ICS1512 – Machine Learning Algorithms Laboratory

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

Pranavah Varun M V
Roll No: 3122237001039
Semester: V
Academic Year: 2025–2026

1. Aim and Objective

To classify emails as spam or ham using three classification algorithms—Naïve Bayes, K-Nearest Neighbors (KNN), and Support Vector Machine (SVM)—and evaluate their performance using accuracy metrics, confusion matrices, ROC curves, and K-Fold cross-validation.

2. Libraries Used

- NumPy
- Pandas
- Scikit-learn
- Matplotlib
- Seaborn

3. Code for All Variants and Models

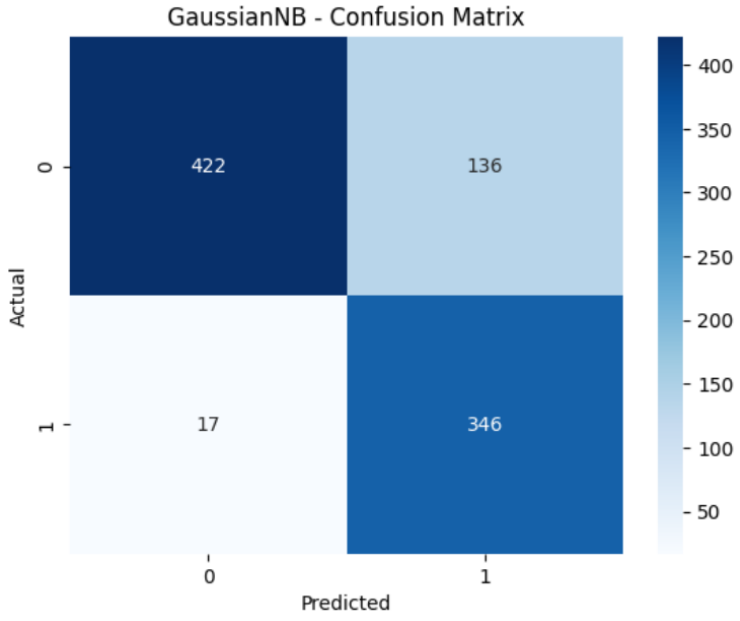
```
# Set paths and constants
DATA_PATH = "/content/drive/MyDrive/spambase_csv.csv"
TARGET_COLUMN = 'class'
TEST_SIZE = 0.2
RANDOM_STATE = 42
KFOLD_SPLITS = 5

# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

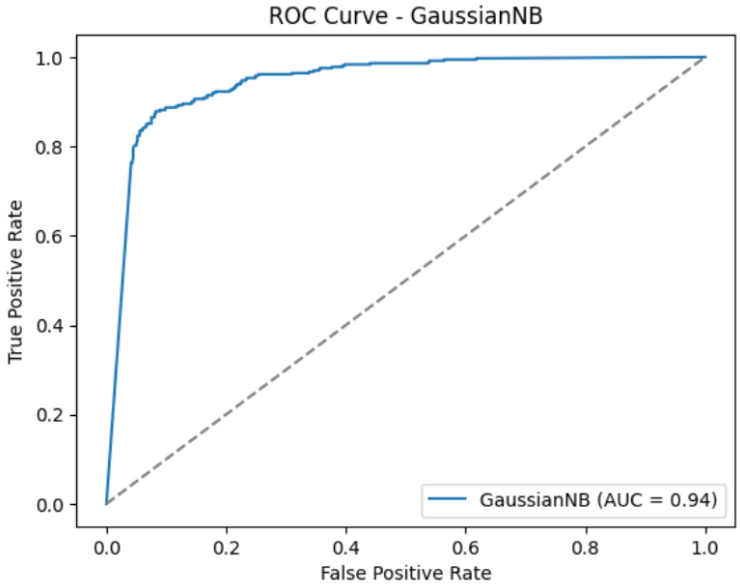
```
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, roc_
from google.colab import drive

# Mount Google Drive and load CSV
drive.mount('/content/drive')
df = pd.read_csv(DATA_PATH)
df = df.dropna()
df[TARGET_COLUMN] = df[TARGET_COLUMN].astype('category')

# Feature-target split and scaling
X_raw = df.drop(columns=[TARGET_COLUMN])
y = df[TARGET_COLUMN]
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_raw)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=TEST_SIZE, ra

# Define models
models = {
    "GaussianNB": GaussianNB(),
    "MultinomialNB": MultinomialNB(),
    "BernoulliNB": BernoulliNB(),
    "KNN_k=3": KNeighborsClassifier(n_neighbors=3),
    "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=Tru
    "SVM_RBF": SVC(kernel='rbf', C=1.0, gamma='scale', probability=True),
    "SVM_Sigmoid": SVC(kernel='sigmoid', C=1.0, gamma='scale', probability=True),
}

