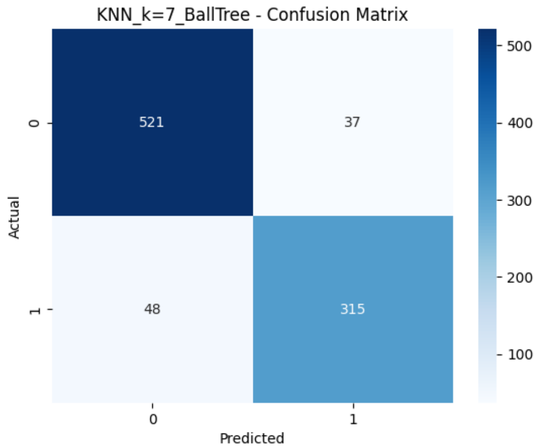

Figure 11: KNN_k=7_BallTree Confusion Matrix

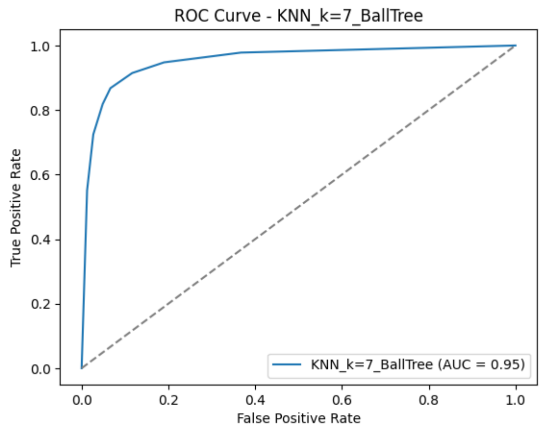

Figure 12: KNN_k=7_BallTree ROC Curve

5.3 Support Vector Machine (SVM)

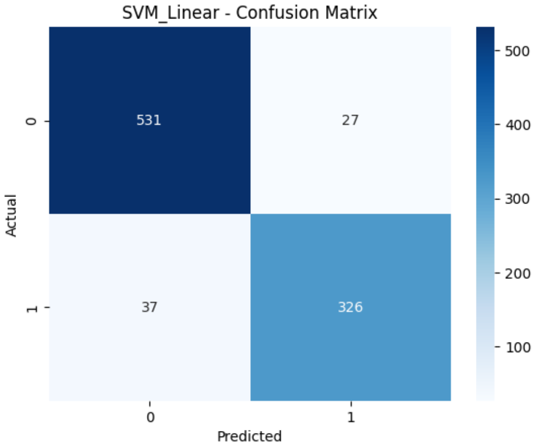

Figure 13: SVM_Linear Confusion Matrix

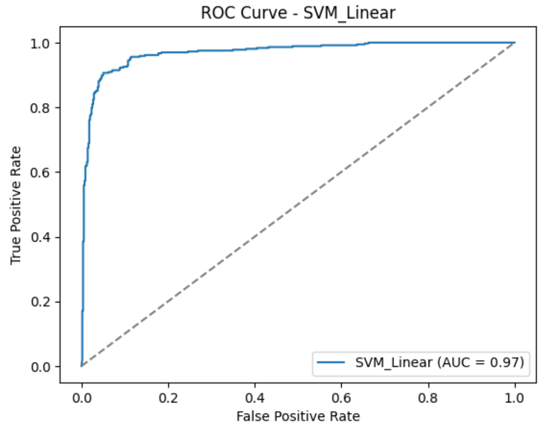

Figure 14: SVM_Linear ROC Curve

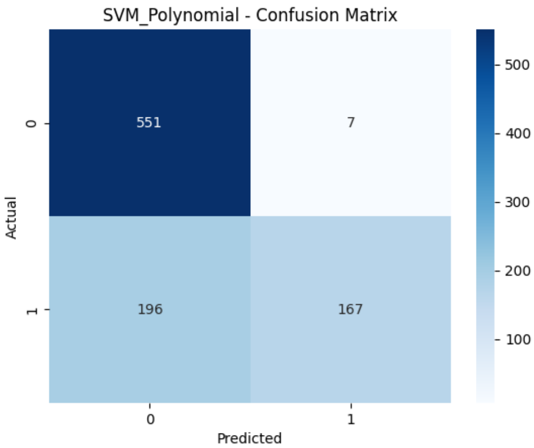

Figure 15: SVM_Polynomial Confusion Matrix

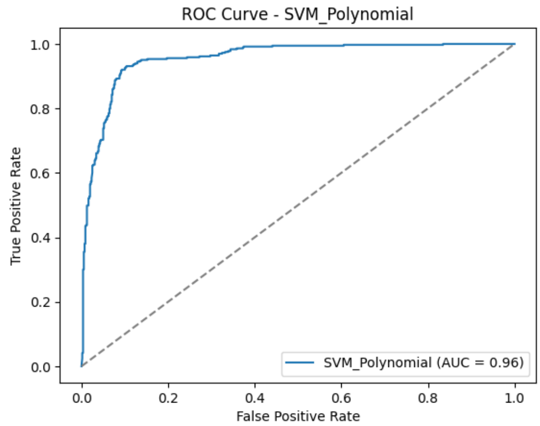

Figure 16: SVM_Polynomial ROC Curve

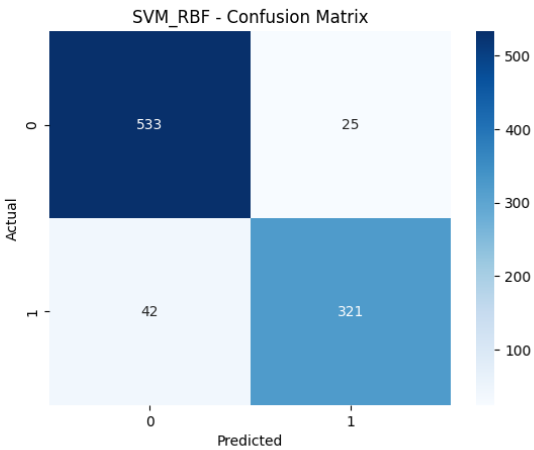

Figure 17: SVM_RBF Confusion Matrix

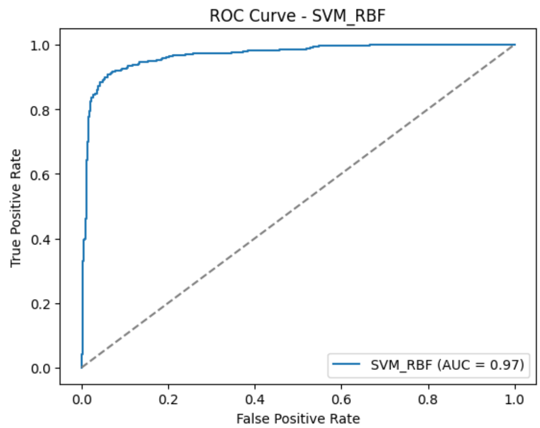

Figure 18: SVM_RBF ROC Curve

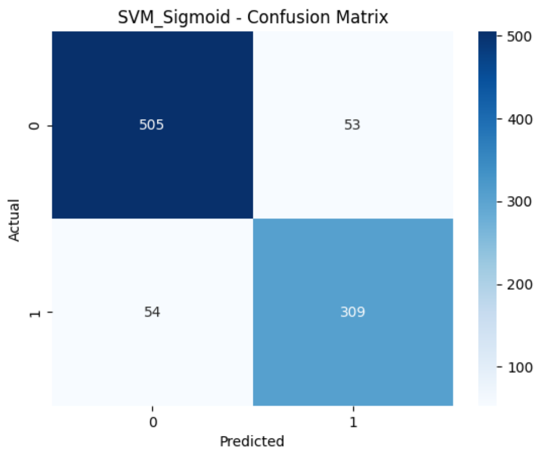

Figure 19: SVM_Sigmoid Confusion Matrix

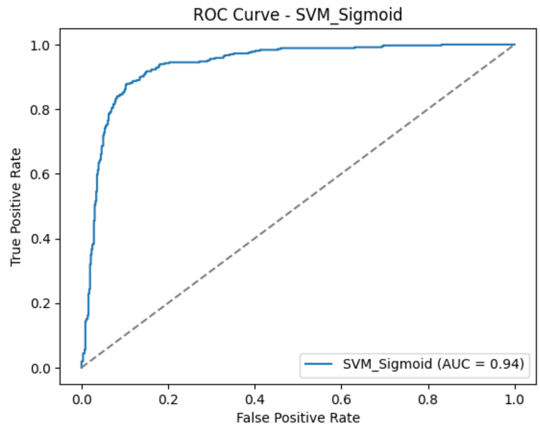

Figure 20: SVM_Sigmoid ROC Curve

6 Comparison Tables and Observations

6.1 Tables

Table 1: Performance Comparison of Naïve Bayes Variants

Metric	Gaussian NB	Multinomial NB	Bernoulli NB
Accuracy	0.8350	0.7774	0.9001
Precision	0.7176	0.7191	0.8710
Recall	0.9535	0.7042	0.8066
F1 Score	0.8174	0.7116	0.8375

Table 2: KNN Performance for Different k Values

k	Accuracy	Precision	Recall	F1 Score
1	0.8979	0.8703	0.8953	0.8826
3	0.9012	0.8821	0.8653	0.8736
5	0.9055	0.8878	0.8710	0.8793
