EXPT – 2 : Email Spam or Ham Classification

REPORT BY GOPIKA GANESAN

AIM:

Email Spam or Ham Classification using Naive Bayes, KNN, and SVM.

LIBRARIES USED:

NumPy, pandas, scikit learn, seaborn, matplotlib

OBJECTIVE:

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

CODE FOR ALL VARIANTS AND MODELS:

1.Naive Bayes

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

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

from sklearn.metrics import (
    confusion_matrix, classification_report, roc_auc_score, roc_curve,
    accuracy_score, precision_score, recall_score, f1_score, fbeta_score
)

from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB

# 3. EDA & Handling Missing Values
print("\nMissing Values:\n")
print(df.isnull().sum())
target="class"
# 4. Outlier Detection Function
def detect_outliers(df, col):
```

```python
def detect_outliers(df, col):
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    return df[(df[col] < Q1 - 1.5 * IQR) | (df[col] > Q3 + 1.5 * IQR)]

# 5. Train-Test Split
X = df.drop(columns=target)
y = df[target]

X_temp, X_test, y_temp, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, test_size=0.25, random_state=42)

# 6. Train and Evaluate
models = {
    "GaussianNB": GaussianNB(),
    "MultinomialNB": MultinomialNB(),
    "BernoulliNB": BernoulliNB()
}


gnb = GaussianNB()
gnb.fit(X_train, y_train)
y_pred_gnb = gnb.predict(X_val)
```

```python
print("\n GaussianNB ")
print("Accuracy:", accuracy_score(y_val, y_pred_gnb))
print("Precision:", precision_score(y_val, y_pred_gnb, average='weighted'))
print("Recall:", recall_score(y_val, y_pred_gnb, average='weighted'))
print("F1 Score:", f1_score(y_val, y_pred_gnb, average='weighted'))
print("F-beta Score (β=0.5):", fbeta_score(y_val, y_pred_gnb, beta=0.5, average='weighted'))

print("\nConfusion Matrix:\n", confusion_matrix(y_val, y_pred_gnb))
print("\nClassification Report:\n", classification_report(y_val, y_pred_gnb))

if hasattr(gnb, "predict_proba"):
    y_proba_gnb = gnb.predict_proba(X_val)[:, 1]
    print("ROC AUC Score:", roc_auc_score(y_val, y_proba_gnb))
    fpr, tpr, _ = roc_curve(y_val, y_proba_gnb)
    plt.plot(fpr, tpr, label="GaussianNB")



mnb = MultinomialNB()
mnb.fit(X_train, y_train)
y_pred_mnb = mnb.predict(X_val)

print("\n MultinomialNB ")
print("Accuracy:", accuracy_score(y_val, y_pred_mnb))
print("Precision:", precision_score(y_val, y_pred_mnb, average='weighted'))
```

```
print("Accuracy:", accuracy_score(y_val, y_pred_mnb))
print("Precision:", precision_score(y_val, y_pred_mnb, average='weighted'))
print("Recall:", recall_score(y_val, y_pred_mnb, average='weighted'))
print("F1 Score:", f1_score(y_val, y_pred_mnb, average='weighted'))
print("F-beta Score (β=0.5):", fbeta_score(y_val, y_pred_mnb, beta=0.5, average='weighted'))

print("\nConfusion Matrix:\n", confusion_matrix(y_val, y_pred_mnb))
print("\nClassification Report:\n", classification_report(y_val, y_pred_mnb))

if hasattr(mnb, "predict_proba"):
    y_proba_mnb = mnb.predict_proba(X_val)[:, 1]
    print("ROC AUC Score:", roc_auc_score(y_val, y_proba_mnb))
    fpr, tpr, _ = roc_curve(y_val, y_proba_mnb)
    plt.plot(fpr, tpr, label="MultinomialNB")



bnb = BernoulliNB()
bnb.fit(X_train, y_train)
y_pred_bnb = bnb.predict(X_val)

print("\n BernoulliNB ")
print("Accuracy:", accuracy_score(y_val, y_pred_bnb))
print("Precision:", precision_score(y_val, y_pred_bnb, average='weighted'))
print("Recall:", recall_score(y_val, y_pred_bnb, average='weighted'))
```

```
print("F1 Score:", f1_score(y_val, y_pred_bnb, average='weighted'))
print("F-beta Score (β=0.5):", fbeta_score(y_val, y_pred_bnb, beta=0.5, average='weighted'))

print("\nConfusion Matrix:\n", confusion_matrix(y_val, y_pred_bnb))
print("\nClassification Report:\n", classification_report(y_val, y_pred_bnb))

if hasattr(bnb, "predict_proba"):
    y_proba_bnb = bnb.predict_proba(X_val)[:, 1]
    print("ROC AUC Score:", roc_auc_score(y_val, y_proba_bnb))
    fpr, tpr, _ = roc_curve(y_val, y_proba_bnb)
    plt.plot(fpr, tpr, label="BernoulliNB")


plt.title("ROC Curves")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.show()
```

2.KNN:

```
X = df.drop(columns="class")
y = df["class"]

