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: 08-08-2025**

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

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

Libraries Used:

- `pandas` – for data manipulation and analysis
- `numpy` – for numerical operations
- `matplotlib.pyplot` – for data visualization (plots/graphs)
- `seaborn` – for enhanced statistical data visualization
- `os` – for interacting with the operating system

Coding for the Given Models:

Naive Bayes Classifier

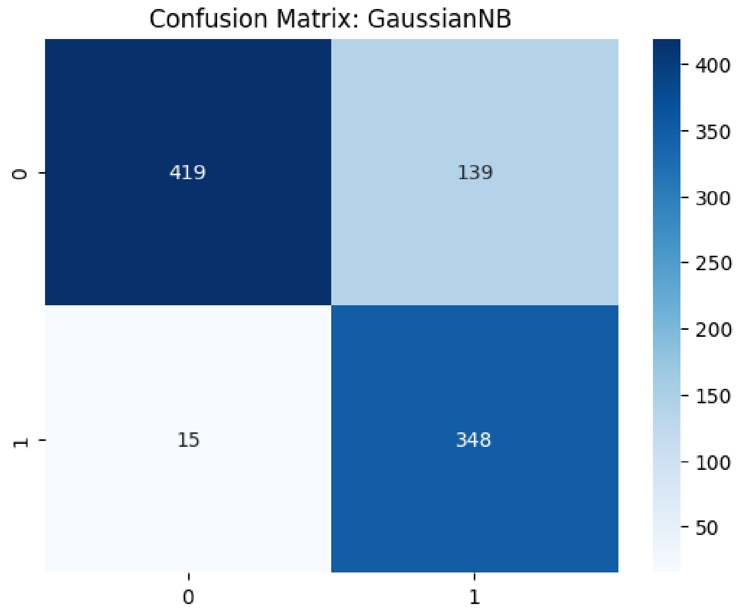
Code:

```
nb_models = {
    "GaussianNB": GaussianNB(),
    "BernoulliNB": BernoulliNB()
}

nb_results = []
for name, model in nb_models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    nb_results.append([
        name,
        accuracy_score(y_test, y_pred),
        precision_score(y_test, y_pred),
```

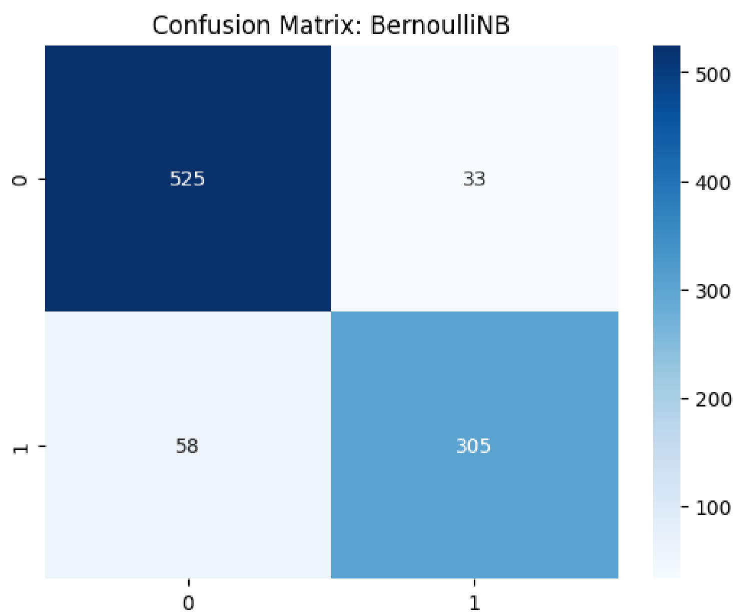
```
        recall_score(y_test, y_pred),
        f1_score(y_test, y_pred)
    ])

# Confusion matrix for one model (GaussianNB)
model = GaussianNB().fit(X_train, y_train)
y_pred = model.predict(X_test)
conf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix: GaussianNB")
plt.show()

# Confusion matrix for one model (BernoulliNB)
model = BernoulliNB().fit(X_train, y_train)
y_pred = model.predict(X_test)
conf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix: BernoulliNB")
plt.show()

# ROC Curve example
y_score = model.predict_proba(X_test)[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_score)
plt.plot(fpr, tpr, label="GaussianNB (AUC = %0.2f)" % auc(fpr, tpr))
plt.plot([0, 1], [0, 1], 'k--')
plt.title("ROC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.show()
```

