

Sri Sivasubramaniya Nadar College of Engineering, Chennai
(An autonomous Institution affiliated to Anna University)

Degree & Branch	B.E. Computer Science & Engineering	Semester	V
Subject Code & Name	ICS1512 & Machine Learning Algorithms Laboratory		
Academic year	2025-2026 (Odd)	Batch:2023-2028	Due date:

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

Aim: To classify emails as **spam** or **ham (not spam)** using machine learning algorithms such as Naïve Bayes, K-Nearest Neighbors (KNN), and Support Vector Machine (SVM), and to evaluate their performance using standard classification metrics and cross-validation.

Objective

- To preprocess and analyze the email dataset obtained from Kaggle's *Spambase* dataset.
- To implement and train classification models using:
 1. Naïve Bayes (Gaussian, Multinomial, Bernoulli)
 2. K-Nearest Neighbors (with different values of k and distance metrics)
 3. Support Vector Machine (SVM) with linear, polynomial, RBF, and sigmoid kernels
- To evaluate the models using:
 - Accuracy, Precision, Recall, and F1-score
 - Confusion Matrix and ROC Curve
 - K-Fold Cross Validation ($K = 5$)
- To compare the performance of the models and draw observations.

Libraries Used

- **NumPy** – For numerical computations and array manipulations.
- **Pandas** – For dataset handling and preprocessing.
- **Matplotlib & Seaborn** – For visualizations (EDA, class balance, confusion matrix, ROC curves).

- **scikit-learn** – For implementing Naïve Bayes, KNN, SVM, evaluation metrics, and cross-validation.

Theoretical Description

- **Naïve Bayes:** A probabilistic classifier based on Bayes' theorem, assuming independence between features. It is simple, efficient, and works well for high-dimensional data like text classification. Common variants include GaussianNB, MultinomialNB, and BernoulliNB.
- **K-Nearest Neighbors (KNN):** A non-parametric algorithm that classifies a data point based on the majority class of its k nearest neighbors in the feature space. Distance metrics like Euclidean and Manhattan are used. The choice of k greatly affects accuracy and robustness.
- **Support Vector Machine (SVM):** A powerful supervised learning model that finds the optimal hyperplane separating spam and ham emails. Different kernels (Linear, Polynomial, RBF, Sigmoid) allow it to handle both linearly separable and non-linear data.
- **Evaluation Metrics:** Accuracy gives the overall correctness, Precision measures correctness of positive predictions, Recall measures how many actual positives were captured, and F1-score balances precision and recall. ROC curves and AUC values are used to evaluate the trade-off between sensitivity and specificity.
- **Cross-Validation:** To avoid overfitting and ensure generalization, 5-Fold Cross Validation is applied. This ensures models are evaluated fairly across different subsets of the data.

CODE:

```
# ===== 1. Imports =====
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from google.colab import drive
drive.mount('/content/drive')

from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score, confusion_matrix,
    roc_curve, auc, fbeta_score, matthews_corrcoef
)
```

```
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.feature_extraction.text import TfidfVectorizer
from scipy.stats import zscore

# ===== 2. Load Dataset =====
df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/ML LAB SEM 5/spam_or_not_spam.csv")
df.dropna(subset=['email'], inplace=True)

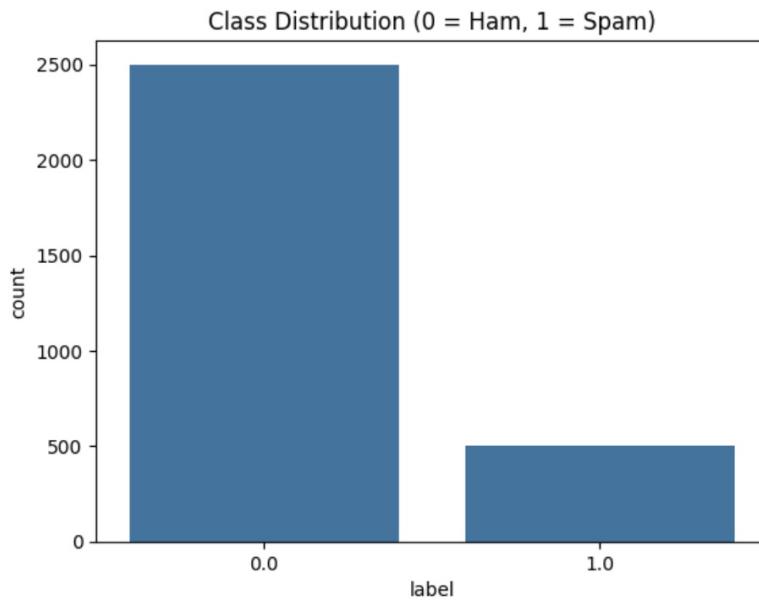
# ===== 3. EDA =====
# Fill missing numeric values if any
num_cols = df.select_dtypes(include=[np.number]).columns
if len(num_cols) > 0:
    imputer = SimpleImputer(strategy='mean')
    df[num_cols] = imputer.fit_transform(df[num_cols])

# Remove outliers from numeric columns
if len(num_cols) > 0:
    z_scores = np.abs(zscore(df[num_cols]))
    df = df[(z_scores < 3).all(axis=1)]

print(df.info())
print(df['label'].value_counts())
sns.countplot(x='label', data=df)
plt.title("Class Distribution (0 = Ham, 1 = Spam)")
plt.show()
```

OUTPUT

```
<class 'pandas.core.frame.DataFrame'>
Index: 2999 entries, 0 to 2999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  -- 
 0   email    2999 non-null   object 
 1   label    2999 non-null   float64 
dtypes: float64(1), object(1)
memory usage: 70.3+ KB
None
label
0.0    2500
1.0    499
Name: count, dtype: int64
```



```
# ===== 4. Text Preprocessing =====
tfidf = TfidfVectorizer(stop_words='english', max_features=1000)
X = tfidf.fit_transform(df['email']).toarray()
y = df['label']

# ===== 5. Scaling =====
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

print("Shape of TF-IDF matrix before scaling:", X.shape)
print("Shape of TF-IDF matrix after scaling:", X_scaled.shape)

# Print first 5 rows before and after scaling
print("\nBefore Scaling (TF-IDF values):")
print(pd.DataFrame(X[:5, :10]))

print("\nAfter Scaling:")
print(pd.DataFrame(X_scaled[:5, :10]))
```

OUTPUT

Shape of TF-IDF matrix before scaling: (2999, 1000)
Shape of TF-IDF matrix after scaling: (2999, 1000)

Before Scaling (TF-IDF values):

	0	1	2	3	4	5	6	7	8	9
0	0.031672	0.0	0.050609	0.0	0.0	0.0	0.0	0.0	0.0	0.0

```
1 0.000000 0.0 0.000000 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
2 0.000000 0.0 0.000000 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
3 0.098441 0.0 0.000000 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
4 0.000000 0.0 0.000000 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
```

After Scaling:

```
          0         1         2         3         4         5         6   \
0  0.435582 -0.13322  1.980391 -0.126859 -0.126388 -0.179581 -0.112595
1 -0.454271 -0.13322 -0.180178 -0.126859 -0.126388 -0.179581 -0.112595
2 -0.454271 -0.13322 -0.180178 -0.126859 -0.126388 -0.179581 -0.112595
3  2.311546 -0.13322 -0.180178 -0.126859 -0.126388 -0.179581 -0.112595
4 -0.454271 -0.13322 -0.180178 -0.126859 -0.126388 -0.179581 -0.112595

          7         8         9
0 -0.120289 -0.137888 -0.125068
1 -0.120289 -0.137888 -0.125068
2 -0.120289 -0.137888 -0.125068
3 -0.120289 -0.137888 -0.125068
4 -0.120289 -0.137888 -0.125068
```

```
# ===== 6. Split Data Separately =====
# For models that require scaled features (KNN, SVM)
X_train_scaled, X_temp_scaled, y_train_scaled, y_temp_scaled = train_test_split(
    X_scaled, y, test_size=0.3, random_state=42, stratify=y
)
X_val_scaled, X_test_scaled, y_val_scaled, y_test_scaled = train_test_split(
    X_temp_scaled, y_temp_scaled, test_size=0.5, random_state=42, stratify=y_temp_scaled
)

# For models that require unscaled features (Naive Bayes)
X_train_raw, X_temp_raw, y_train_raw, y_temp_raw = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)
X_val_raw, X_test_raw, y_val_raw, y_test_raw = train_test_split(
    X_temp_raw, y_temp_raw, test_size=0.5, random_state=42, stratify=y_temp_raw
)

# ===== 7. Evaluation Helper =====
def evaluate_model(model, name, X_test, y_test):
    y_pred = model.predict(X_test)
    print(f"\n{name} Evaluation:")
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("Precision:", precision_score(y_test, y_pred))
    print("Recall:", recall_score(y_test, y_pred))
    print("F1 Score:", f1_score(y_test, y_pred))
```

```
print("F-beta Score ( $\beta=0.5$ ):", fbeta_score(y_test, y_pred, beta=0.5))
print("Matthews Corr Coef:", matthews_corrcoef(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title(f"{name} - Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

if hasattr(model, "predict_proba"):
    y_score = model.predict_proba(X_test)[:, 1]
else:
    y_score = model.decision_function(X_test)

fpr, tpr, _ = roc_curve(y_test, y_score)
plt.plot(fpr, tpr, label=f"AUC = {auc(fpr, tpr):.2f}")
plt.plot([0, 1], [0, 1], linestyle="--")
plt.title(f"{name} - ROC Curve")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.legend()
plt.grid()
plt.show()

# ===== 8. Naïve Bayes =====
for name, model in {
    "GaussianNB": GaussianNB(),
    "MultinomialNB": MultinomialNB(),
    "BernoulliNB": BernoulliNB()
}.items():
    model.fit(X_train_raw, y_train_raw)
    evaluate_model(model, name, X_test_raw, y_test_raw)
```