X_temp, X_test, y_temp, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, test_size=0.25, random_state=42)


print("\n KDTree Algorithm (k=5)")
knn_kdtree = Pipeline([
    ("scaler", StandardScaler()),
    ("knn", KNeighborsClassifier(n_neighbors=5, algorithm="kd_tree"))
])
start_time = time.time()
knn_kdtree.fit(X_train, y_train)
training_time=time.time()-start_time
y_pred = knn_kdtree.predict(X_val)

print(f"Training Time: {training_time:.4f} seconds")
print("Accuracy:", accuracy_score(y_val, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_val, y_pred))
print("Classification Report:\n", classification_report(y_val, y_pred))
```

```
print("\n BallTree Algorithm (k=5)")
knn_balltree = Pipeline([
    ("scaler", StandardScaler()),
    ("knn", KNeighborsClassifier(n_neighbors=5, algorithm="ball_tree"))
])
start_time = time.time()
knn_balltree.fit(X_train, y_train)
training_time = time.time() - start_time
y_pred = knn_balltree.predict(X_val)

print(f"Training Time: {training_time:.4f} seconds")
print("Accuracy:", accuracy_score(y_val, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_val, y_pred))
print("Classification Report:\n", classification_report(y_val, y_pred))



print("\n (Varying K)")
for k in [3, 5, 7, 9]:
    knn_auto = Pipeline([
        ("scaler", StandardScaler()),
        ("knn", KNeighborsClassifier(n_neighbors=k, algorithm="auto"))
    ])
    knn_auto.fit(X_train, y_train)
```

```
    y_pred = knn_auto.predict(X_val)

    print(f"\n K = {k} ")
    print("Accuracy:", accuracy_score(y_val, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_val, y_pred))
    print("Classification Report:\n", classification_report(y_val, y_pred))
```

3.SVM

```
X = df.drop(columns="class")
y = df["class"]
classes = y.unique()
y_bin = label_binarize(y, classes=classes)
n_classes = y_bin.shape[1] if y_bin.ndim > 1 else 1
X_temp, X_test, y_temp, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, test_size=0.25, random_state=42)

# Kernels to try
kernels = ['linear', 'poly', 'rbf', 'sigmoid']

for kernel in kernels:
    print(f"\nSVM with {kernel} kernel")
    svm_pipeline = Pipeline([
        ("scaler", StandardScaler()),
        ("svm", SVC(kernel=kernel, probability=True))  # probability=True needed for ROC
    ])

    start_time = time.time()
    svm_pipeline.fit(X_train, y_train)
    training_time = time.time() - start_time

    y_pred = svm_pipeline.predict(X_val)
    y_proba = svm_pipeline.predict_proba(X_val)
```

```
    print(f"Training Time: {training_time:.4f} seconds")
    print("Accuracy:", accuracy_score(y_val, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_val, y_pred))
    print("Classification Report:\n", classification_report(y_val, y_pred))
    y_val_bin = label_binarize(y_val, classes=classes)
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_val_bin[:, i], y_proba[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    plt.figure()
    for i in range(n_classes):
        plt.plot(fpr[i], tpr[i], label=f'Class {classes[i]} (AUC = {roc_auc[i]:.2f})')
    plt.plot([0, 1], [0, 1], 'k--', lw=1)
    plt.title(f"ROC Curve - SVM ({kernel} kernel)")
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend(loc="lower right")
    plt.grid()
    plt.show()
```

```
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
svm_cv_pipeline = Pipeline([
    ("scaler", StandardScaler()),
    ("svm", SVC(kernel='linear'))
])
cv_scores = cross_val_score(svm_cv_pipeline, X, y, cv=kfold, scoring='accuracy')
print("Cross-Validation Accuracies:", cv_scores)
print("Mean CV Accuracy: {:.4f}".format(np.mean(cv_scores)))
```

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import loguniform, randint

# Base pipeline
pipeline = Pipeline([
    ("scaler", StandardScaler()),
    ("svm", SVC(probability=True))
])

# Parameter spaces for each kernel
param_spaces = {
    "linear":  {"svm__kernel": ["linear"],  "svm__C": loguniform(1e-3, 1e3)},
    "poly":    {"svm__kernel": ["poly"],    "svm__C": loguniform(1e-3, 1e3),
                "svm__degree": randint(2, 6), "svm__gamma": loguniform(1e-4, 1e1)},
    "rbf":     {"svm__kernel": ["rbf"],     "svm__C": loguniform(1e-3, 1e3),
                "svm__gamma": loguniform(1e-4, 1e1)},
    "sigmoid": {"svm__kernel": ["sigmoid"], "svm__C": loguniform(1e-3, 1e3),
                "svm__gamma": loguniform(1e-4, 1e1)}
}

best_params = {}
for kernel, params in param_spaces.items():
    print(f"\n=== {kernel.upper()} Kernel ===")
    search = RandomizedSearchCV(
        pipeline, param_distributions=params,
        n_iter=10, cv=3, scoring='accuracy',
        n_jobs=-1, random_state=42
```

```
        n_jobs=-1, random_state=42
    )
    search.fit(X_train, y_train)
    best_params[kernel] = search.best_params_
    print("Best Params:", search.best_params_)
    print("Best CV Accuracy:", search.best_score_)

    # Validation
    y_pred = search.best_estimator_.predict(X_val)
    print("Validation Accuracy:", accuracy_score(y_val, y_pred))
    print("Classification Report:\n", classification_report(y_val, y_pred))

print("\nSummary of Best Parameters:", best_params)
```

```python
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, classification_report
import numpy as np

# Pipeline
pipeline = Pipeline([
    ("scaler", StandardScaler()),
    ("svm", SVC(probability=True))
])

# Grid parameters
param_grid = [
    {"svm__kernel": ["linear"], "svm__C": np.logspace(-3, 3, 5)},
    {"svm__kernel": ["poly"], "svm__C": np.logspace(-3, 3, 5),
     "svm__degree": [2, 3, 4], "svm__gamma": ["scale", "auto"]},
    {"svm__kernel": ["rbf"], "svm__C": np.logspace(-3, 3, 5),
     "svm__gamma": ["scale", "auto"]},
    {"svm__kernel": ["sigmoid"], "svm__C": np.logspace(-3, 3, 5),
     "svm__gamma": ["scale", "auto"]}
]
```

```python
# Grid search
grid = GridSearchCV(pipeline, param_grid, cv=3, scoring="accuracy", n_jobs=-1)
grid.fit(X_train, y_train)

print("Best Parameters:", grid.best_params_)
print("Best CV Accuracy:", grid.best_score_)

# Validation
y_pred = grid.best_estimator_.predict(X_val)
print("Validation Accuracy:", accuracy_score(y_val, y_pred))
print("Classification Report:\n", classification_report(y_val, y_pred))
```