Confusion Matrix

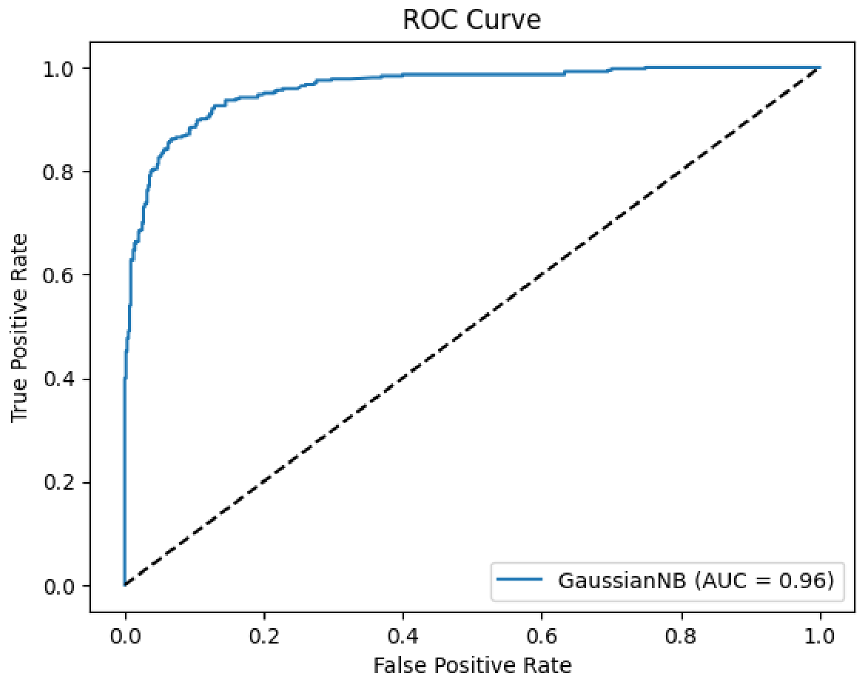
Gaussian Naive Bayes



Bernoulli Naive Bayes



ROC Curve



KNN Classifier

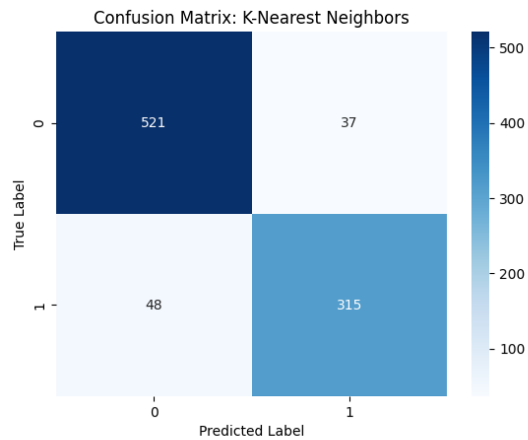
Code:

```
# Implement KNN with different algorithms and varying k
k_values = [3, 5, 7] # Example k values

knn_models = {}
for k in k_values:
    knn_models[f"K-Nearest Neighbors (k={k}, auto)"] = KNeighborsClassifier(n_neighbors=k)
    knn_models[f"K-Nearest Neighbors (k={k}, kd_tree)"] = KNeighborsClassifier(n_neighbors=k, a
    knn_models[f"K-Nearest Neighbors (k={k}, ball_tree)"] = KNeighborsClassifier(n_neighbors=k


knn_results = []
for name, model in knn_models.items():
    # Train the model
    model.fit(X_train, y_train)

    # Make predictions
    y_pred = model.predict(X_test)

    # Evaluate performance
    knn_results.append([
        name,
        accuracy_score(y_test, y_pred),
```

