Experiment 2: Spam or Ham Classification using Naïve Bayes, KNN, and SVM with Hyperparameter Optimization

Machine Learning Lab Report

Academic Year 2025–2026

Aim

To classify emails as spam or ham using Naïve Bayes, K-Nearest Neighbors (KNN), and Support Vector Machine (SVM), and to evaluate their performance using standard classification metrics, K-Fold cross-validation, and hyperparameter optimization via GridSearchCV.

Libraries Used

- pandas
- numpy
- matplotlib
- seaborn
- scikit-learn

Objective

- Load and preprocess the dataset
- Perform exploratory data analysis (EDA)
- Train classifiers: Naïve Bayes, KNN, and SVM

- Optimize KNN and SVM hyperparameters using GridSearchCV
- Evaluate using accuracy, precision, recall, F1-score, confusion matrix, and ROC curve
- Compare performance using 5-fold cross-validation

Code Snippets

1. Data Loading and Preprocessing:

```
df = pd.read_csv("spambase_csv.csv")
df.fillna(df.mean(), inplace=True)
X_raw = df.drop(columns=['class'])
y = df['class']
```

2. Train-Test Split and Scaling:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X_train_raw, X_test_raw, y_train_raw, y_test_raw = train_test_split(
    X_raw, y, test_size=0.2, random_state=42
)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_raw)
X_train_scaled, X_test_scaled, y_train_scaled, y_test_scaled =
   train_test_split(
    X_scaled, y, test_size=0.2, random_state=42
)
```

3. Evaluation Function:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score,
    f1_score, confusion_matrix, roc_curve, auc

def evaluate(name, model, X_test, y_test):
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    print(f"{name} -- Accuracy: {acc:.2f}, Precision: {prec:.2f}, Recall:
   {rec:.2f}, F1 Score: {f1:.2f}")
```

4. Training Gaussian Naïve Bayes:

```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(X_train_raw, y_train_raw)
evaluate("GaussianNB", model, X_test_raw, y_test_raw)
```

5. KNN with GridSearchCV for Optimal k:

```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

param_grid = {'n_neighbors': [1, 3, 5, 7]}
knn = KNeighborsClassifier()
knn_grid = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
knn_grid.fit(X_train_scaled, y_train_scaled)
best_k = knn_grid.best_params_['n_neighbors']

# Evaluate best K
best_knn = knn_grid.best_estimator_
evaluate(f"KNN (Best k={best_k})", best_knn, X_test_scaled, y_test_scaled)
```

6. KNN KDTree vs BallTree:

```
knn_kd = KNeighborsClassifier(n_neighbors=best_k, algorithm='kd_tree')
knn_kd.fit(X_train_scaled, y_train_scaled)
evaluate(f"KNN - KDTree (k={best_k})", knn_kd, X_test_scaled, y_test_scaled)

knn_bt = KNeighborsClassifier(n_neighbors=best_k, algorithm='ball_tree')
knn_bt.fit(X_train_scaled, y_train_scaled)
evaluate(f"KNN - BallTree (k={best_k})", knn_bt, X_test_scaled, y_test_scaled)
```

7. SVM with GridSearchCV for Different Kernels:

```
from sklearn.svm import SVC

# Linear
param_grid_linear = {'C':[0.1,1,10]}
grid_linear = GridSearchCV(SVC(kernel='linear', probability=True), param_grid_linear, cv=5)
grid_linear.fit(X_train_scaled, y_train_scaled)
evaluate(f"SVM - Linear (C={grid_linear.best_params_['C']})", grid_linear.best_estimator_, X_test_scaled, y_test_scaled)

# Polynomial
param_grid_poly = {'C':[0.1,1,10], 'degree':[2,3,4], 'gamma':['scale','auto']}
grid_poly = GridSearchCV(SVC(kernel='poly', probability=True), param_grid_poly, cv=5)
grid_poly.fit(X_train_scaled, y_train_scaled)
best_poly = grid_poly.best_estimator_
evaluate(f"SVM - Polynomial {grid_poly.best_params_}", best_poly, X_test_scaled, y_test_scaled)

# RBF
param_grid_rbf = {'C':[0.1,1,10], 'gamma':['scale','auto']}
grid_rbf = GridSearchCV(SVC(kernel='rbf', probability=True), param_grid_rbf, cv=5)
```

```
grid_rbf.fit(X_train_scaled, y_train_scaled)
best_rbf = grid_rbf.best_estimator_
evaluate(f"SVM - RBF {grid_rbf.best_params_}", best_rbf, X_test_scaled,
    y_test_scaled)

# Sigmoid
param_grid_sig = {'C':[0.1,1,10], 'gamma':['scale','auto']}
grid_sig = GridSearchCV(SVC(kernel='sigmoid', probability=True),
    param_grid_sig, cv=5)
grid_sig.fit(X_train_scaled, y_train_scaled)
best_sig = grid_sig.best_estimator_
evaluate(f"SVM - Sigmoid {grid_sig.best_params_}", best_sig, X_test_scaled
    , y_test_scaled)
```