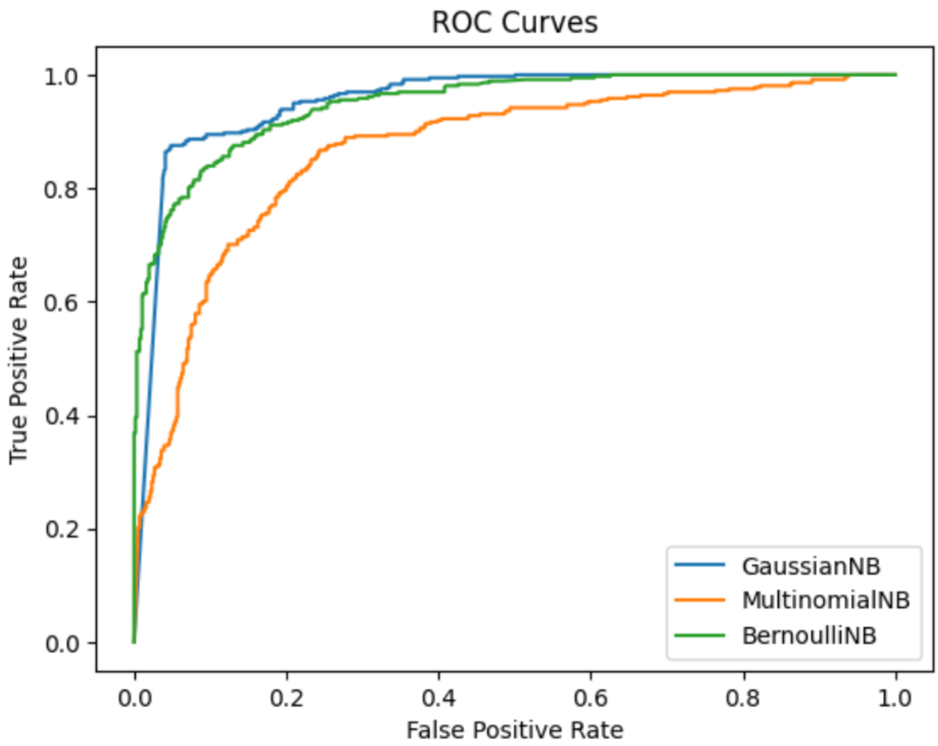
CONFUSION MATRIX AND ROC FOR EACH:

1.Naïve Bayes:

```
 GaussianNB
Accuracy: 0.8358695652173913
Precision: 0.8666295251955612
Recall: 0.8358695652173913
F1 Score: 0.8377216632152633
F-beta Score (β=0.5): 0.8517900866226876

Confusion Matrix:
 [[425 134]
 [ 17 344]]

Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.76      0.85       559
           1       0.72      0.95      0.82       361

    accuracy                           0.84       920
   macro avg       0.84      0.86      0.83       920
weighted avg       0.87      0.84      0.84       920

ROC AUC Score: 0.9545215784022716
```