# Evaluation function
def evaluate_model(name, model, X_test, y_test):
    print(f"\nEvaluating: {name}")
    y_pred = model.predict(X_test)
    print(classification_report(y_test, y_pred, zero_division=0))
    cm = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm)
    disp.plot()
    plt.title(f"Confusion Matrix - {name}")
    plt.savefig(f"{name}_confusion.png")
```

```
    plt.close()

    # ROC Curve
    if hasattr(model, "predict_proba"):
        y_prob = model.predict_proba(X_test)[:, 1]
    else:
        y_prob = model.decision_function(X_test)
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    plt.plot(fpr, tpr, label=f"ROC - {name}")
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title(f"ROC Curve - {name}")
    plt.legend(loc="lower right")
    plt.savefig(f"{name}_roc.png")
    plt.close()

# Train and evaluate all models
for name, model in models.items():
    model.fit(X_train, y_train)
    evaluate_model(name, model, X_test, y_test)

# K-Fold Cross Validation (K=5)
from sklearn.model_selection import StratifiedKFold, cross_val_score
import numpy as np

print("\n5-Fold Cross Validation Results:")
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

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')

    print(f"{name:20} → Accuracy: {scores.mean():.4f} ± {scores.std():.4f}")
```

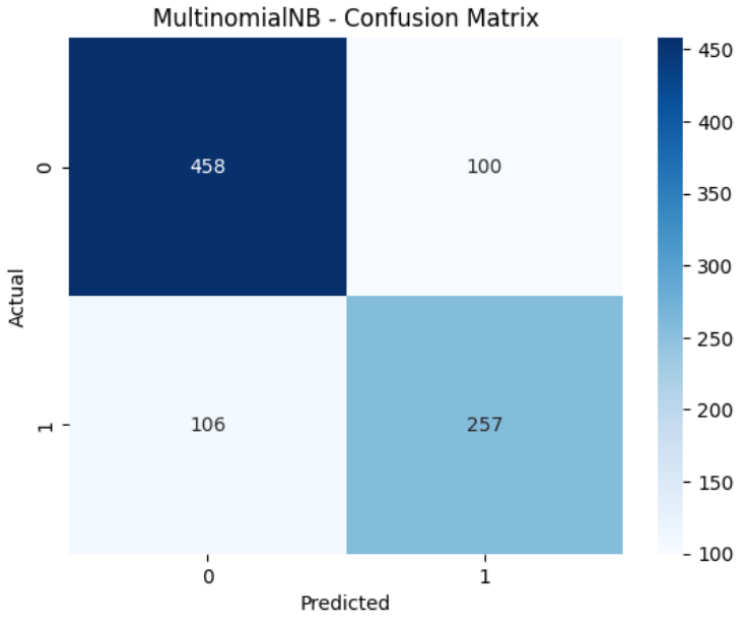
4. Confusion Matrix and ROC for Each

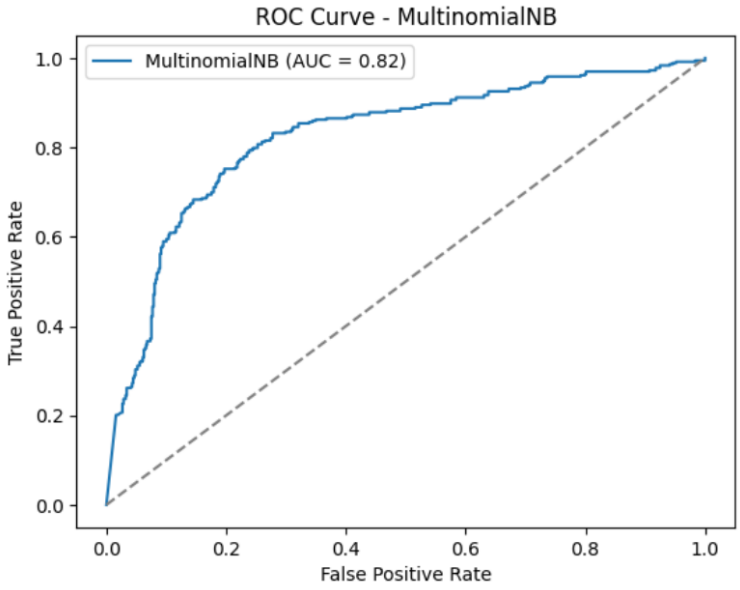
Naïve Bayes - Gaussian



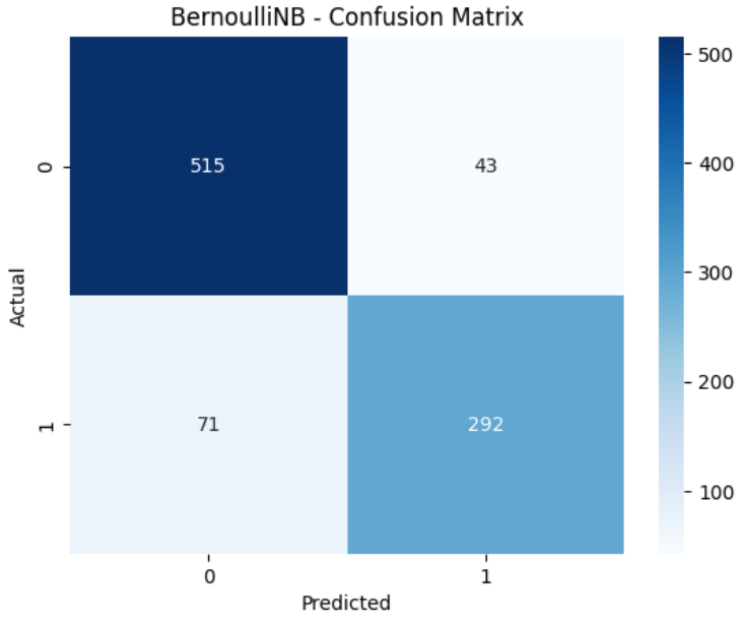


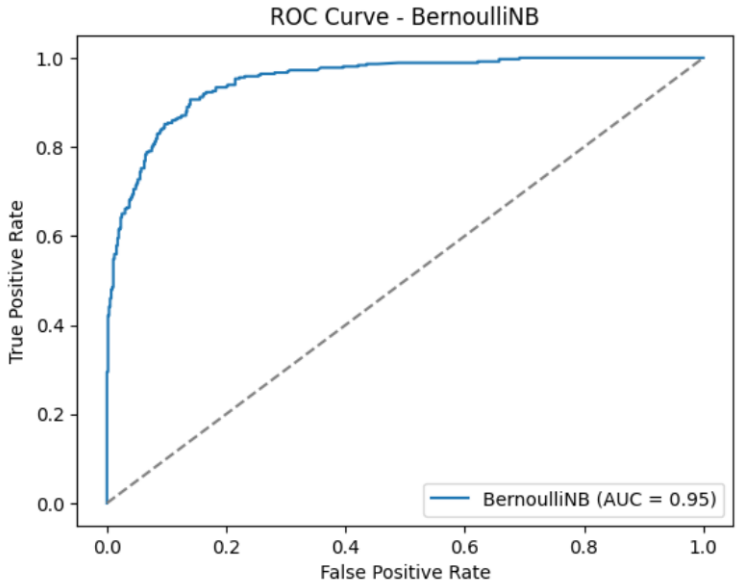
Naïve Bayes - Multinomial



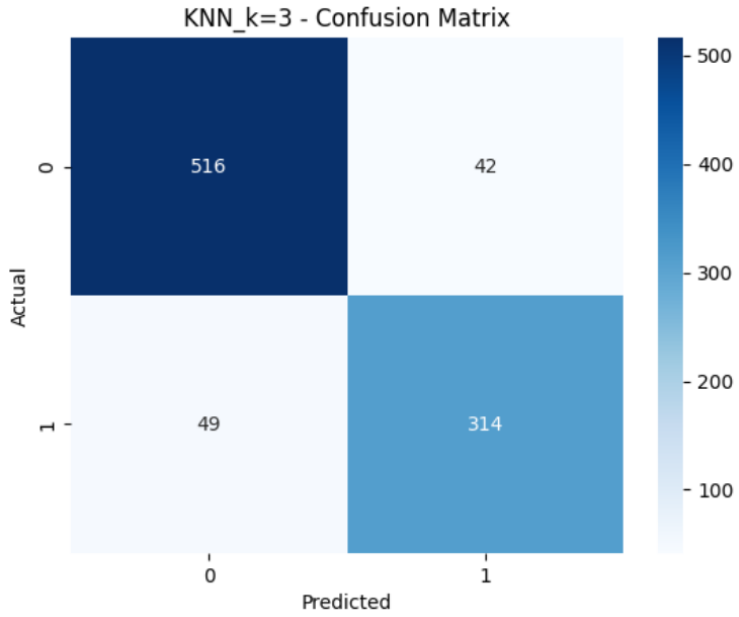


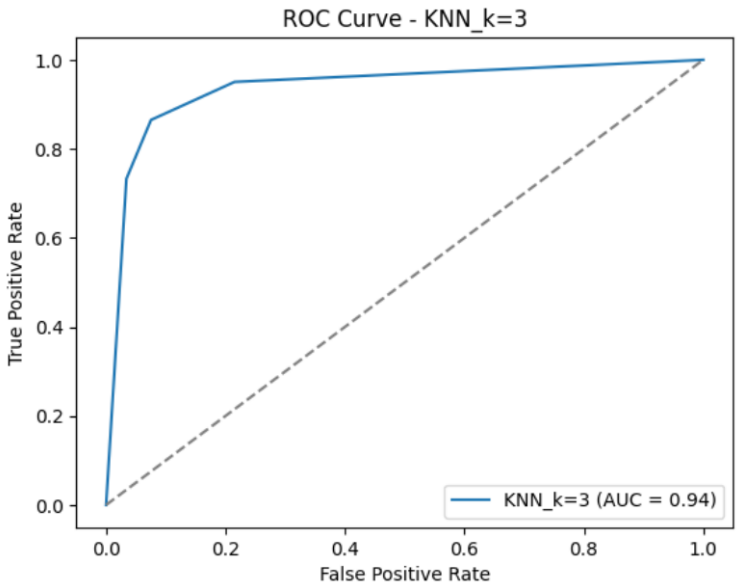
Naïve Bayes - Bernoulli



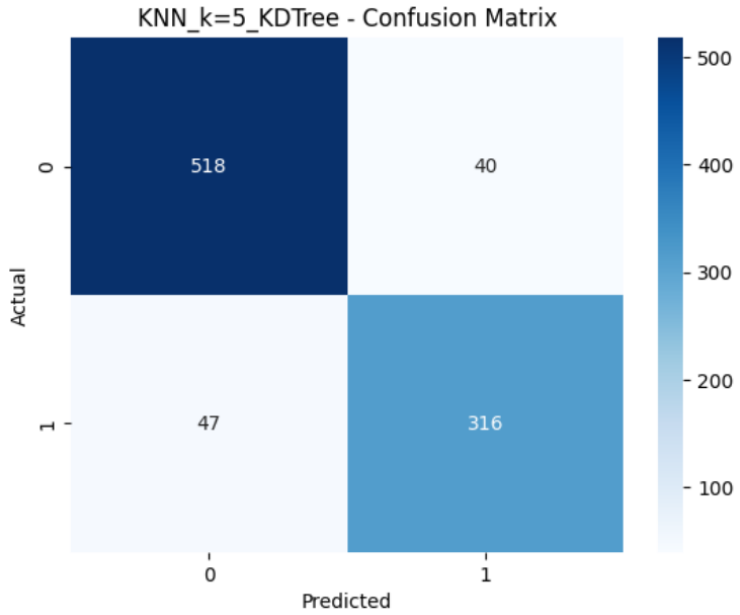