Results and Comparisons

Table 1: Naïve Bayes Performance

Model	Accuracy	Precision	Recall	F1 Score
GaussianNB	0.820847	0.719298	0.946154	0.817276
MultinomialNB	0.786102	0.764384	0.715385	0.739073
BernoulliNB	0.880565	0.906977	0.800000	0.850136

Table 2: KNN (GridSearchCV) Performance

Model	Accuracy	Precision	Recall	F1 Score
Best KNN (k=7)	0.895765	0.910615	0.835897	0.871658
1	0.895765	0.876923	0.876923	0.876923
3	0.893594	0.888298	0.856140	0.872063
5	0.895765	0.901639	0.846154	0.873016

Table 3: KNN KDTree vs BallTree Performance

Metric	KDTree	BallTree
Accuracy	0.895765	0.895765
Precision	0.910615	0.910615
Recall	0.835897	0.835897
F1 Score	0.871658	0.871658
Training Time (s)	0.895523	0.908345

Table 4: SVM (GridSearchCV) Performance

Kernel	Best Hyperparameters	Accuracy	Precision	Recall	F1 Score	CV S
Linear	{C=1, kernel=linear}	0.925081	0.934959	0.884615	0.909091	0.92
Polynomial	{C=10, degree=2, gamma=scale}	0.918567	0.941176	0.861538	0.899598	0.91
RBF	{C=1, gamma=scale}	0.934853	0.950820	0.892308	0.920635	0.93
Sigmoid	{C=0.1, gamma=scale}	0.891422	0.921512	0.812821	0.863760	0.89

Table 5: K-Fold CV Accuracies (K=5)

Fold	GaussianNB	Best KNN	Linear SVM	Poly SVM	RBF SVM	Sigmoid SVM
1	0.820847	0.895765	0.925081	0.918567	0.934853	0.891422
2	0.817391	0.913043	0.928261	0.925000	0.933696	0.895652
3	0.801087	0.913043	0.916304	0.919565	0.922826	0.894565
4	0.820652	0.900000	0.936957	0.919565	0.935870	0.902174
5	0.835870	0.911957	0.930435	0.915217	0.930435	0.891304
Average	0.819169	0.906762	0.927408	0.919583	0.931536	0.895024

ROC Curves and Confusion Matrices

Each model's ROC curve and confusion matrix are shown below:

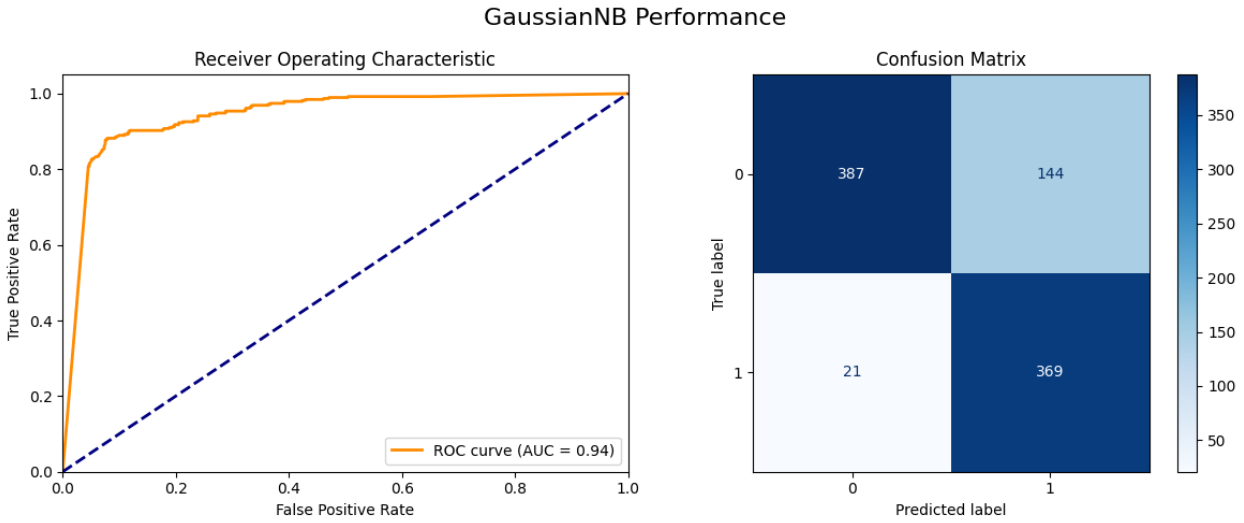

Figure 1: Gaussian Naïve Bayes Results

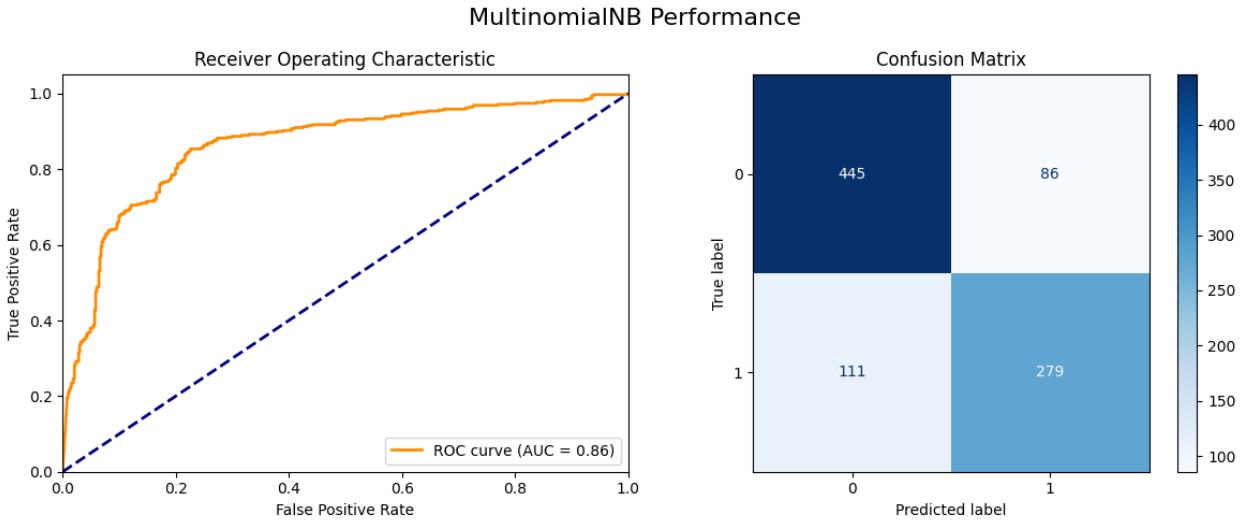

Figure 2: Multinomial Naïve Bayes Results

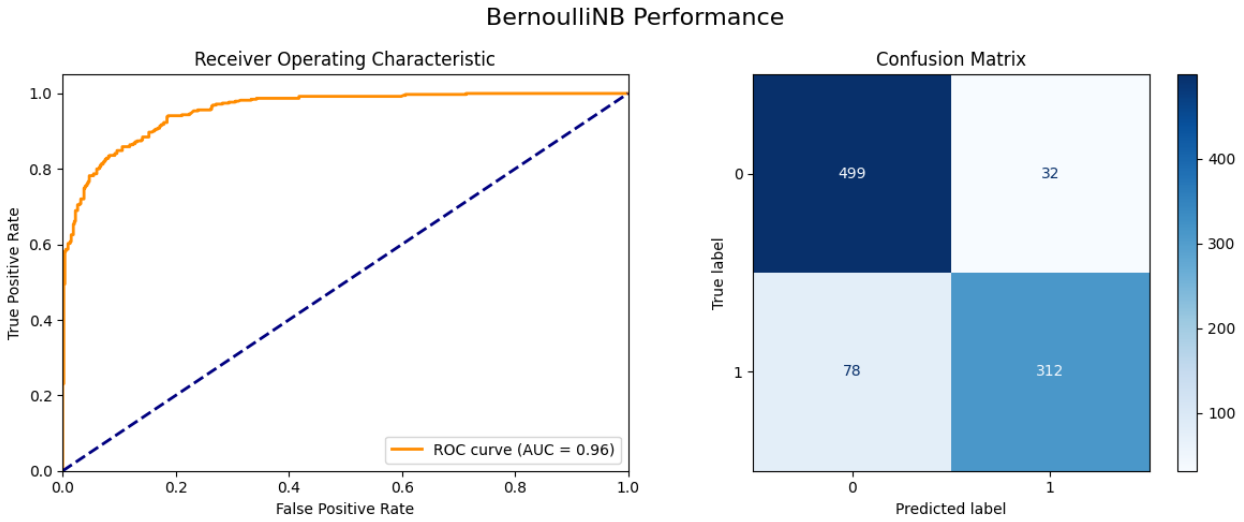

Figure 3: Bernoulli Naïve Bayes Results

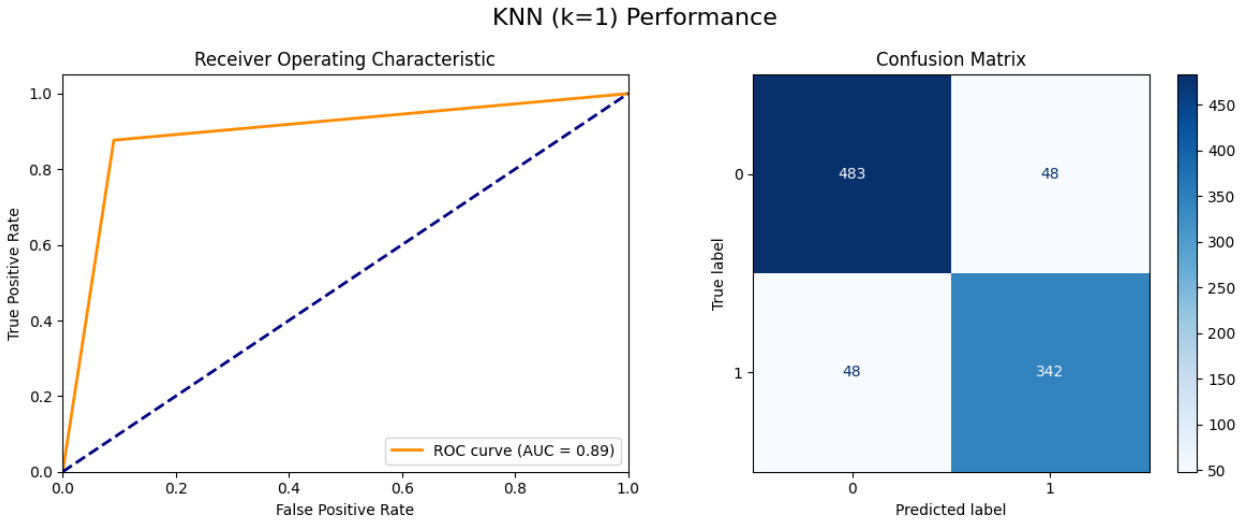