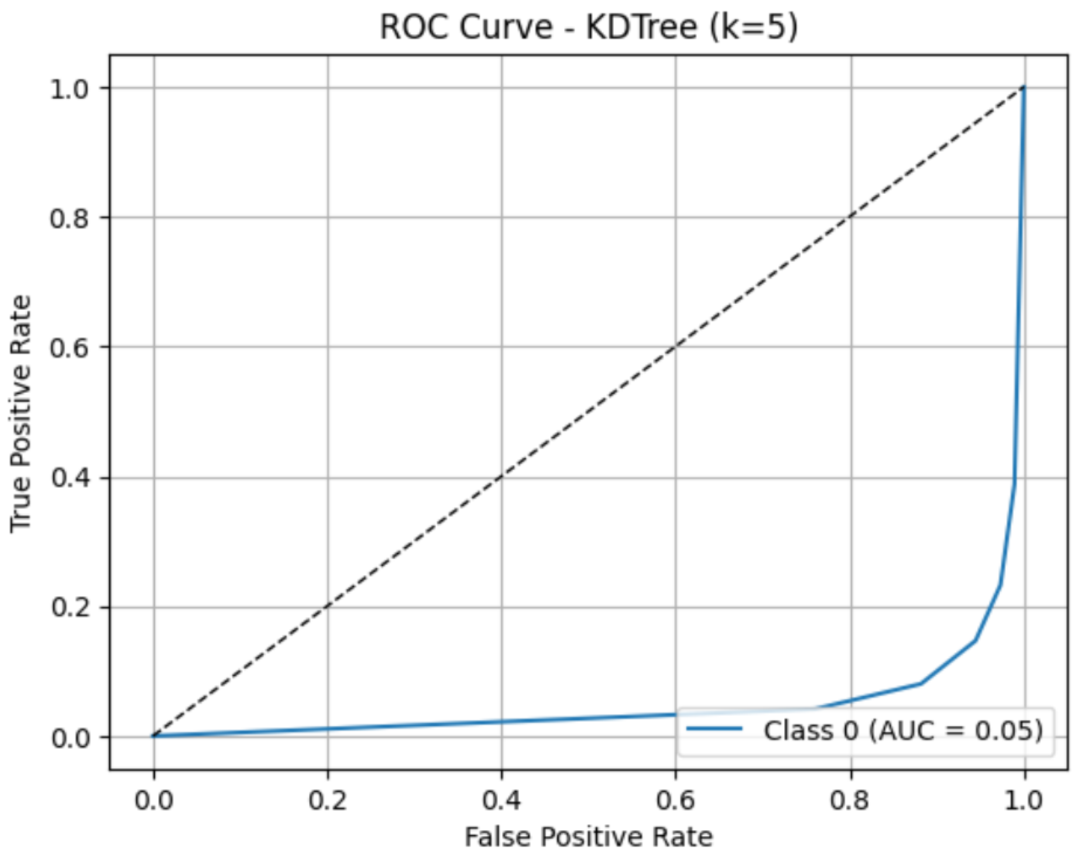
```
 MultinomialNB
Accuracy: 0.8
Precision: 0.7984079601990048
Recall: 0.8
F1 Score: 0.7985370950888192
F-beta Score (β=0.5): 0.7982959114000469

Confusion Matrix:
 [[480  79]
 [105 256]]

Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.86      0.84       559
           1       0.76      0.71      0.74       361

    accuracy                           0.80       920
   macro avg       0.79      0.78      0.79       920
weighted avg       0.80      0.80      0.80       920

ROC AUC Score: 0.8662803086239277
```

```
BernoulliNB
Accuracy: 0.8771739130434782
Precision: 0.8766807714723064
Recall: 0.8771739130434782
F1 Score: 0.8766128740859677
F-beta Score (β=0.5): 0.8765770388407951

Confusion Matrix:
 [[511  48]
 [ 65 296]]

Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.91      0.90       559
           1       0.86      0.82      0.84       361

    accuracy                           0.88       920
   macro avg       0.87      0.87      0.87       920
weighted avg       0.88      0.88      0.88       920

ROC AUC Score: 0.9488426602708634
```