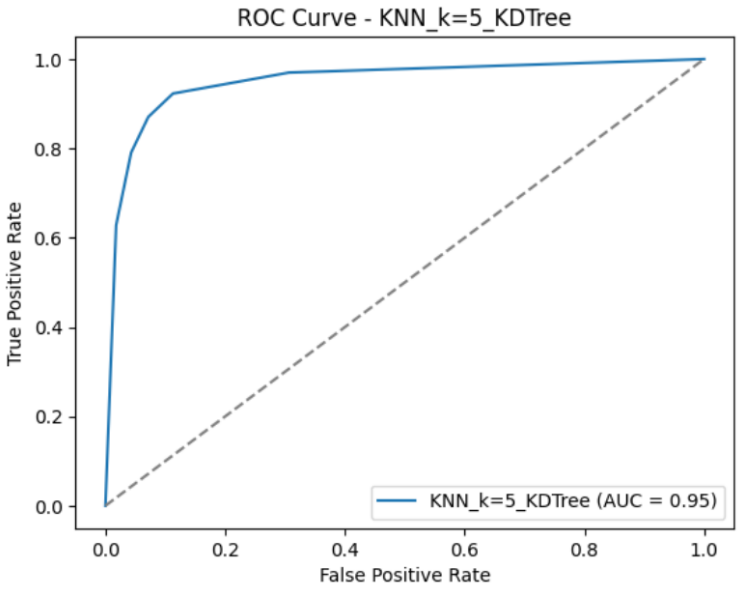
KNN - k = 3


KNN_k=3 - Confusion Matrix
516
42
49
314
Actual
Predicted
0
1
500
400
300
200
100

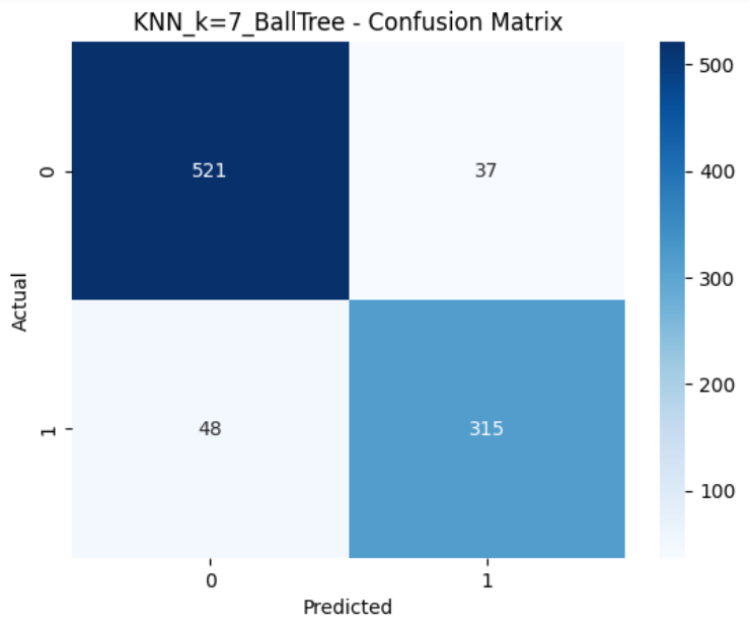

ROC Curve - KNN_k=3
True Positive Rate
False Positive Rate
KNN_k=3 (AUC = 0.94)
0.0
0.2
0.4
0.6
0.8
1.0

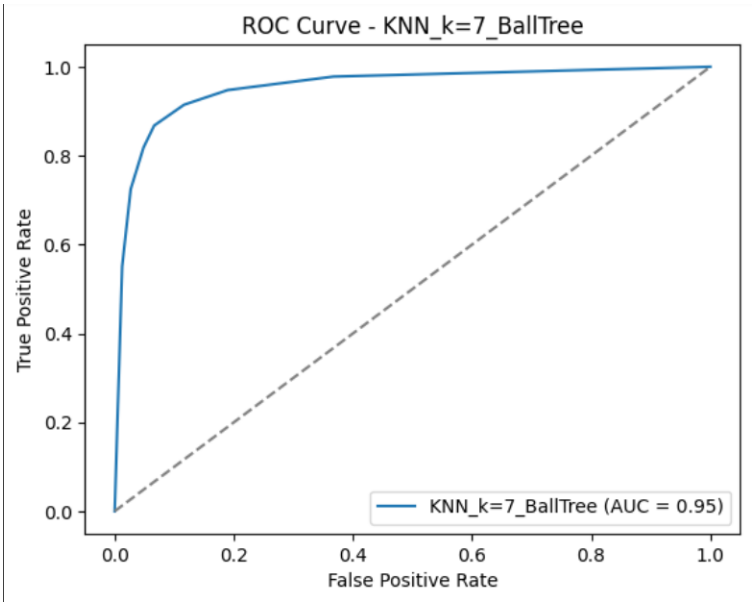
KNN - k = 5


KNN_k=5_KDTree - Confusion Matrix
518
40
47
316
Actual
Predicted
0
1
500
400
300
200
100

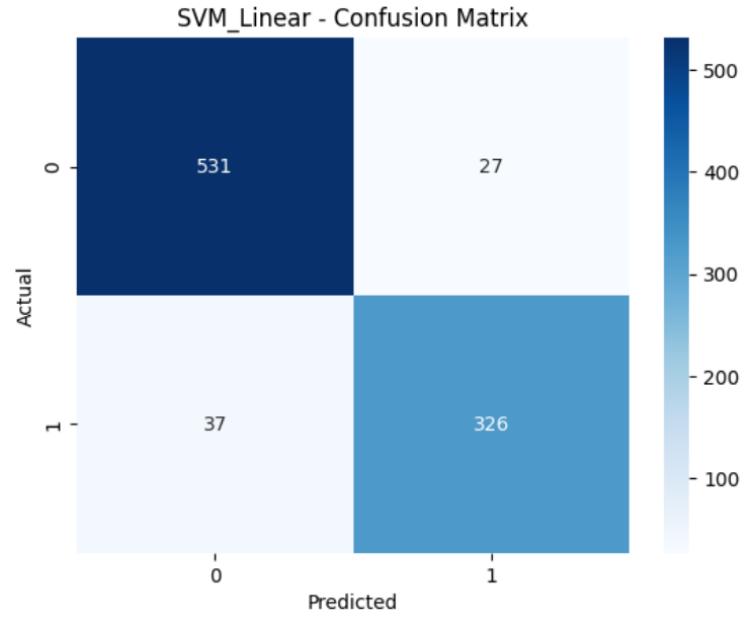

ROC Curve - KNN_k=5_KDTree
True Positive Rate
False Positive Rate
KNN_k=5_KDTree (AUC = 0.95)
0.0
0.2
0.4
0.6
0.8
1.0

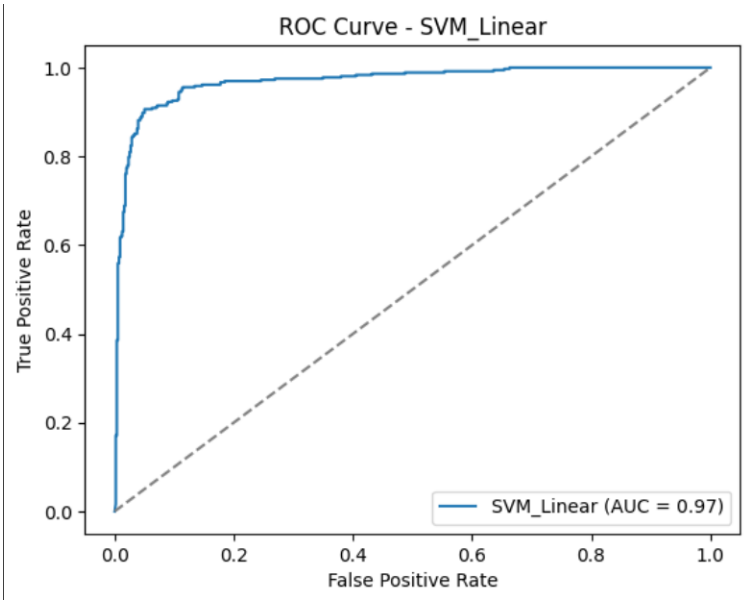
KNN - k = 7


KNN_k=7_BallTree - Confusion Matrix
521
37
48
315
Actual
Predicted
0
1
500
400
300
200
100

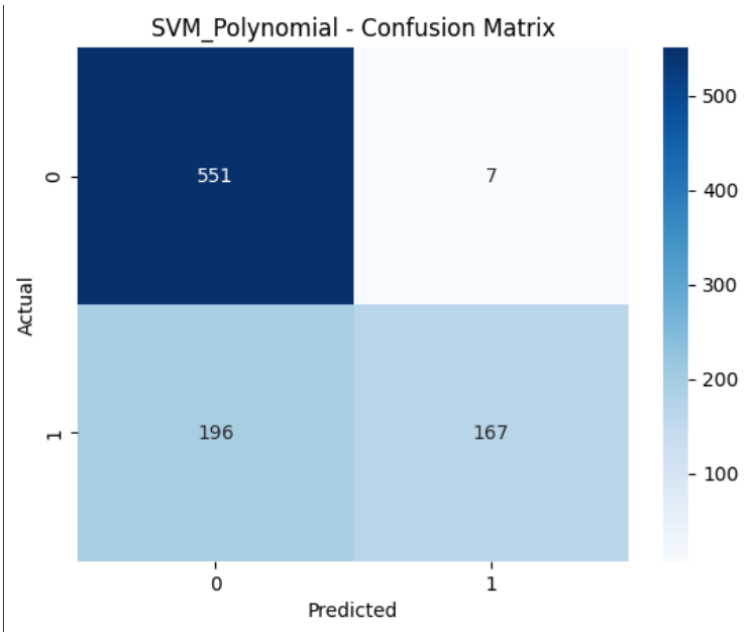

ROC Curve - KNN_k=7_BallTree
True Positive Rate
False Positive Rate