Figure 4: K-Nearest Neighbors (k=1) Results

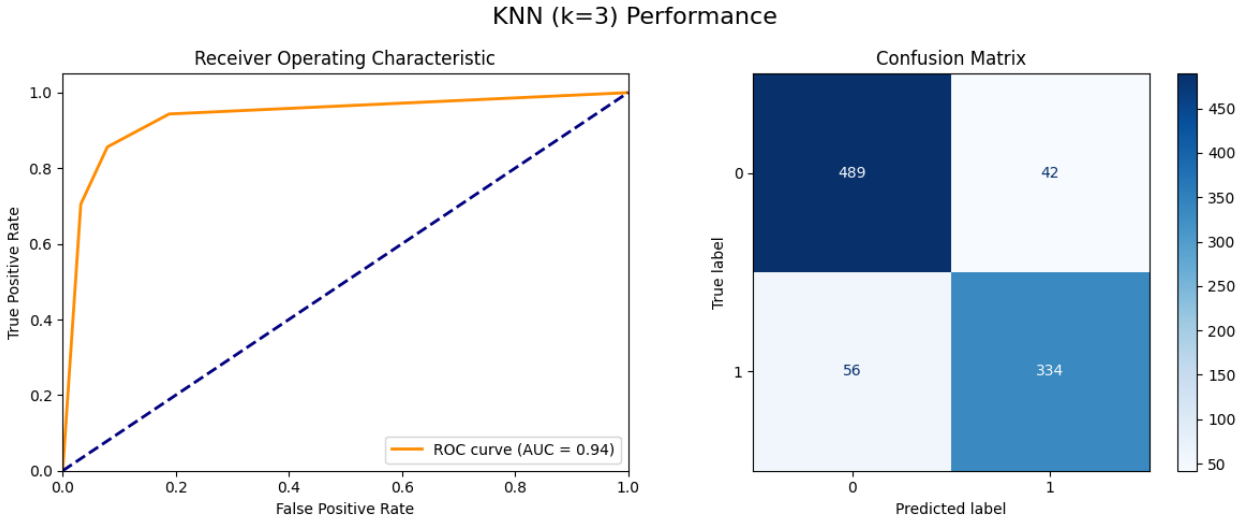

Figure 5: K-Nearest Neighbors (k=3) Results

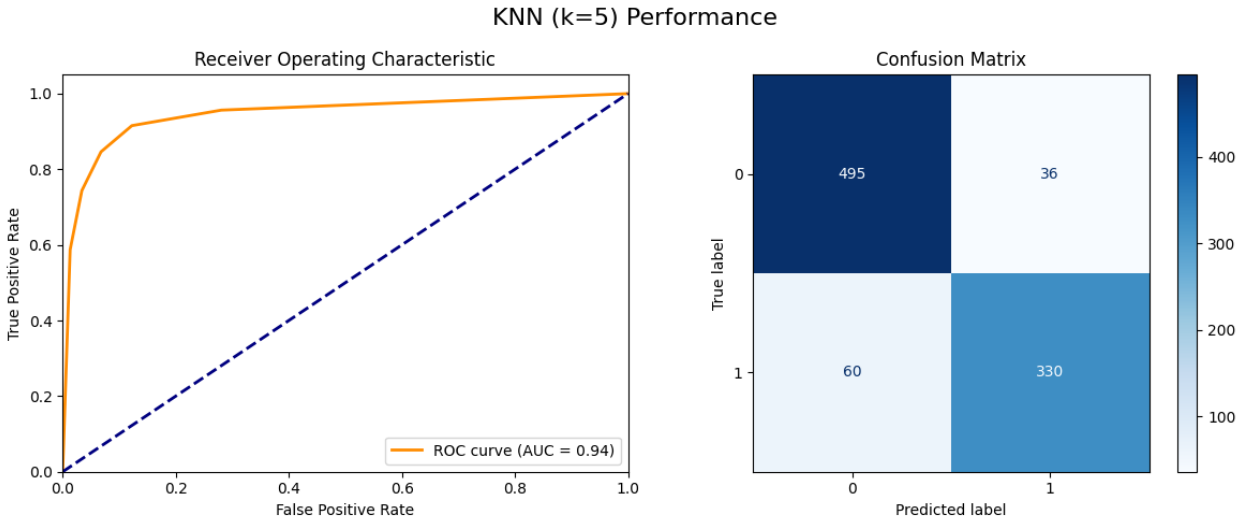

Figure 6: K-Nearest Neighbors (k=5) Results

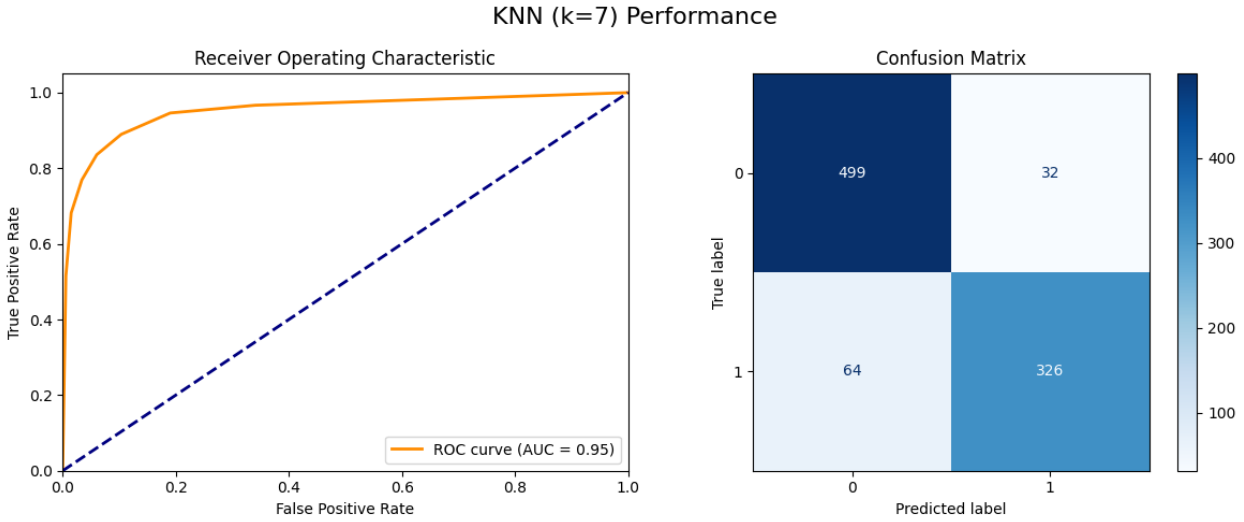