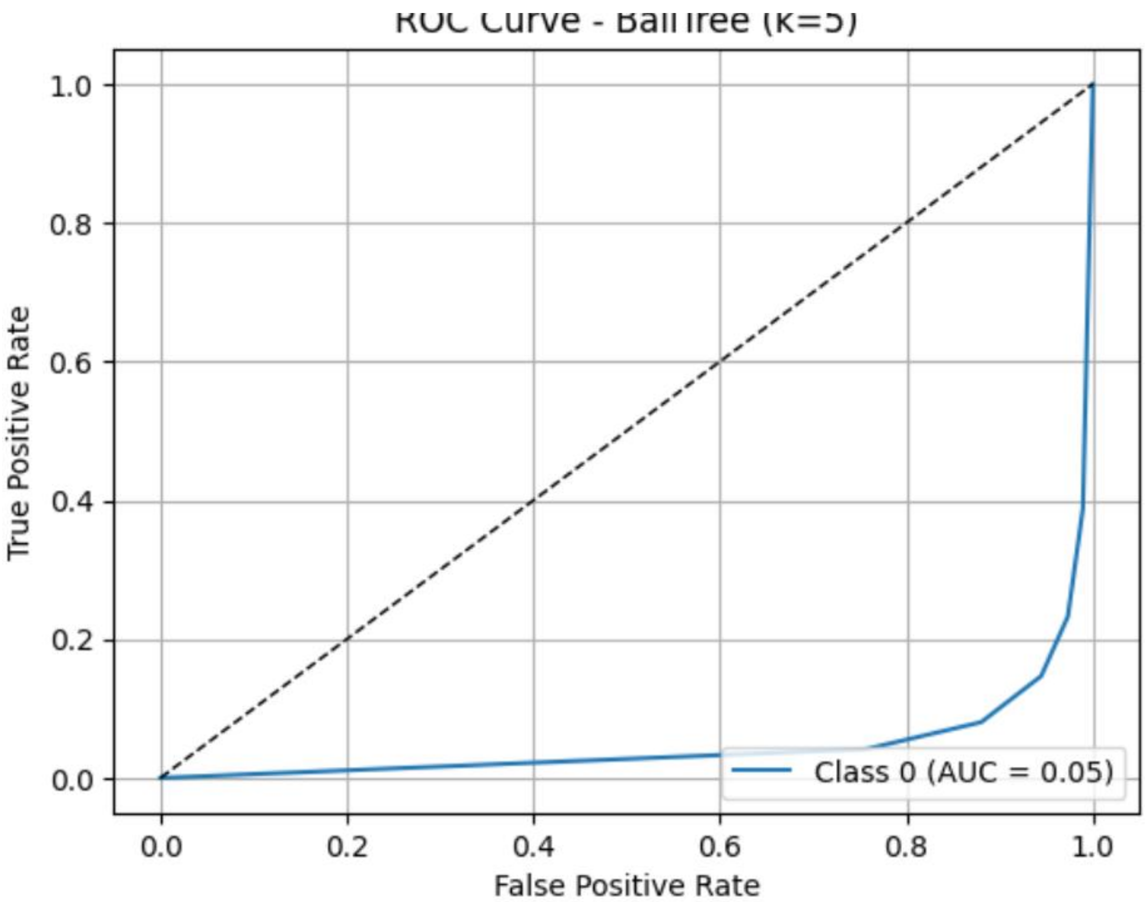


2.KNN

```
 KDTree Algorithm (k=5)
Training Time: 0.0596 seconds
Accuracy: 0.908695652173913
Confusion Matrix:
 [[528  31]
 [ 53 308]]
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.94      0.93       559
           1       0.91      0.85      0.88       361

    accuracy                           0.91       920
   macro avg       0.91      0.90      0.90       920
weighted avg       0.91      0.91      0.91       920
```



```
 BallTree Algorithm (k=5)
Training Time: 0.0728 seconds
Accuracy: 0.908695652173913
Confusion Matrix:
 [[528  31]
 [ 53 308]]
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.94      0.93       559
           1       0.91      0.85      0.88       361

    accuracy                           0.91       920
   macro avg       0.91      0.90      0.90       920
weighted avg       0.91      0.91      0.91       920
```



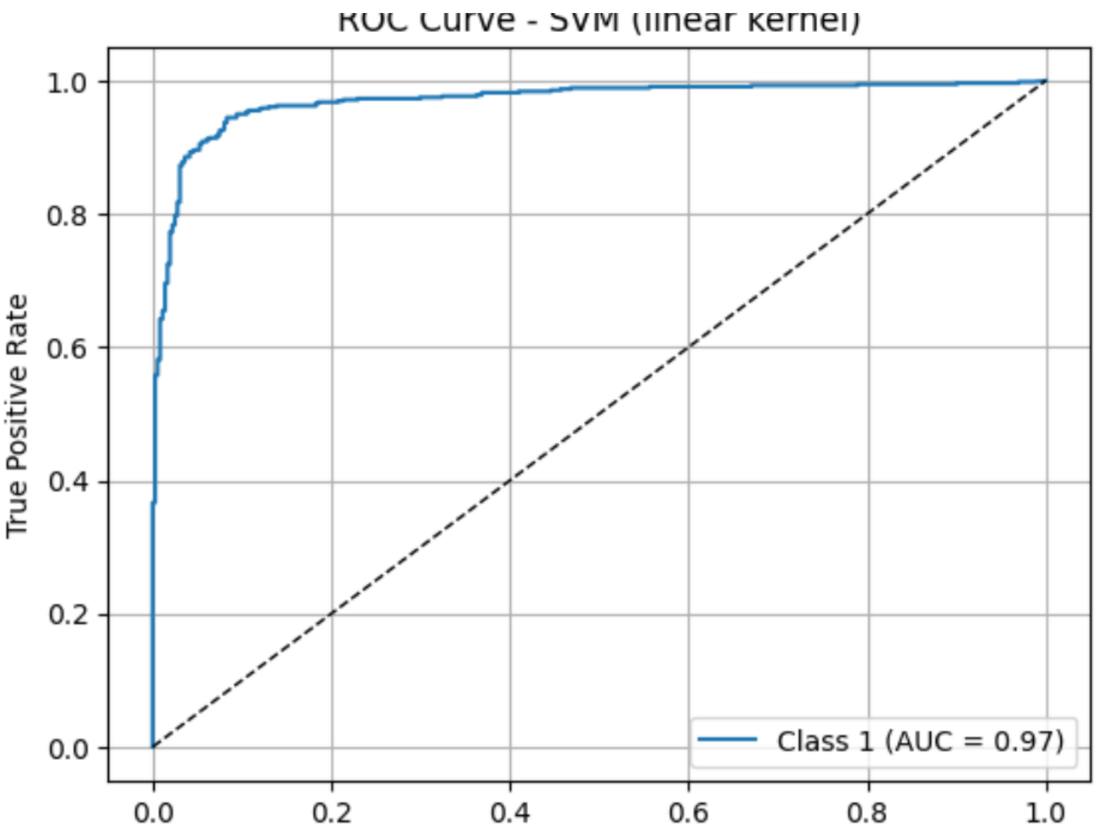
```
 K = 3
Accuracy: 0.9119565217391304
Confusion Matrix:
 [[524  35]
 [ 46 315]]
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.94      0.93       559
           1       0.90      0.87      0.89       361

    accuracy                           0.91       920
   macro avg       0.91      0.90      0.91       920
weighted avg       0.91      0.91      0.91       920
```

```
 K = 5
Accuracy: 0.908695652173913
Confusion Matrix:
 [[528  31]
 [ 53 308]]
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.94      0.93       559
           1       0.91      0.85      0.88       361

    accuracy                           0.91       920
   macro avg       0.91      0.90      0.90       920
weighted avg       0.91      0.91      0.91       920
```