OUTPUT

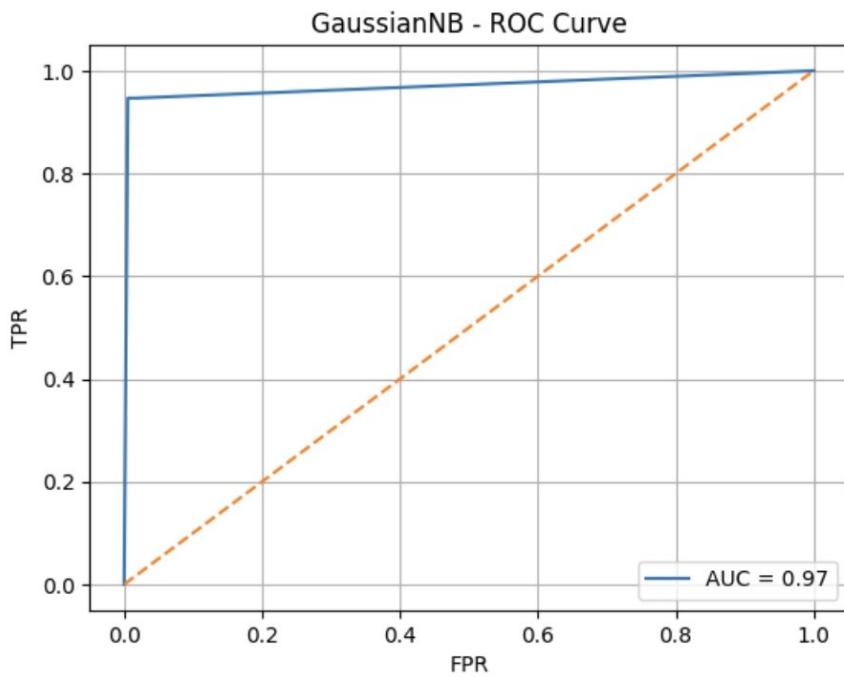
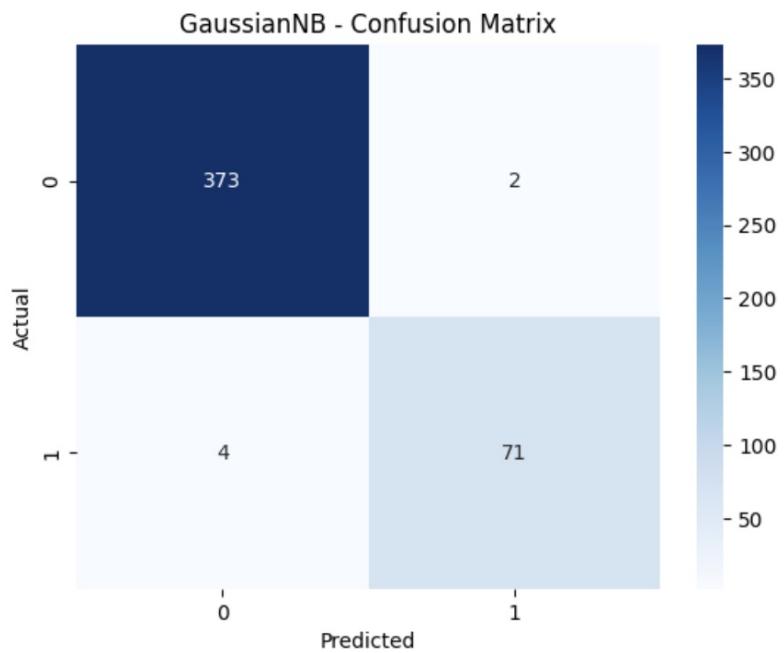
GaussianNB Evaluation:
Accuracy: 0.9866666666666667
Precision: 0.9726027397260274
Recall: 0.9466666666666667
F1 Score: 0.9594594594594594
F-beta Score ($\beta=0.5$): 0.9673024523160763
Matthews Corr Coef: 0.9516069343072303

Date: 08-08-2025

Experiment: 3

Name: Harini LV

Roll No: 3122237001016



MultinomialNB Evaluation:

Accuracy: 0.9866666666666667

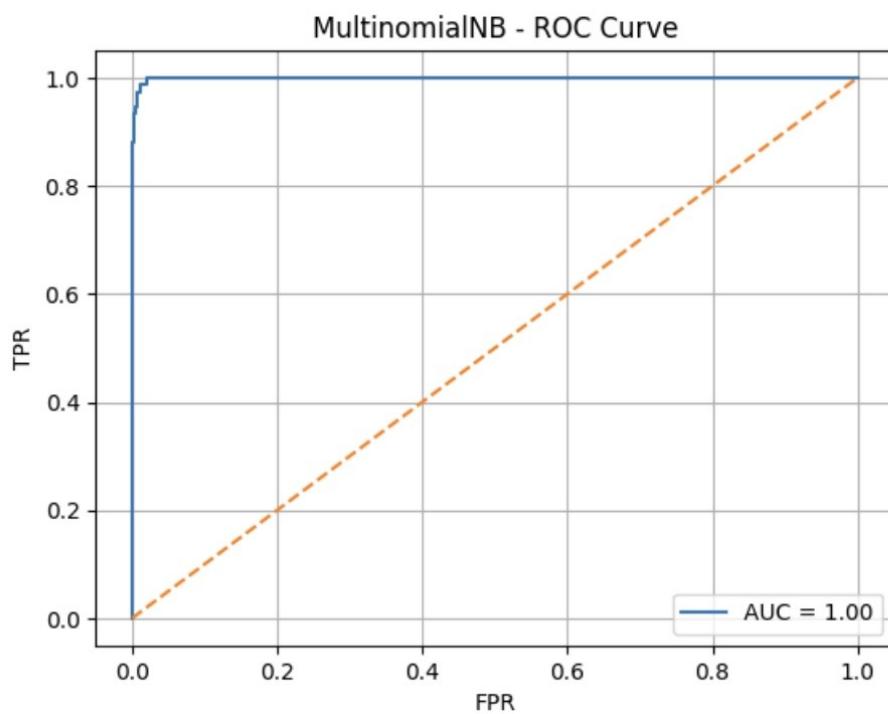
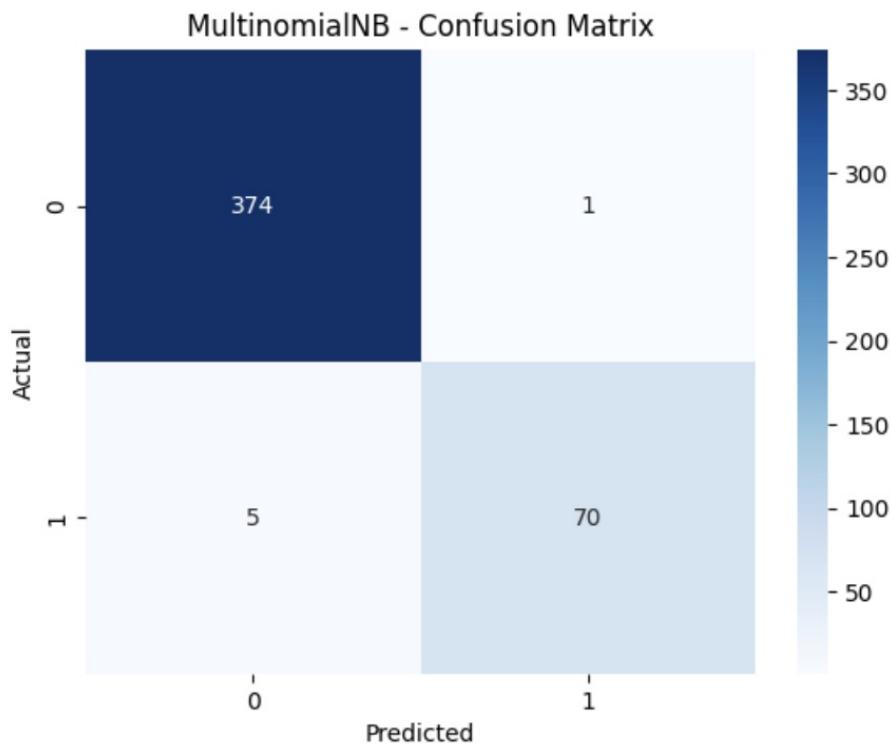
Precision: 0.9859154929577465