Figure 7: K-Nearest Neighbors (k=7) Results

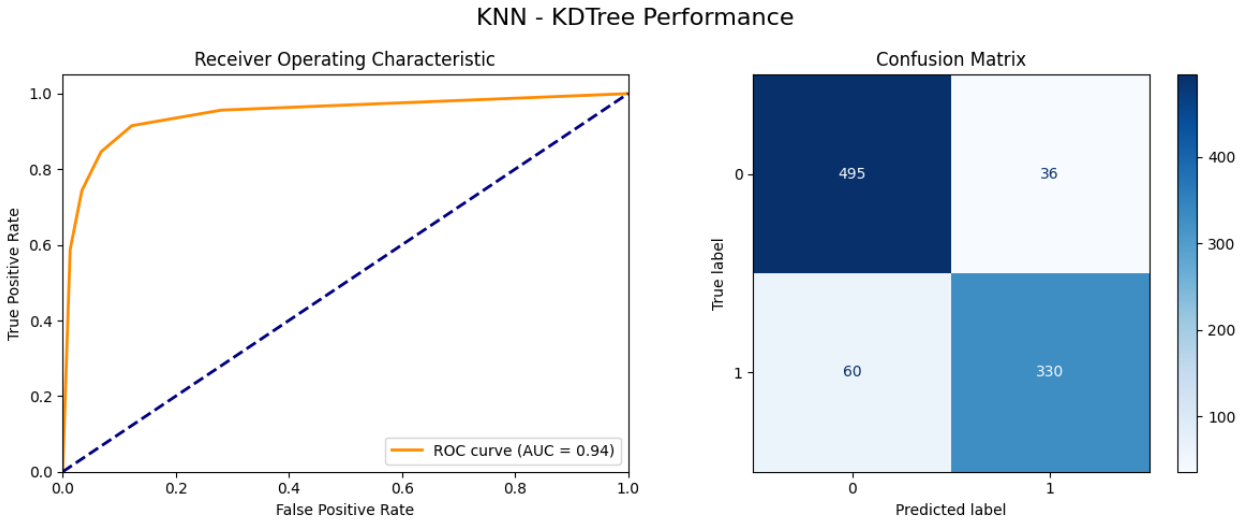

Figure 8: K-Nearest Neighbors (KDTree) Results

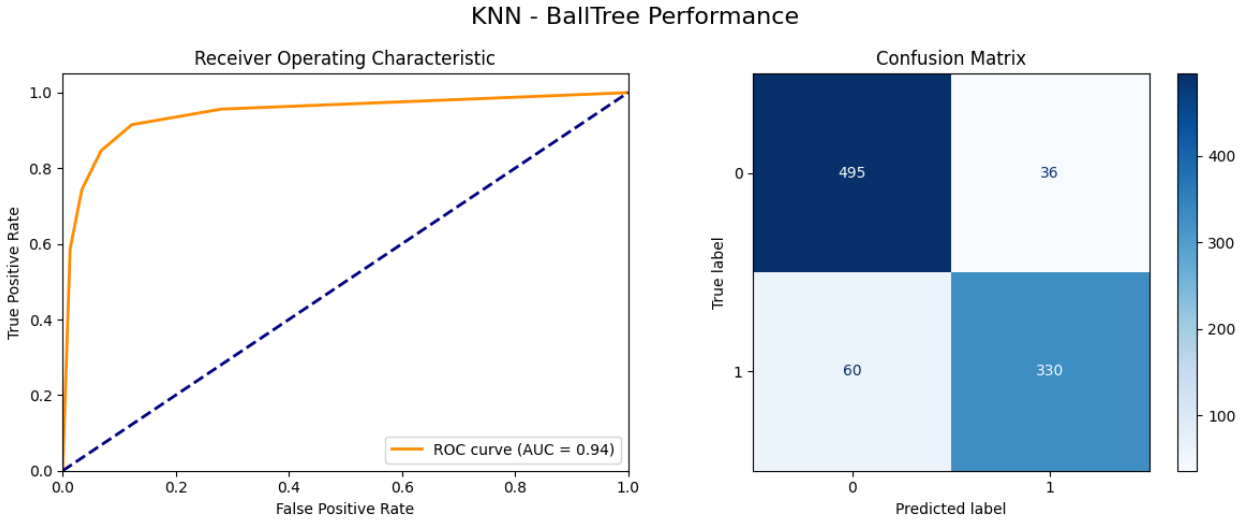

Figure 9: K-Nearest Neighbors (BallTree) Results

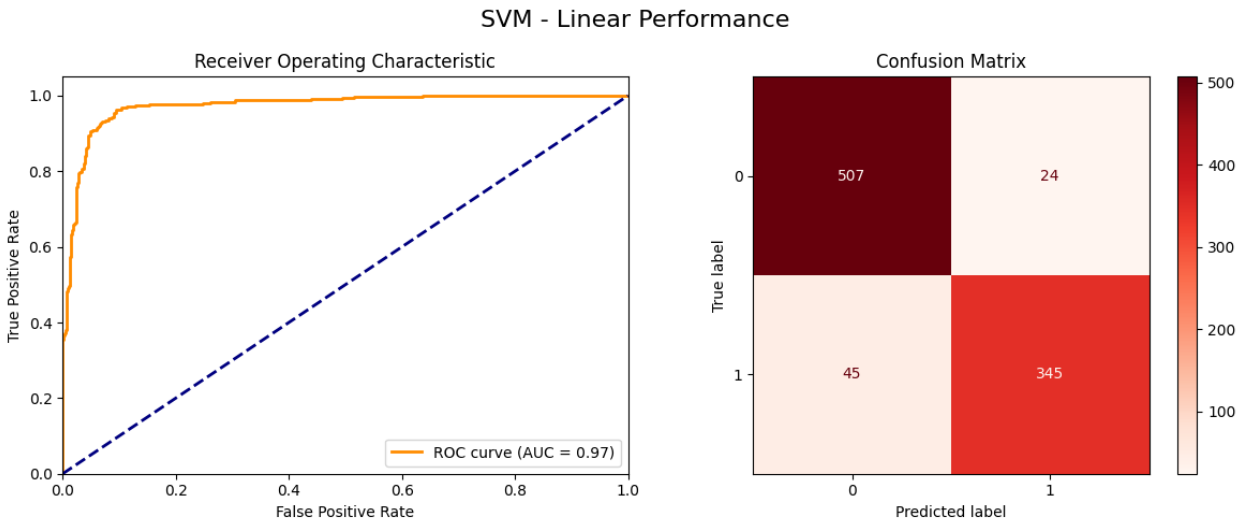

Figure 10: SVM (Linear Kernel) Results