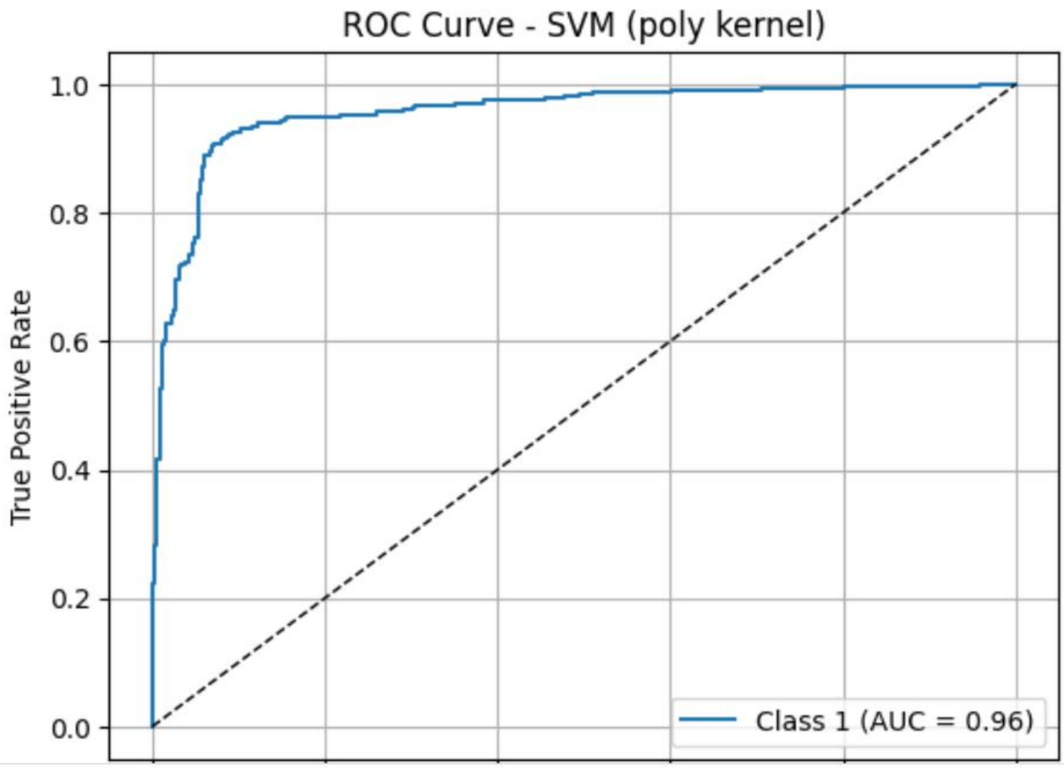
```
 K = 7
Accuracy: 0.9021739130434783
Confusion Matrix:
 [[526  33]
 [ 57 304]]
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.94      0.92       559
           1       0.90      0.84      0.87       361

    accuracy                           0.90       920
   macro avg       0.90      0.89      0.90       920
weighted avg       0.90      0.90      0.90       920
```

3.SVC

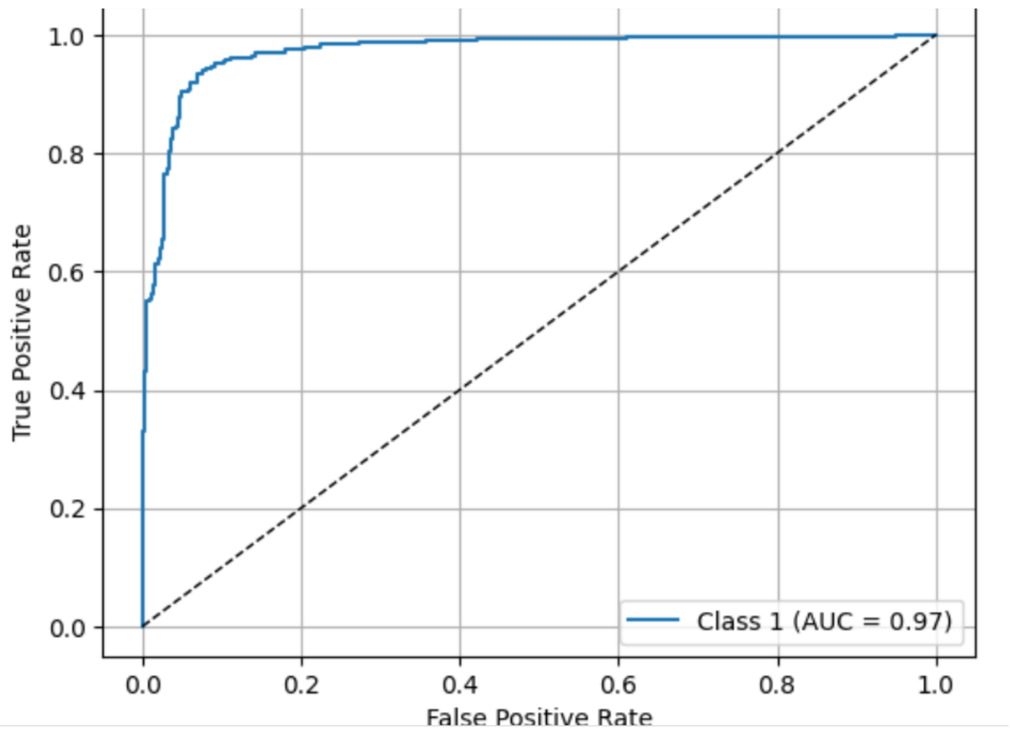
```
SVM with linear kernel
Training Time: 1.6313 seconds
Accuracy: 0.925
Confusion Matrix:
 [[536  23]
 [ 46 315]]
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.96      0.94       559
           1       0.93      0.87      0.90       361

    accuracy                           0.93       920
   macro avg       0.93      0.92      0.92       920
weighted avg       0.93      0.93      0.92       920
```



```
SVM with poly kernel
Training Time: 1.8258 seconds
Accuracy: 0.7760869565217391
Confusion Matrix:
 [[553   6]
 [200 161]]
Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.99      0.84       559
           1       0.96      0.45      0.61       361

    accuracy                           0.78       920
   macro avg       0.85      0.72      0.73       920
weighted avg       0.82      0.78      0.75       920
```