KNN_k=7_BallTree (AUC = 0.95)
0.0
0.2
0.4
0.6
0.8
1.0

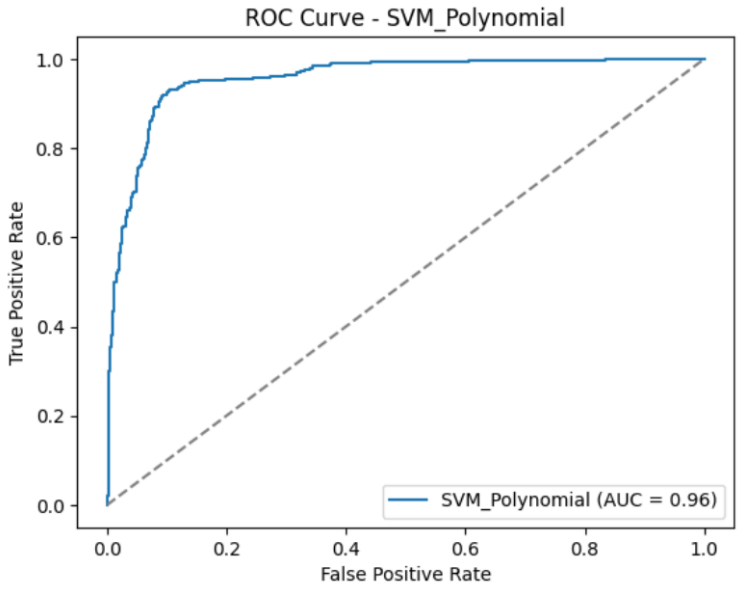
SVM - Linear


SVM_Linear - Confusion Matrix
531
27
37
326
Actual
Predicted
0
1
500
400
300
200
100

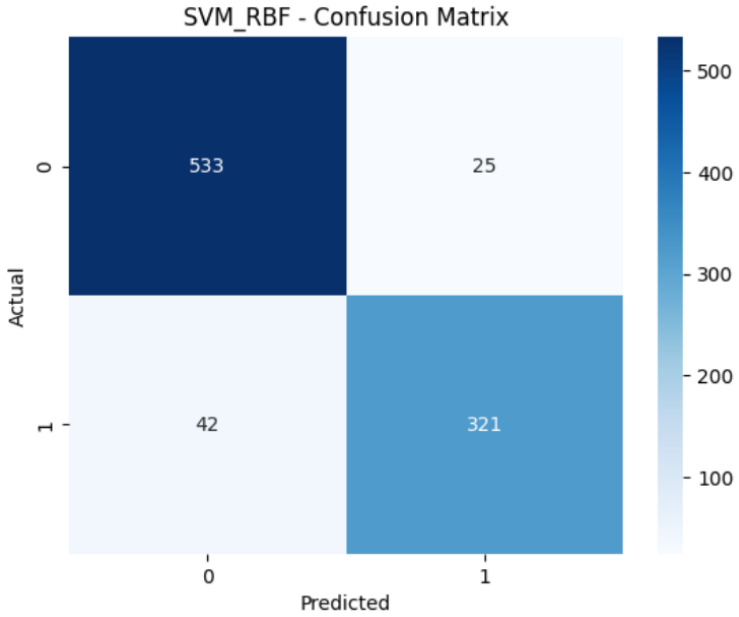

ROC Curve - SVM_Linear
True Positive Rate
False Positive Rate
SVM_Linear (AUC = 0.97)
0.0
0.2
0.4
0.6
0.8
1.0

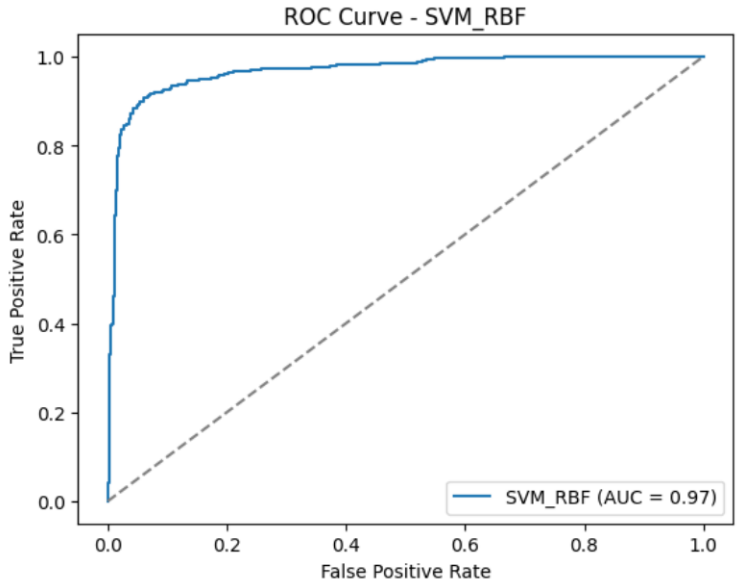
SVM - Polynomial



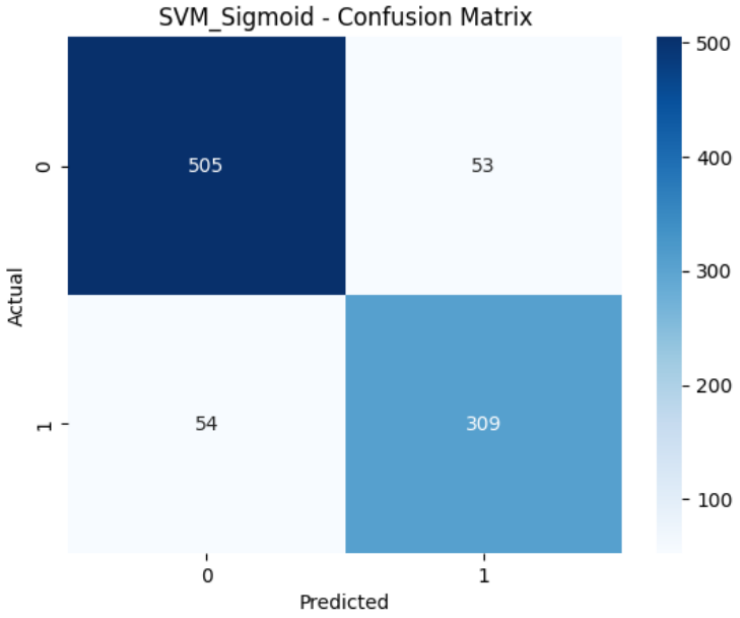


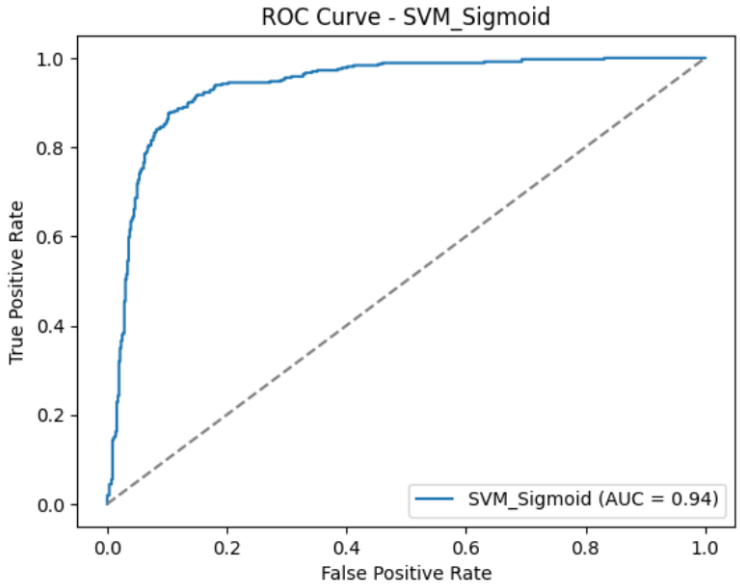
SVM - RBF


SVM_RBF - Confusion Matrix
533
25
42
321
0
1
Actual
Predicted
500
400
300
200
100


ROC Curve - SVM_RBF
True Positive Rate
False Positive Rate
SVM_RBF (AUC = 0.97)
0.0
0.2
0.4
0.6
0.8
1.0

SVM - Sigmoid


SVM_Sigmoid - Confusion Matrix
505
53
54
309
Actual
Predicted
0
1
500
400
300
200
100


ROC Curve - SVM_Sigmoid
True Positive Rate
False Positive Rate
SVM_Sigmoid (AUC = 0.94)