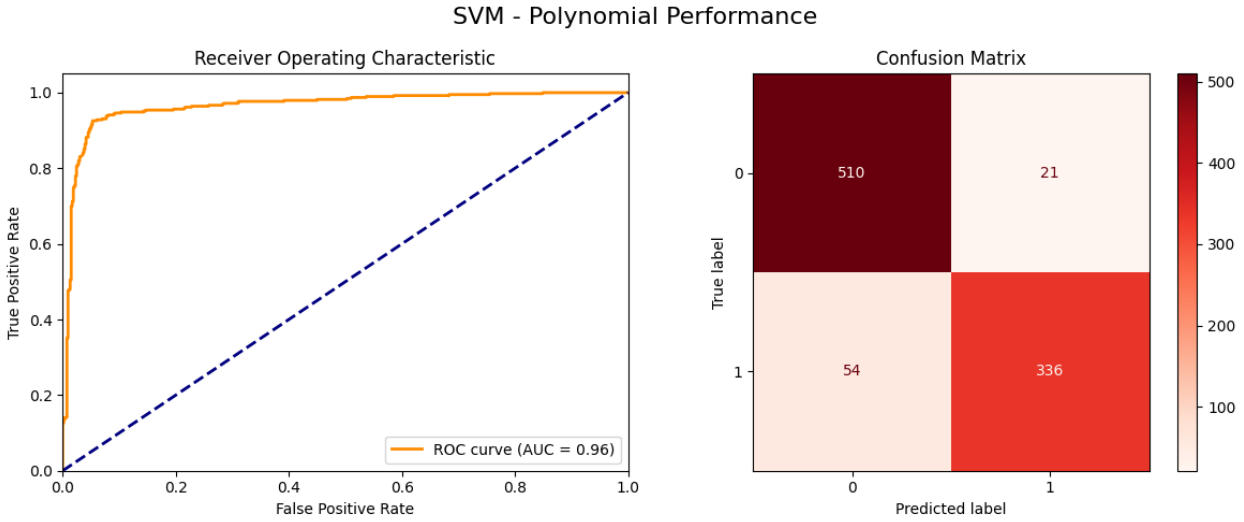

Figure 11: SVM (Polynomial Kernel) Results

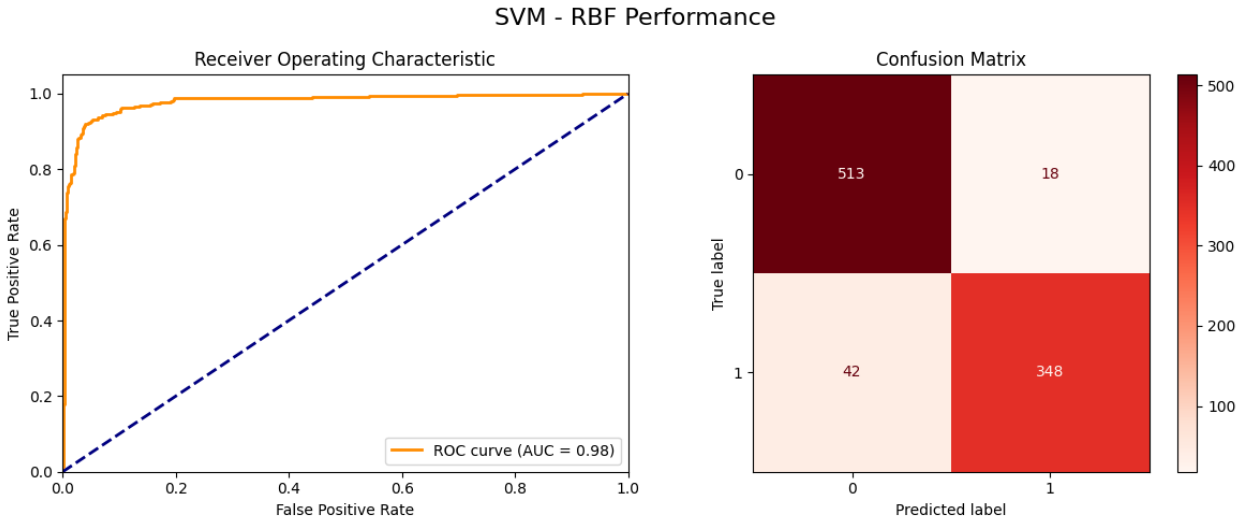

Figure 12: SVM (RBF Kernel) Results

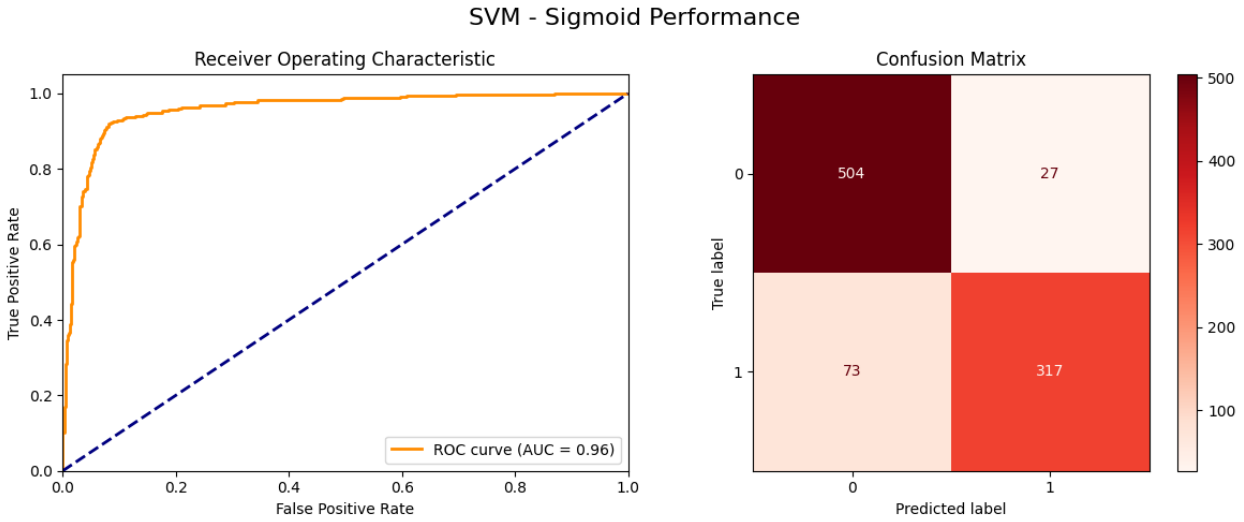

Figure 13: SVM (Sigmoid Kernel) Results

Conclusion

In this experiment, we implemented and evaluated Naïve Bayes, KNN, and SVM classifiers on the Spambase dataset.

- **Naïve Bayes:** Fastest, but GaussianNB and BernoulliNB performed better than

MultinomialNB.

- **KNN:** Best result at $k = 7$. KDTree and BallTree yielded identical accuracy and F1-scores.
- **SVM:** RBF kernel achieved the highest performance overall with 93.5% accuracy and F1-score of 0.921.
- **Cross-validation:** SVM (RBF) consistently outperformed others across folds.

Thus, SVM with RBF kernel is the most suitable model for spam classification in this study.