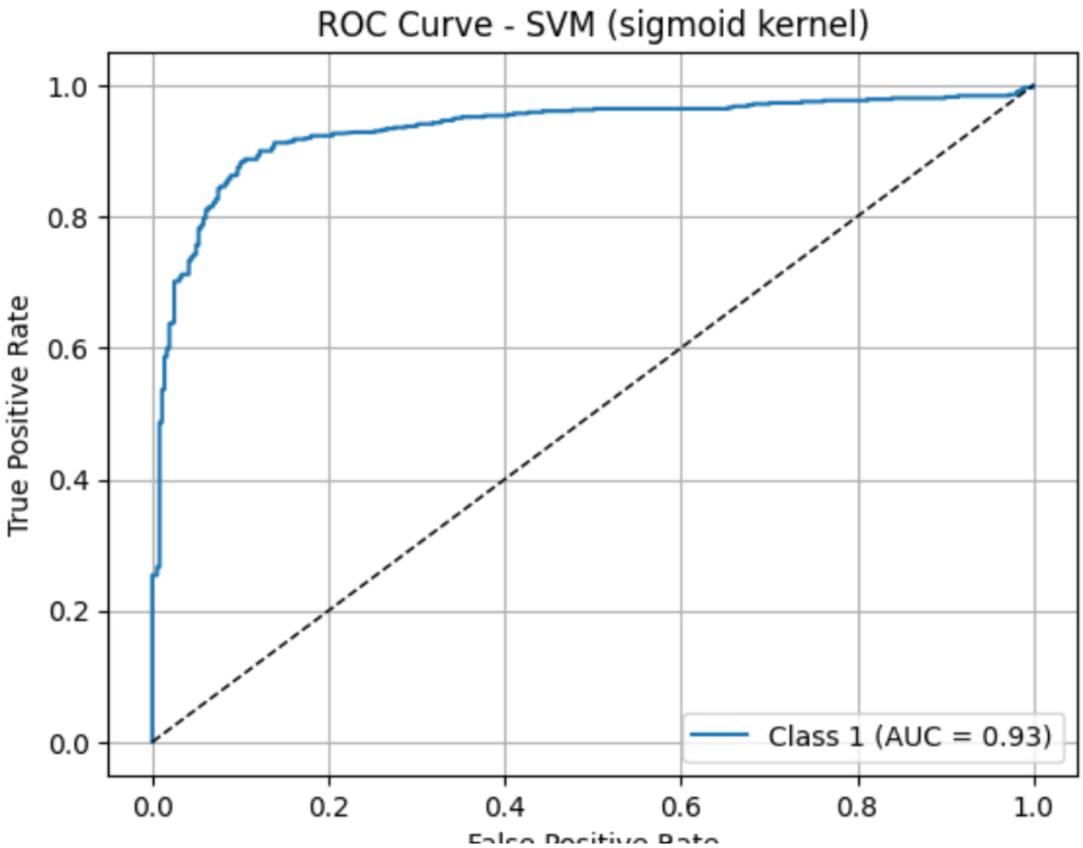


```
SVM with rbf kernel
Training Time: 1.1693 seconds
Accuracy: 0.9271739130434783
Confusion Matrix:
 [[538  21]
 [ 46 315]]
Classification Report:
               precision    recall  f1-score   support

           0       0.92      0.96      0.94       559
           1       0.94      0.87      0.90       361

    accuracy                           0.93       920
   macro avg       0.93      0.92      0.92       920
weighted avg       0.93      0.93      0.93       920
```