Recall: 0.9333333333333333

F1 Score: 0.958904109589041

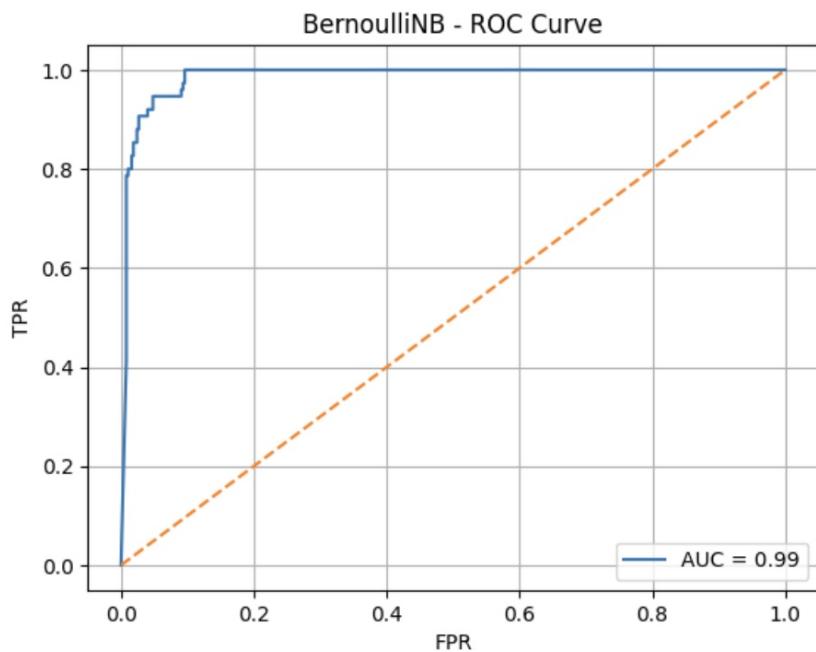
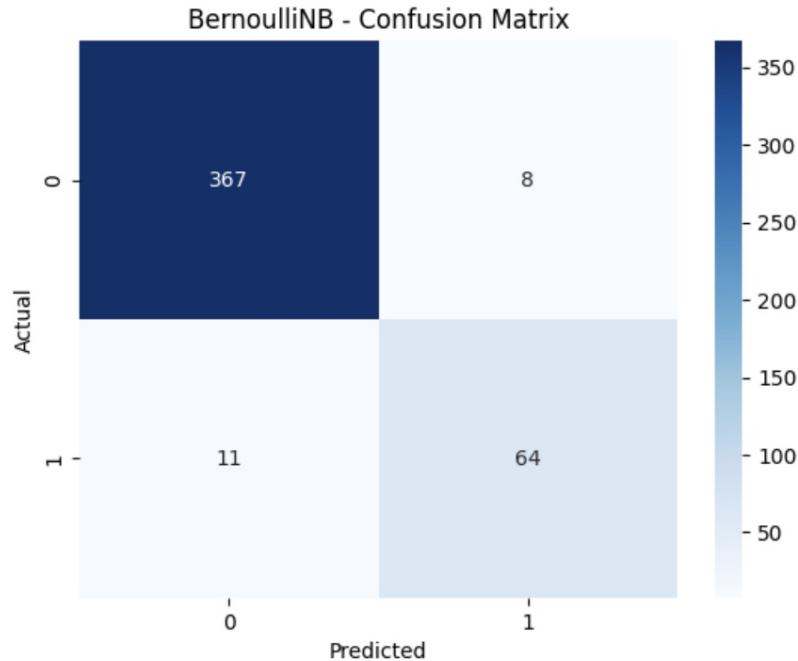
F-beta Score ($\beta=0.5$): 0.9749303621169917

Matthews Corr Coef: 0.9514624328283552



BernoulliNB Evaluation:
Accuracy: 0.9577777777777777
Precision: 0.8888888888888888
Recall: 0.8533333333333334
F1 Score: 0.8707482993197279

F-beta Score ($\beta=0.5$): 0.8815426997245179
Matthews Corr Coef: 0.8457800632220621

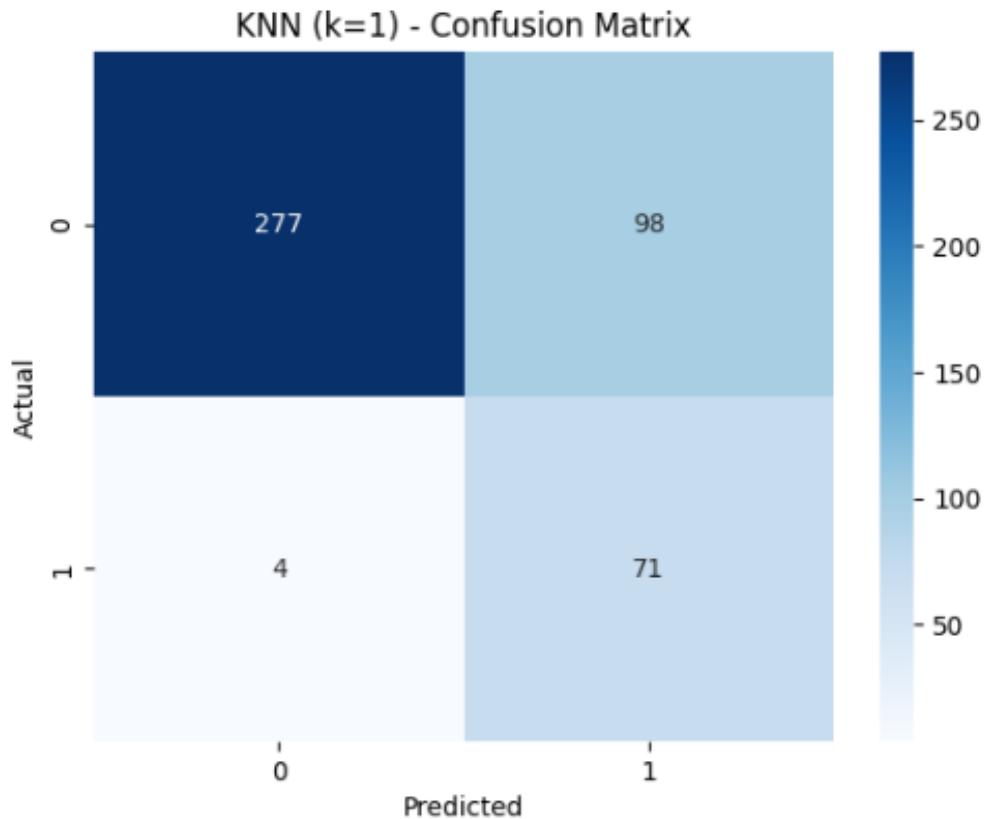


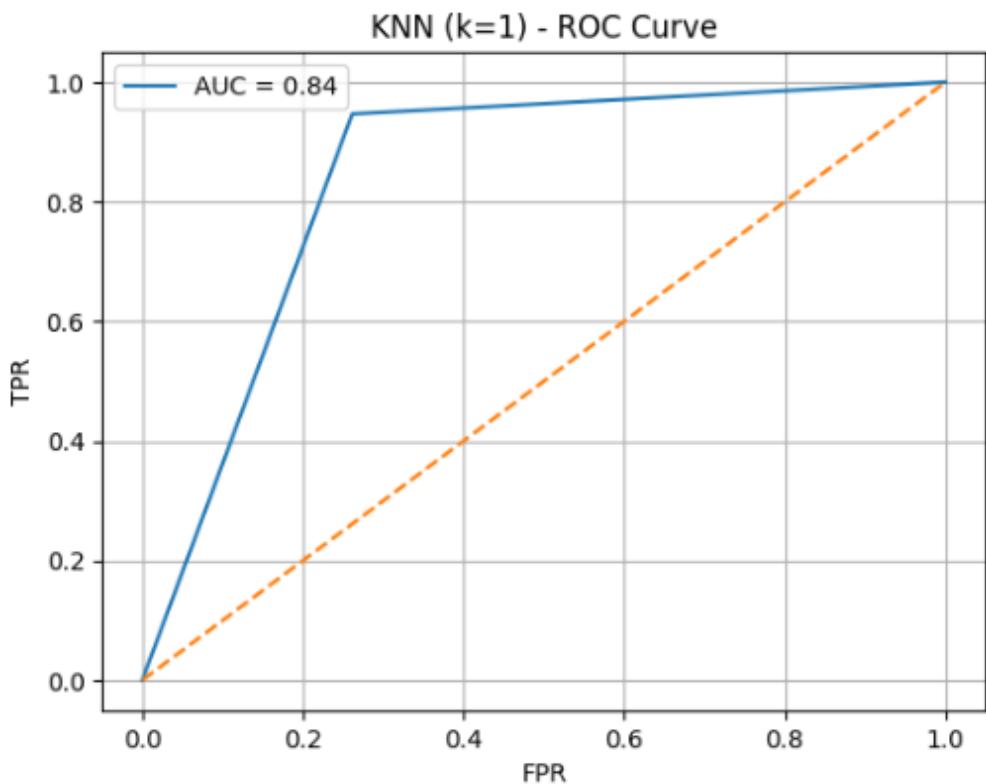
```
# ===== 9. KNN =====
for k in [1, 3, 5, 7]:
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train_scaled, y_train_scaled)
    evaluate_model(model, f"KNN (k={k})", X_test_scaled, y_test_scaled)
```

```
for algo in ["kd_tree", "ball_tree"]:  
    model = KNeighborsClassifier(algorithm=algo)  
    model.fit(X_train_scaled, y_train_scaled)  
    evaluate_model(model, f"KNN ({algo})", X_test_scaled, y_test_scaled)
```