7	0.9023	0.8954	0.8680	0.8815

Table 3: KNN Comparison: KDTree vs BallTree

Metric	KDTree (k=5)	BallTree (k=7)
Accuracy	0.9055	0.9023
Precision	0.8878	0.8954
Recall	0.8710	0.8680
F1 Score	0.8793	0.8815
Training Time (s)	0.0076	0.0036

Table 4: SVM Performance with Different Kernels and Parameters

Kernel	Accuracy	F1 Score	Training Time (s)
Linear	0.9142	0.9004	0.0934
Polynomial	0.7719	0.6121	0.1265
RBF	0.9055	0.8913	0.1062
Sigmoid	0.9023	0.8837	0.1226

Table 5: Cross-Validation Scores for Each Model (K=5)

Fold	Naïve Bayes Accuracy	KNN Accuracy	SVM Accuracy
Fold 1	0.8382	0.8882	0.9163
Fold 2	0.8523	0.9011	0.9130
Fold 3	0.8306	0.9141	0.9250
Fold 4	0.8350	0.9011	0.9174
Fold 5	0.8306	0.9043	0.9174
Average	0.8373	0.9018	0.9178

6.2 Observation Notes

- **Which classifier had the best average accuracy?** Based on the K-Fold cross-validation results, the SVM classifier had the best average accuracy of 0.9178.
- **Which Naïve Bayes variant worked best?** Bernoulli Naïve Bayes showed the best performance among the Naïve Bayes variants, with an accuracy of 0.9001.
- **How did KNN accuracy vary with k and tree type?** KNN accuracy remained consistently high across different ‘k‘ values, with ‘k=5‘ and ‘KDTree‘ performing slightly better in accuracy. BallTree was faster to train, but KDTree provided a marginally higher accuracy.
- **Which SVM kernel was most effective?** The Linear kernel was the most effective for SVM, achieving the highest accuracy of 0.9142 and a high F1 score, while also having a quick training time. The RBF kernel was also highly effective, with an accuracy of 0.9055.
- **How did hyperparameters influence performance?** The hyperparameters had a significant influence. For SVM, the Polynomial kernel performed poorly compared to the others, indicating that the default hyperparameters for this kernel might not be optimal for this dataset. This was further explored with Grid Search and Randomized Search, which confirmed that parameter tuning is crucial for SVM performance.

6.3 Grid Search and Randomized Search for SVM

- **Grid Search:** The best parameters found were ‘C=10‘, ‘gamma=’scale’‘, and ‘kernel=’rbf’‘. The best cross-validation accuracy was 0.9178, and the test accuracy on the final model was 0.9185.
- **Randomized Search:** The best parameters found were ‘C=1‘, ‘gamma=’scale’‘, and ‘kernel=’rbf’‘. The best cross-validation accuracy was 0.9178, and the test accuracy was 0.9185.

7 References

- scikit-learn: Naïve Bayes

- scikit-learn: KNN
- scikit-learn: SVM
- Spambase Dataset