Terminal

```
SVM with sigmoid kernel
Training Time: 1.3583 seconds
Accuracy: 0.8847826086956522
Confusion Matrix:
 [[511  48]
 [ 58 303]]
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.91      0.91       559
           1       0.86      0.84      0.85       361

    accuracy                           0.88       920
   macro avg       0.88      0.88      0.88       920
weighted avg       0.88      0.88      0.88       920
```




```
K-Fold Cross Validation (k=5) using Linear Kernel
Cross-Validation Accuracies: [0.92616721 0.92717391 0.91630435 0.93804348 0.93043478]
Mean CV Accuracy: 0.9276
```

```
=== LINEAR Kernel ===
Best Params: {'svm__C': np.float64(157.41890047456639), 'svm__kernel': 'linear'}
Best CV Accuracy: 0.9301598570854347
Validation Accuracy: 0.9239130434782609
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.96      0.94       559
           1       0.94      0.86      0.90       361

    accuracy                           0.92       920
   macro avg       0.93      0.91      0.92       920
weighted avg       0.92      0.92      0.92       920
```

```
=== POLY Kernel ===
Best Params: {'svm__C': np.float64(0.10051981180656774), 'svm__degree': 5, 'svm__gamma': np.float64(0.3470266988650412), 'svm__kernel': 'poly'}
Best CV Accuracy: 0.9018962504426195
Validation Accuracy: 0.9760869565217392
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.98       559
           1       0.98      0.96      0.97       361

    accuracy                           0.98       920
   macro avg       0.98      0.97      0.97       920
weighted avg       0.98      0.98      0.98       920
```

```
=== RBF Kernel ===
Best Params: {'svm__C': np.float64(98.77700294007911), 'svm__gamma': np.float64(0.0011526449540315614), 'svm__kernel': 'rbf'}
Best CV Accuracy: 0.9372264789029509
Validation Accuracy: 0.9456521739130435
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.97      0.96       559
           1       0.95      0.91      0.93       361

    accuracy                           0.95       920
   macro avg       0.95      0.94      0.94       920
weighted avg       0.95      0.95      0.95       920
```

```
=== SIGMOID Kernel ===
Best Params: {'svm__C': np.float64(98.77700294007911), 'svm__gamma': np.float64(0.0011526449540315614), 'svm__kernel': 'sigmoid'}
Best CV Accuracy: 0.9149428328442916
Validation Accuracy: 0.9184782608695652
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.96      0.93       559
           1       0.93      0.86      0.89       361

    accuracy                           0.92       920
   macro avg       0.92      0.91      0.91       920
weighted avg       0.92      0.92      0.92       920
```

```
Best Parameters: {'svm__C': np.float64(1.0), 'svm__gamma': 'scale', 'svm__kernel': 'rbf'}
Best CV Accuracy: 0.9290740821989202
Validation Accuracy: 0.9434782608695652
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.97      0.95       559
           1       0.95      0.90      0.93       361

    accuracy                           0.94       920
   macro avg       0.94      0.94      0.94       920
weighted avg       0.94      0.94      0.94       920
```

COMPARISON TABLES:

Naïve Bayes Variant Comparison:

Metric	**Gaussian NB**	**Multinomial NB**	**Bernoulli NB**
Accuracy	0.836	0.8	0.877
Precision	0.866	0.799	0.877
Recall	0.836	0.8	0.877
F1 Score	0.838	0.799	0.877

KNN: Varying k values:

K	**Accuracy**	**Precision**	**Recall**	**F1 Score**
1	0.921	0.92	0.92	0.92
3	0.912	0.91	0.90	0.91
5	0.901	0.91	0.90	0.90
7	0.902	0.90	0.89	0.90

KNN: KDTree vs BallTree:

Metric	**KDTree**	**BallTree**
Accuracy	0.909	0.909
Precision	0.91	0.91
Recall	0.90	0.90
F1 Score	0.90	0.90
Training Time (s)	0.0357	0.0244

K-Fold Cross-Validation Results (K = 5):

Fold	**Naıve Bayes Accuracy**	**KNN Accuracy**	**SVM Accuracy**
Fold 1	0.820	0.893	0.926

Fold 2	0.817	0.908	0.927
Fold 3	0.801	0.925	0.916
Fold 4	0.820	0.906	0.938
Fold 5	0.835	0.908	0.930
Average	0.819	0.908	0.927

SVM Performance with Different Kernels and Parameters

Kernel	Hyperparameters	Accuracy	F1 Score	Training Time
Linear	C=157.419	0.930	0.92	
Polynomial	C=0.101, degree=5, gamma=0.347	0.902	0.97	
RBF	C=98.77, gamma=0.0011	0.94	0.94	
Sigmoid	C=98.77, gamma=0.001,	0.92	0.91	

OBSERVATIONS:

1. **Which classifier had the best average accuracy?**
   SVM performed best with an average accuracy of 0.927, compared to KNN (0.908) and Naïve Bayes (0.819).
2. **Which Naïve Bayes variant worked best?**
   Bernoulli NB was the best, achieving 0.877 accuracy, higher than Gaussian NB (0.836) and Multinomial NB (0.800).
3. **How did KNN accuracy vary with k and tree type?**
   Accuracy decreased slightly as k increased: k=1 → 0.921

- k=3 → 0.912
- k=5 → 0.901

- k=7 → 0.902
  KDTree vs BallTree gave identical accuracy (0.909), but BallTree trained faster (0.024s vs 0.036s).

4. **Which SVM kernel was most effective?**
   The RBF kernel had the best performance overall with 0.94 accuracy and balanced precision/recall.

   Linear kernel: 0.930 accuracy

   Polynomial kernel: 0.902 accuracy (higher F1 of 0.97 but lower accuracy)

   Sigmoid kernel: 0.92 accuracy.

5. **How did hyperparameters influence performance?**

- For SVM, tuning C and gamma had a strong effect:
  - High C (≈98–157) improved Linear and RBF accuracy.
  - Polynomial kernel with degree=5 underperformed in accuracy despite good F1.
- For KNN, smaller k gave higher accuracy, while higher k smoothed predictions but reduced accuracy.
- For Naïve Bayes, model choice (distribution assumption) itself acted as the key hyperparameter: Bernoulli worked best for this dataset