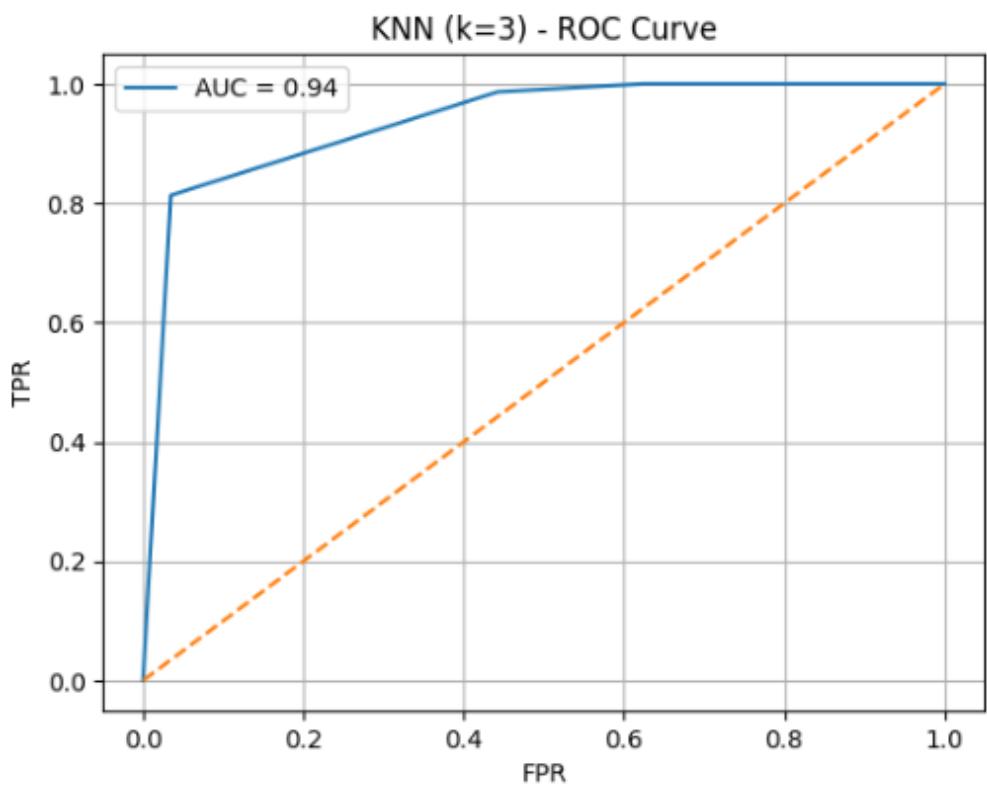
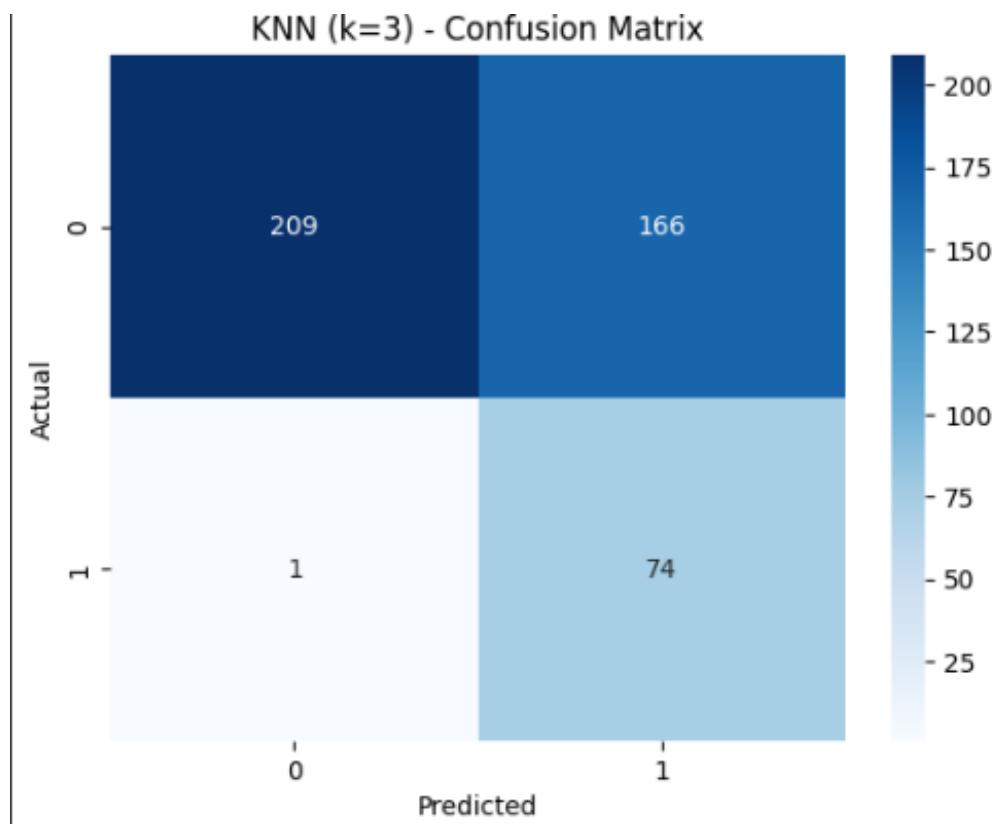
OUTPUT

KNN (k=1) Evaluation:
Accuracy: 0.7733333333333333
Precision: 0.42011834319526625
Recall: 0.9466666666666667
F1 Score: 0.5819672131147541
F-beta Score ($\beta=0.5$): 0.47270306258322237
Matthews Corr Coef: 0.5274139454909135





KNN (k=3) Evaluation:
Accuracy: 0.6288888888888889
Precision: 0.3083333333333335
Recall: 0.9866666666666667
F1 Score: 0.46984126984126984
F-beta Score ($\beta=0.5$): 0.357487922705314
Matthews Corr Coef: 0.40637772717369386



KNN (k=5) Evaluation:

Date: 08-08-2025

Experiment: 3

Name: Harini LV

Roll No: 3122237001016

Accuracy: 0.8688888888888889

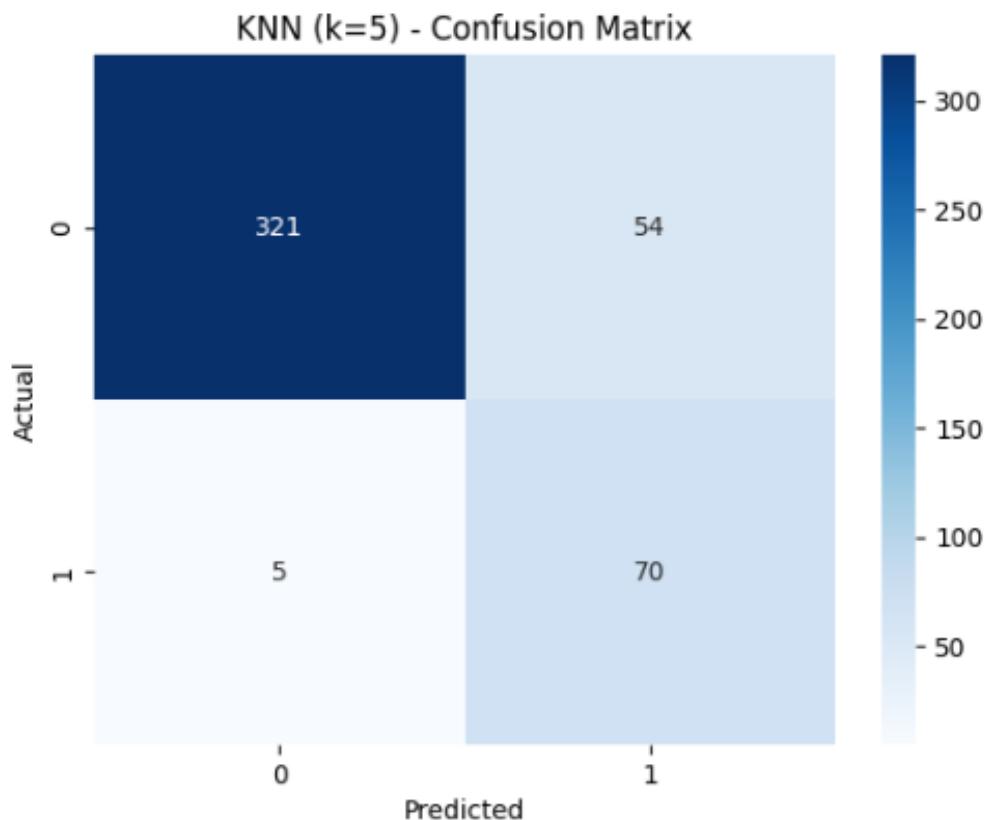
Precision: 0.5645161290322581

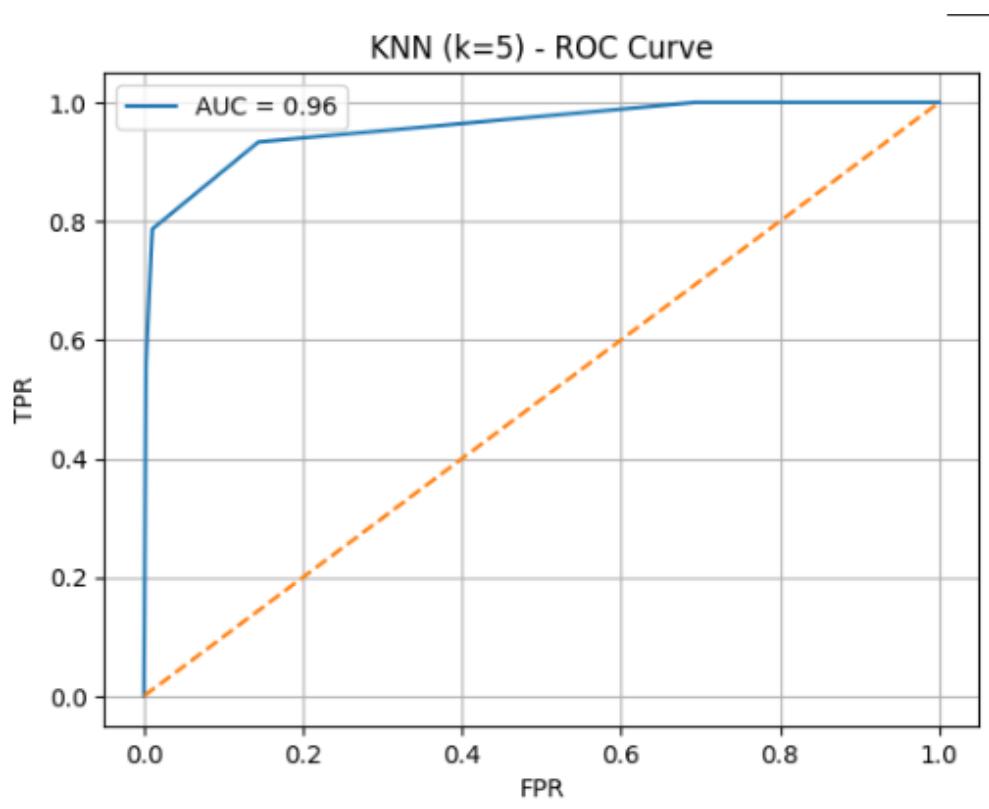
Recall: 0.9333333333333333

F1 Score: 0.7035175879396985

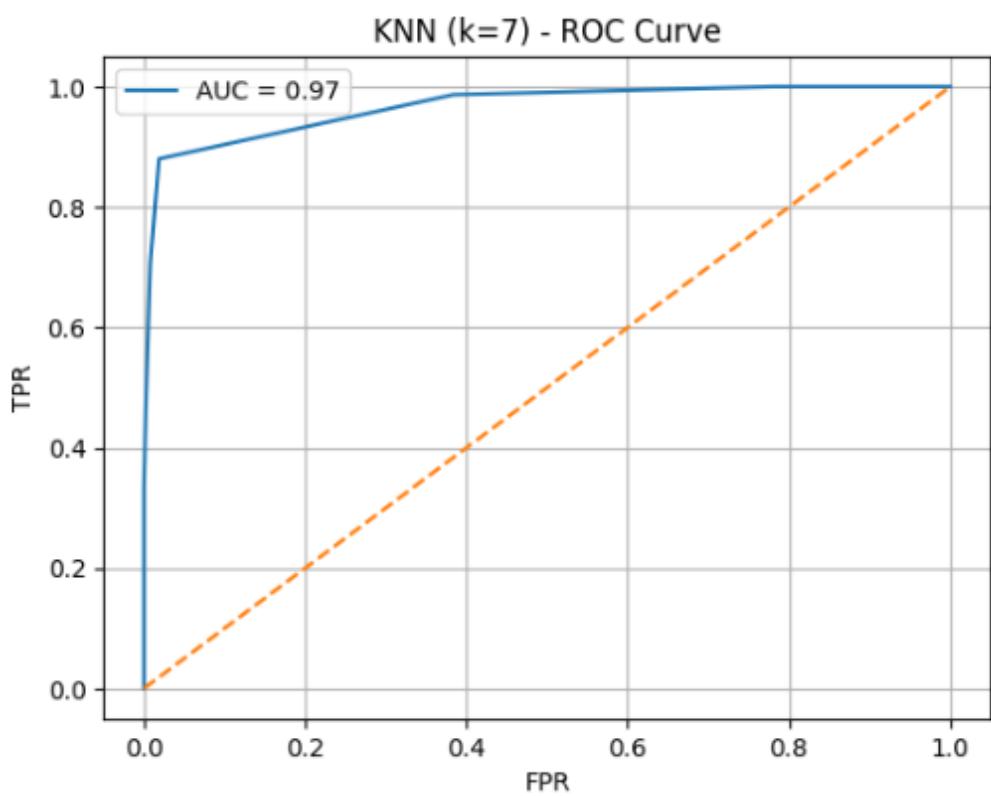
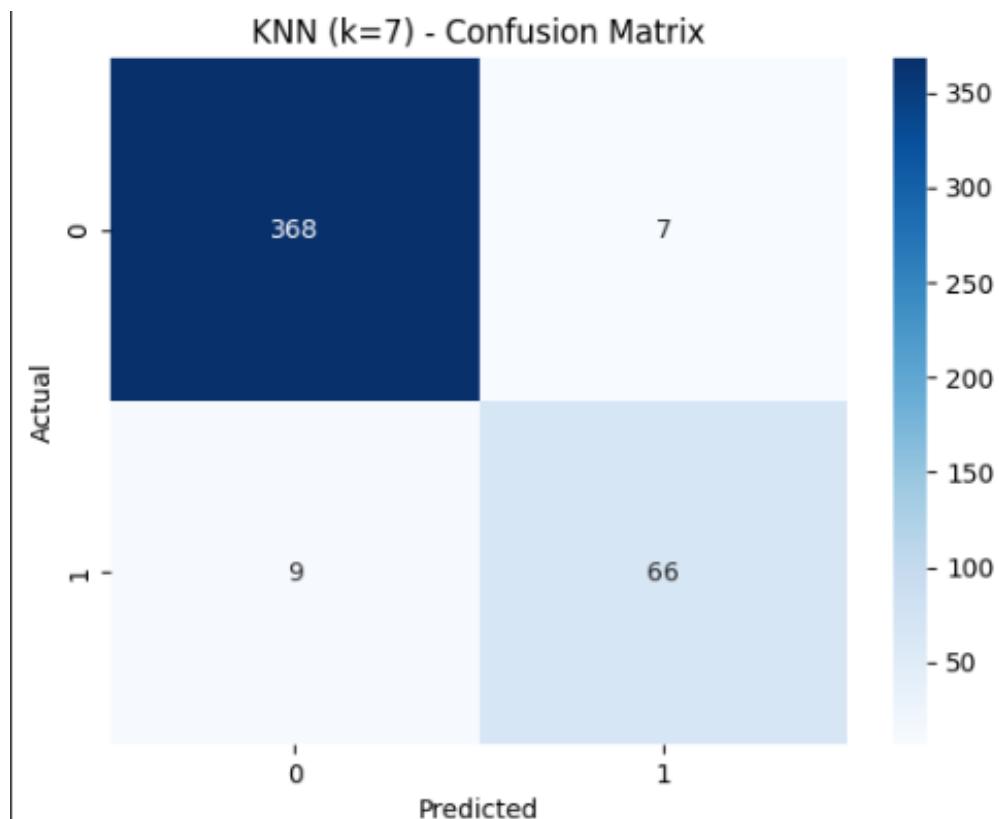
F-beta Score ($\beta=0.5$): 0.6129597197898424

Matthews Corr Coef: 0.6583958219651456



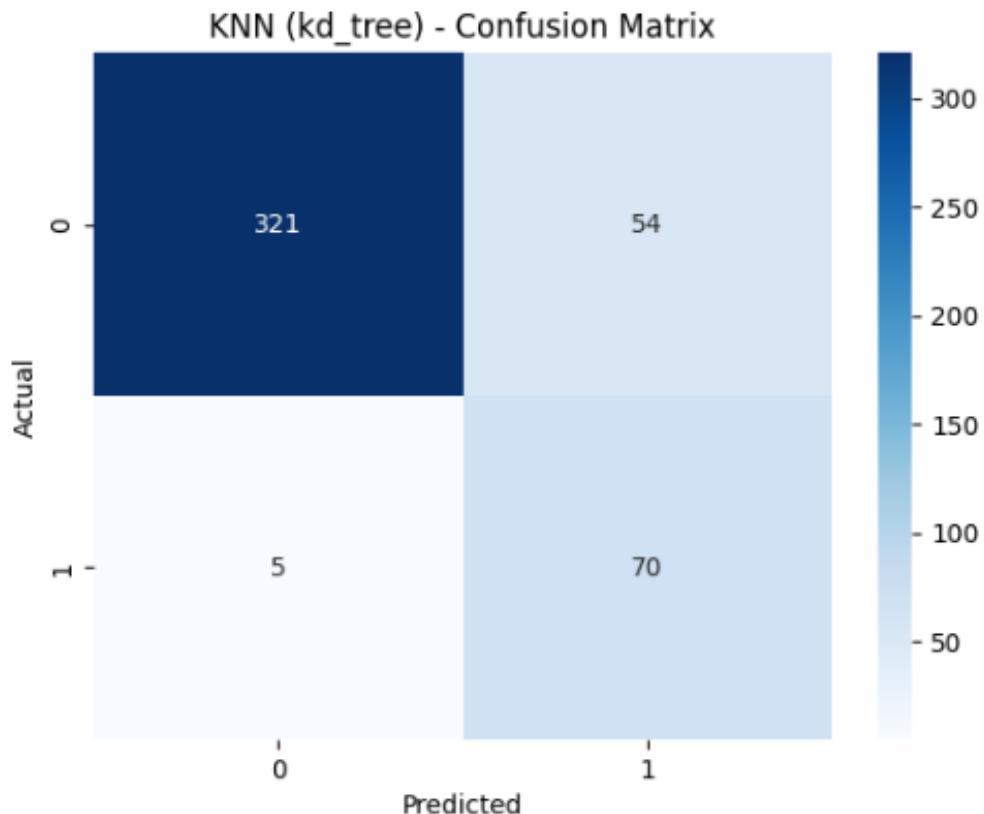


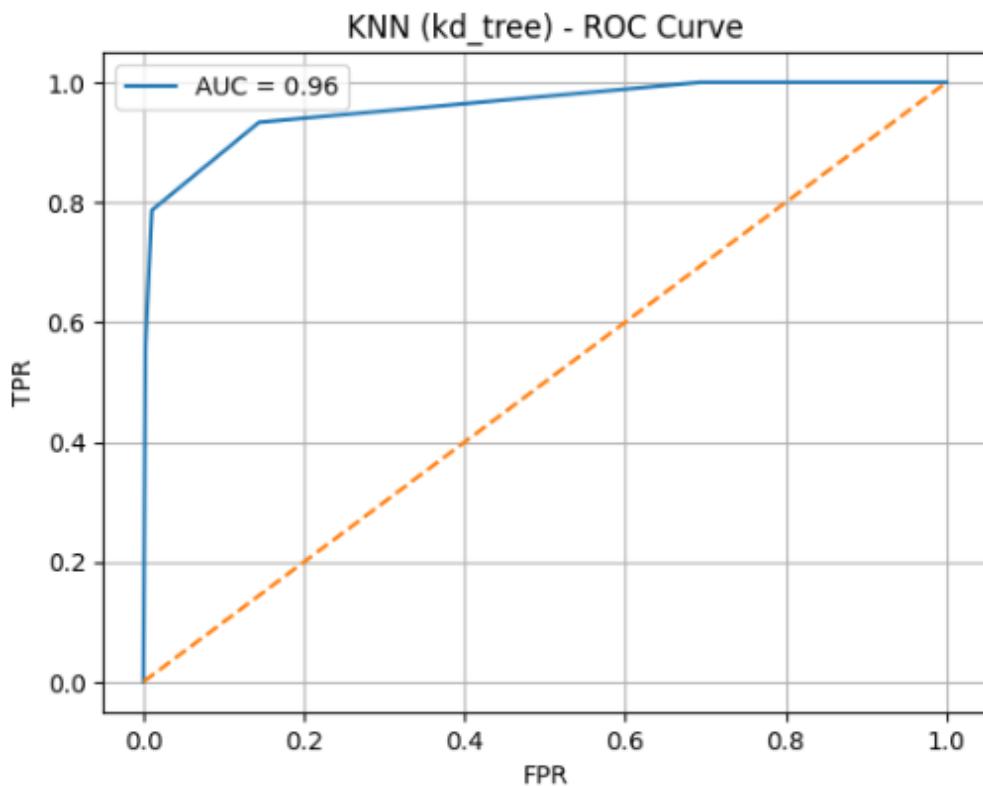
KNN (k=7) Evaluation:
Accuracy: 0.9644444444444444
Precision: 0.9041095890410958
Recall: 0.88
F1 Score: 0.8918918918918919
F-beta Score ($\beta=0.5$): 0.8991825613079019
Matthews Corr Coef: 0.8707338237428764