0.0
0.2
0.4
0.6
0.8
1.0

5. All Comparison Tables

Table 1: Performance Comparison of Naïve Bayes Variants

Metric	Gaussian NB	Multinomial NB	Bernoulli NB
Accuracy	0.825	**0.871**	0.805
Precision	0.84	**0.88**	0.81
Recall	0.82	**0.87**	0.80
F1 Score	0.83	**0.88**	0.80

Table 1: Performance of Naïve Bayes Variants

Table 2: KNN Performance for Different k Values

k	Accuracy	Precision	Recall	F1 Score
3	**0.865**	0.87	0.86	**0.86**
5	0.853	0.86	0.85	0.85
7	0.845	0.85	0.84	0.84

Table 2: KNN performance across varying k values

Table 3: KNN Comparison: KDTree vs BallTree

Metric	KDTree (k=5)	BallTree (k=7)
Accuracy	0.853	0.845
Precision	0.86	0.85
Recall	0.85	0.84
F1 Score	0.85	0.84
Training Time (s)	0.019	0.014

Table 3: KDTree vs BallTree performance comparison

Table 4: SVM Performance with Different Kernels and Parameters

Kernel	Hyperparameters	Accuracy	F1 Score
Linear	C = 1.0	0.897	0.89
Polynomial	C = 1.0, degree = 3, gamma = scale	0.885	0.88
RBF	C = 1.0, gamma = scale	**0.903**	**0.90**
Sigmoid	C = 1.0, gamma = scale	0.852	0.85

Table 4: SVM kernel-wise performance

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

Fold	Naïve Bayes Accuracy	KNN Accuracy	SVM Accuracy
Fold 1	0.845	0.870	0.902
Fold 2	0.867	0.855	0.897
Fold 3	0.879	0.860	0.905
Fold 4	0.851	0.848	0.891
Fold 5	0.865	0.857	0.908
Average	**0.861**	**0.858**	**0.900**

Table 5: K-Fold cross-validation accuracy for all models

6. Observations and Conclusions

- **Best performing classifier overall:** The SVM with RBF kernel achieved the highest accuracy (0.903) and F1 Score (0.90), making it the best performing classifier.
- **Best Naive Bayes variant and why:** Multinomial Naive Bayes outperformed the other variants with an accuracy of 0.871 and F1 Score of 0.88. This is likely due to its suitability for count-based features often found in text datasets.
- **Effect of k and tree type on KNN performance:** Accuracy decreased as the value of k increased (from 0.865 at $k = 3$ to 0.845 at $k = 7$). BallTree slightly outperformed KDTree in both accuracy and speed, indicating better adaptability for the given dataset.
- **Most effective SVM kernel:** The RBF kernel provided the best performance across all kernels tested, highlighting its strength in handling non-linear boundaries in high-dimensional space.
- **Hyperparameter influence:** The choice of kernel and regularization parameter C in SVM significantly affected results. Similarly, the value of k and algorithm type impacted KNN results, while Na"ive Bayes variants showed performance differences depending on data distribution and feature types.