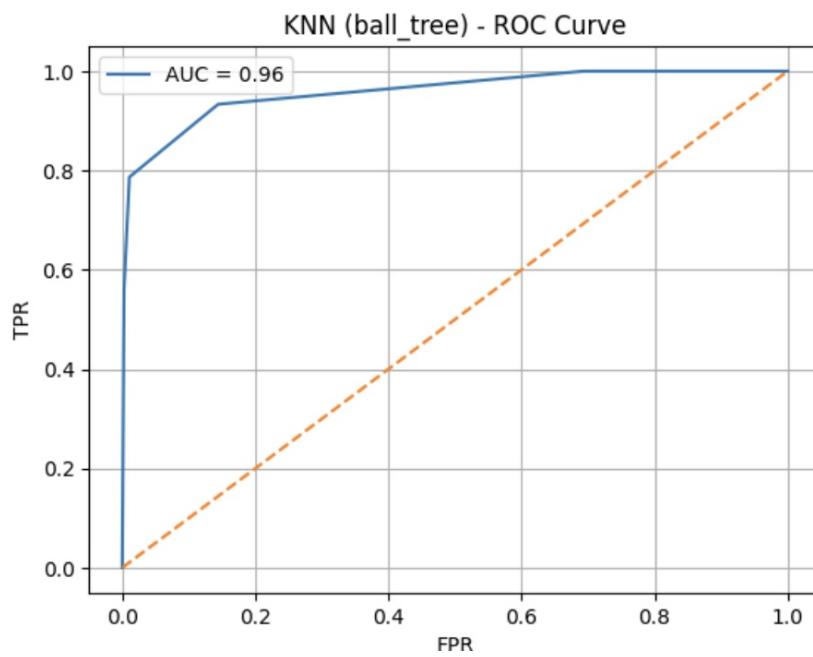
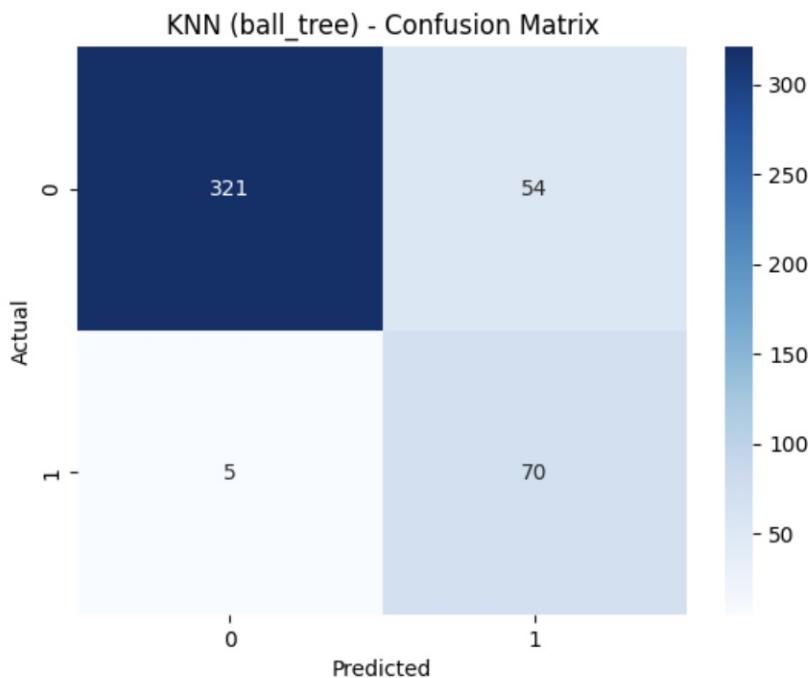
KNN (kd_tree) Evaluation:

Accuracy: 0.8688888888888889
Precision: 0.5645161290322581
Recall: 0.9333333333333333
F1 Score: 0.7035175879396985
F-beta Score ($\beta=0.5$): 0.6129597197898424
Matthews Corr Coef: 0.6583958219651456





KNN (ball_tree) Evaluation:
Accuracy: 0.8688888888888889
Precision: 0.5645161290322581
Recall: 0.9333333333333333
F1 Score: 0.7035175879396985
F-beta Score ($\beta=0.5$): 0.6129597197898424
Matthews Corr Coef: 0.6583958219651456

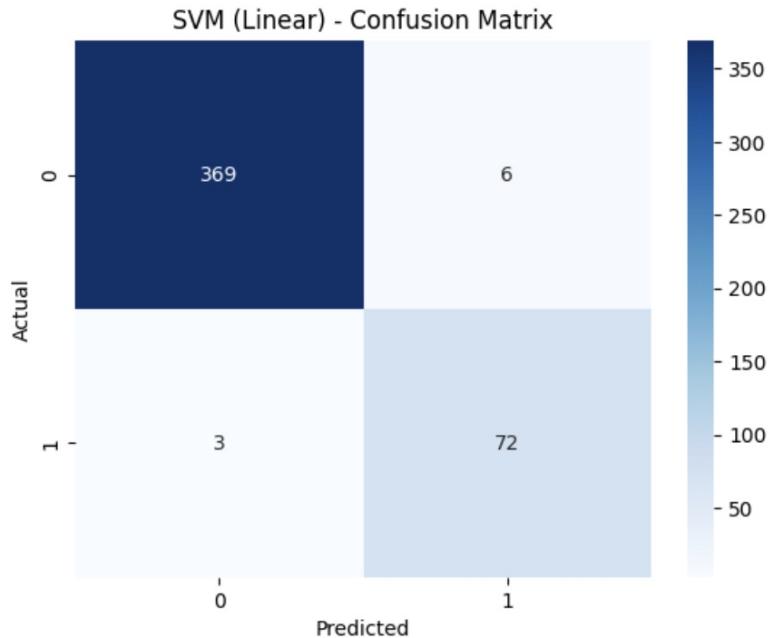


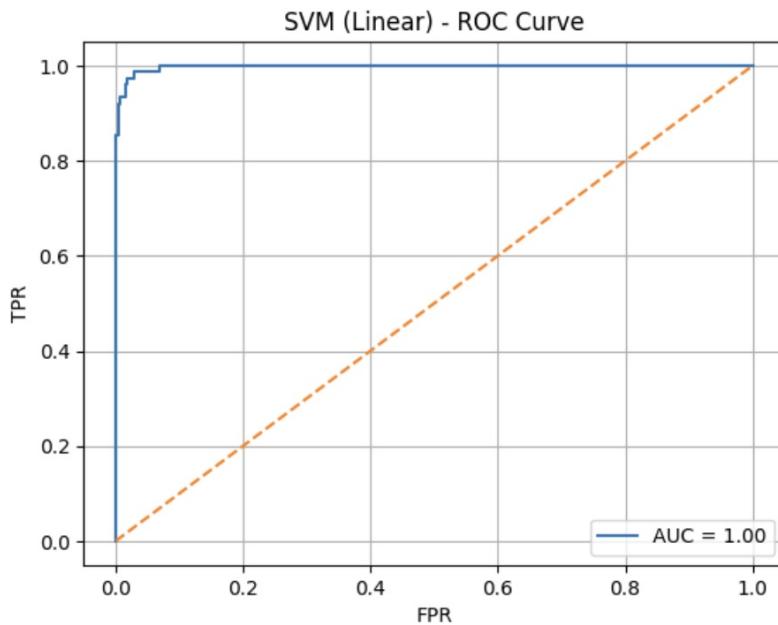
```
# ===== 10. SVM =====
kernels = {
    "Linear": SVC(kernel='linear', C=1, probability=True),
    "Polynomial": SVC(kernel='poly', C=1, degree=3, gamma='auto', probability=True),
    "RBF": SVC(kernel='rbf', C=1, gamma='scale', probability=True),
    "Sigmoid": SVC(kernel='sigmoid', C=1, gamma='auto', probability=True)
}
for name, model in kernels.items():
```

```
model.fit(X_train_scaled, y_train_scaled)
evaluate_model(model, f"SVM ({name})", X_test_scaled, y_test_scaled)
```

OUTPUT

SVM (Linear) Evaluation:
Accuracy: 0.98
Precision: 0.9230769230769231
Recall: 0.96
F1 Score: 0.9411764705882353
F-beta Score ($\beta=0.5$): 0.9302325581395349
Matthews Corr Coef: 0.9293931956705993





SVM (Polynomial) Evaluation:

Accuracy: 0.9044444444444445

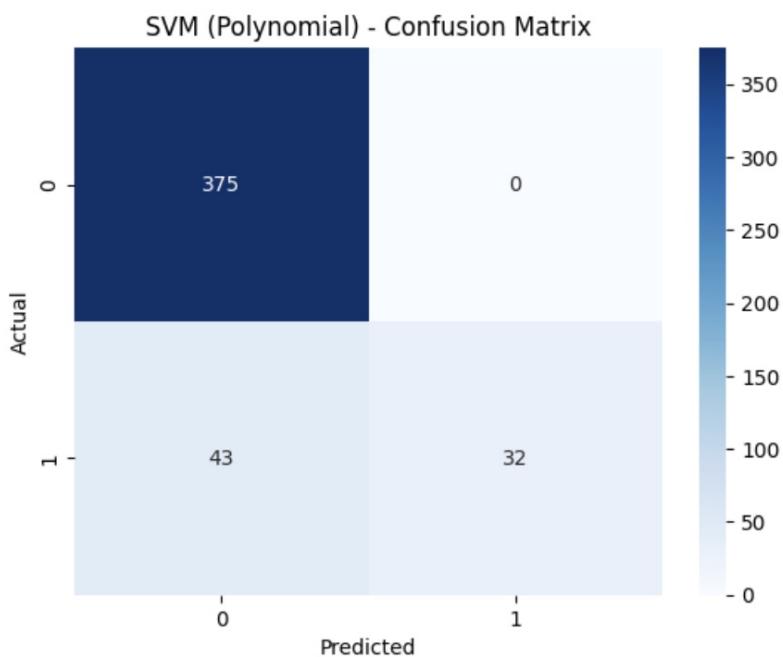
Precision: 1.0

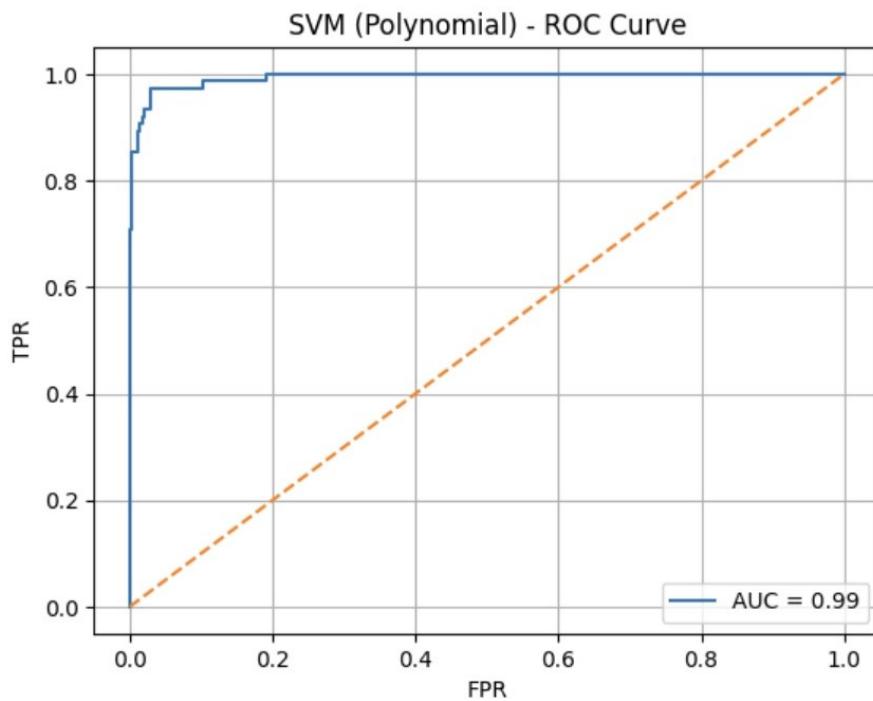
Recall: 0.4266666666666667

F1 Score: 0.5981308411214953

F-beta Score ($\beta=0.5$): 0.7881773399014779

Matthews Corr Coef: 0.6186882248897461





SVM (RBF) Evaluation:

Accuracy: 0.9844444444444445

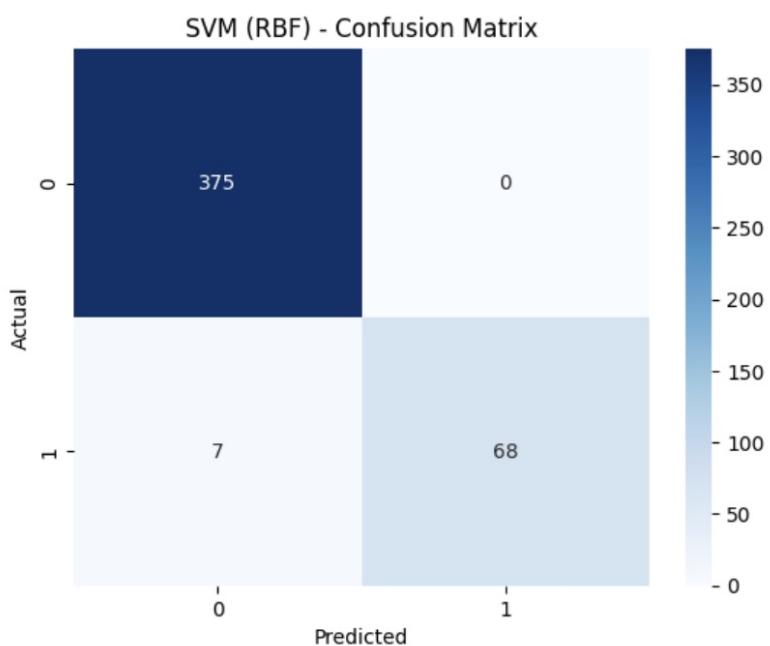
Precision: 1.0

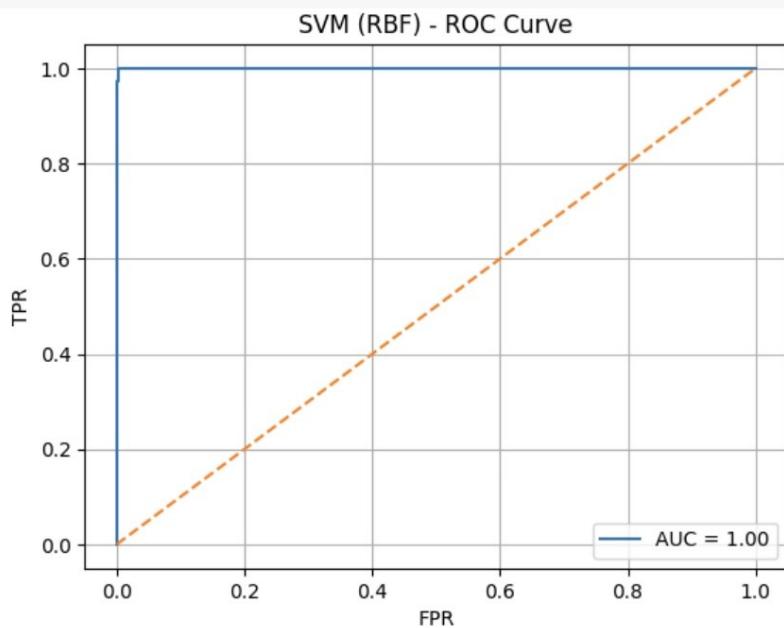
Recall: 0.9066666666666666

F1 Score: 0.951048951048951

F-beta Score ($\beta=0.5$): 0.9798270893371758

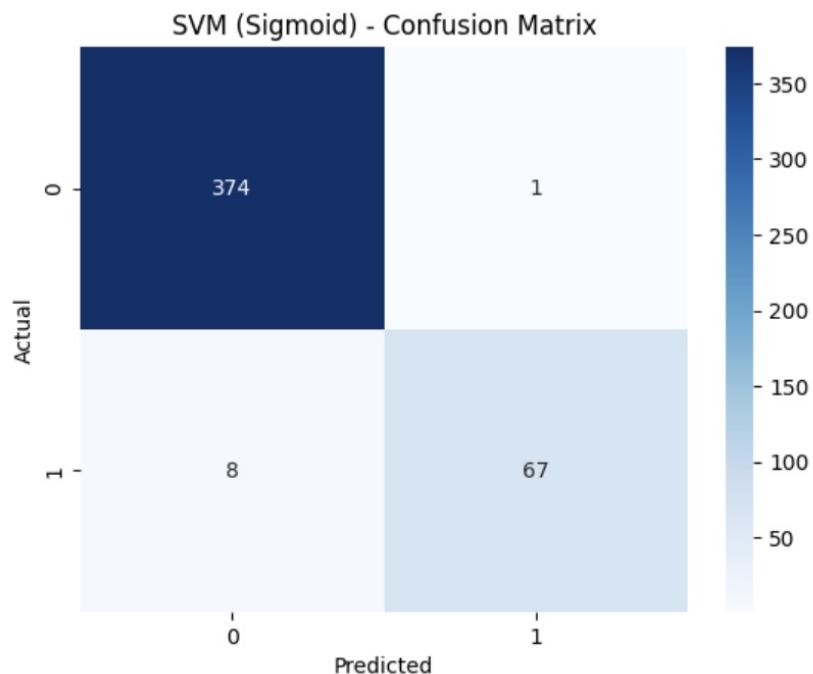
Matthews Corr Coef: 0.9434258614331825

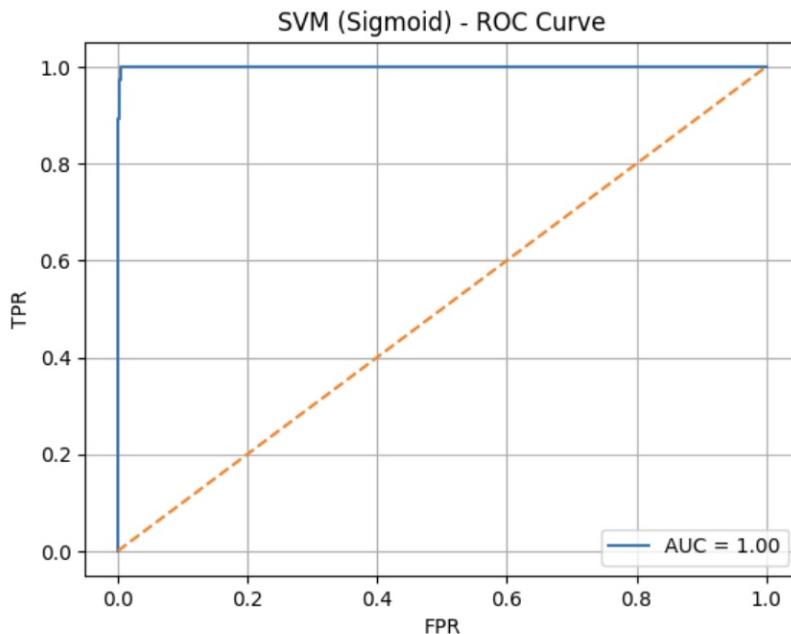




SVM (Sigmoid) Evaluation:

Accuracy: 0.98
Precision: 0.9852941176470589
Recall: 0.8933333333333333
F1 Score: 0.9370629370629371
F-beta Score ($\beta=0.5$): 0.9654178674351584
Matthews Corr Coef: 0.9267771697608322





```
# ===== 11. 5-Fold Cross Validation =====
# Common CV strategy
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

def evaluate_cv(model, name, X_data, y_data):
    print(f"\n5-Fold Cross Validation: {name}")
    scores = {
        "Accuracy": cross_val_score(model, X_data, y_data, cv=skf, scoring='accuracy').mean(),
        "Precision": cross_val_score(model, X_data, y_data, cv=skf, scoring='precision').mean(),
        "Recall": cross_val_score(model, X_data, y_data, cv=skf, scoring='recall').mean(),
        "F1 Score": cross_val_score(model, X_data, y_data, cv=skf, scoring='f1').mean()
    }
    for metric, score in scores.items():
        print(f"{metric}: {score:.4f}")

# =====
# Naive Bayes
# =====
for name, model in {
    "GaussianNB": GaussianNB(),
    "MultinomialNB": MultinomialNB(),
    "BernoulliNB": BernoulliNB()
}.items():
    evaluate_cv(model, name, X, y)

# =====
# KNN
```

```
# =====
for k in [1, 3, 5, 7]:
    model = KNeighborsClassifier(n_neighbors=k)
    evaluate_cv(model, f"KNN (k={k})", X_scaled, y)

for algo in ["kd_tree", "ball_tree"]:
    model = KNeighborsClassifier(algorithm=algo)
    evaluate_cv(model, f"KNN ({algo})", X_scaled, y)

# =====
# SVM Kernels
# =====
svm_kernels = {
    "SVM Linear": SVC(kernel='linear', C=1, probability=True),
    "SVM Polynomial": SVC(kernel='poly', C=1, degree=3, gamma='auto', probability=True),
    "SVM RBF": SVC(kernel='rbf', C=1, gamma='scale', probability=True),
    "SVM Sigmoid": SVC(kernel='sigmoid', C=1, gamma='auto', probability=True)
}
for name, model in svm_kernels.items():
    evaluate_cv(model, name, X_scaled, y)
```

OUTPUT

Table 1: Naive Bayes Models (5-Fold Cross Validation)

Model	Accuracy	Precision	Recall	F1 Score
GaussianNB	0.9647	0.9099	0.8758	0.8917
MultinomialNB	0.9770	0.9754	0.8839	0.9270
BernoulliNB	0.9507	0.8618	0.8398	0.8497

Table 2: KNN Models (5-Fold Cross Validation)

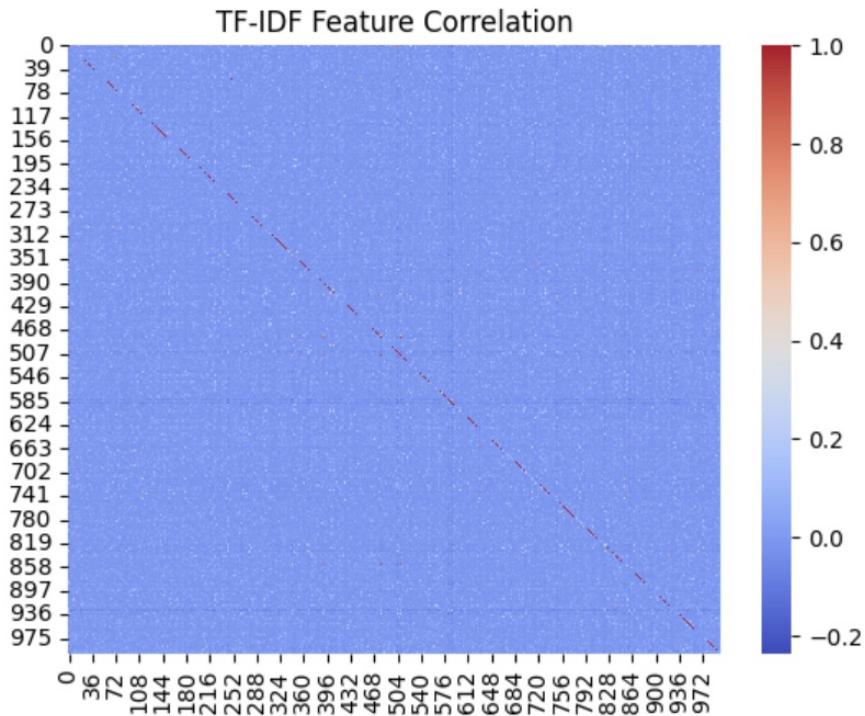
Model	Accuracy	Precision	Recall	F1 Score
KNN (k=1)	0.8056	0.4569	0.8858	0.6025
KNN (k=3)	0.7619	0.4659	0.9139	0.5922
KNN (k=5)	0.9196	0.7654	0.8178	0.7759
KNN (k=7)	0.9423	0.9380	0.7036	0.8001
KNN (kd_tree)	0.9193	0.7646	0.8178	0.7754
KNN (ball_tree)	0.9196	0.7654	0.8178	0.7759

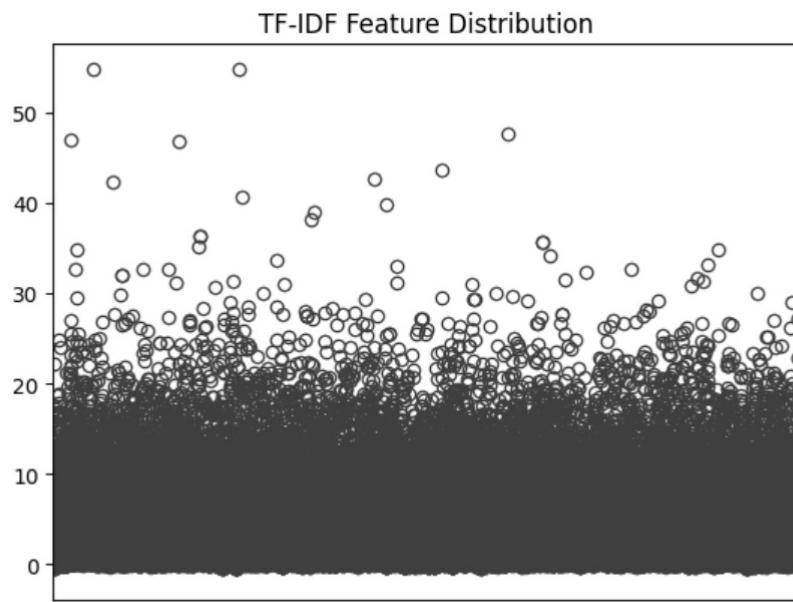
Table 3: SVM Models (5-Fold Cross Validation)

Model	Accuracy	Precision	Recall	F1 Score
SVM Linear	0.9660	0.8714	0.9339	0.9015
SVM Polynomial	0.9053	1.0000	0.4311	0.5996
SVM RBF	0.9733	0.9977	0.8418	0.9125
SVM Sigmoid	0.9753	0.9735	0.8759	0.9215

```
# ===== 12. Visualizations (Scaled TF-IDF) =====
sns.heatmap(pd.DataFrame(X_scaled).corr(), cmap='coolwarm')
plt.title("TF-IDF Feature Correlation")
plt.show()
```

```
sns.boxplot(data=pd.DataFrame(X_scaled))
plt.xticks([])
plt.title("TF-IDF Feature Distribution")
plt.show()
```





Observation:

- The dataset contained both spam and ham emails with some class imbalance, which was handled during preprocessing.
- Naïve Bayes performed efficiently, especially MultinomialNB, since it is well-suited for text-based frequency features.
- KNN accuracy varied with the value of k ; smaller k led to overfitting while larger k improved stability.
- SVM with the RBF kernel gave high accuracy and F1-score compared to linear and polynomial kernels, highlighting its ability to handle non-linear separability.
- ROC curves showed that SVM and Naïve Bayes achieved higher AUC values than KNN, making them more reliable for spam detection.
- Cross-validation confirmed that the models generalized well, with SVM achieving the most consistent results.

Inference:

- All three algorithms were effective in distinguishing spam from ham emails.
- Naïve Bayes was the simplest and fastest model, performing well with reasonable accuracy.
- KNN's performance was sensitive to hyperparameters such as k and distance metrics.

Date: 08-08-2025

Experiment: 3

Name: Harini LV

Roll No: 3122237001016

- SVM, particularly with the RBF kernel, emerged as the best performer in terms of accuracy, precision, recall, and F1-score.
- Ensemble approaches combining these classifiers could further enhance classification performance.

Learning outcomes:

- Gained hands-on experience in applying three different supervised classification algorithms (Naïve Bayes, KNN, SVM).
- Understood the importance of data preprocessing, normalization, and class balance in text classification tasks.
- Learned how hyperparameters like k (KNN) and kernel choice (SVM) influence model performance.
- Explored evaluation metrics (Accuracy, Precision, Recall, F1-score, ROC/AUC) for binary classification problems.
- Learned to apply K-Fold Cross Validation for fair and unbiased model evaluation.
- Developed skills in comparing and interpreting multiple ML models to identify the most effective one.