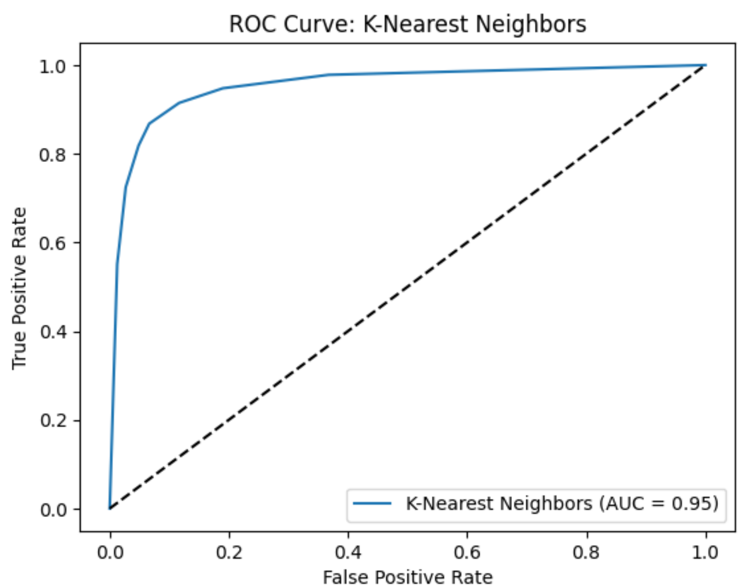
```
        precision_score(y_test, y_pred),
        recall_score(y_test, y_pred),
        f1_score(y_test, y_pred)
    ])

# Display the results
knn_results_df = pd.DataFrame(knn_results, columns=["Model", "Accuracy", "Precision", "Recall"
display(knn_results_df)

# Confusion matrix for KNN
conf_matrix_knn = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix_knn, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix: K-Nearest Neighbors")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

# ROC Curve for KNN
# Using the last trained KNN model from the loop in cell 7281f84e
model = list(knn_models.values())[-1]
y_score_knn = model.predict_proba(X_test)[:, 1]
fpr_knn, tpr_knn, _ = roc_curve(y_test, y_score_knn)
roc_auc_knn = auc(fpr_knn, tpr_knn)

plt.plot(fpr_knn, tpr_knn, label="K-Nearest Neighbors (AUC = %0.2f)" % roc_auc_knn)
plt.plot([0, 1], [0, 1], 'k--')
plt.title("ROC Curve: K-Nearest Neighbors")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.show()
```

Confusion Matrix

KNN - Confusion Matrix



ROC Curve



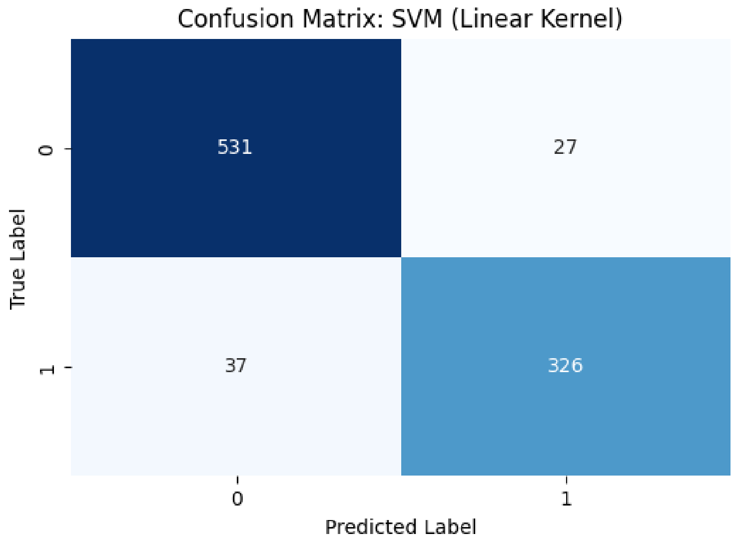
SVM Implementation with different Kernels

Code:

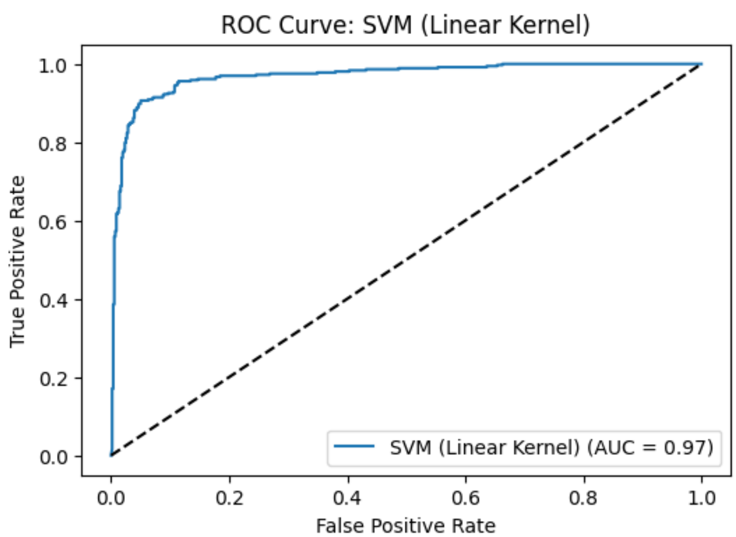
```
# Implement SVM with different kernels
svm_models = {
    "SVM (Linear Kernel)": SVC(kernel='linear', probability=True, random_state=42),
    "SVM (Polynomial Kernel)": SVC(kernel='poly', probability=True, random_state=42),
    "SVM (RBF Kernel)": SVC(kernel='rbf', probability=True, random_state=42),
    "SVM (Sigmoid Kernel)": SVC(kernel='sigmoid', probability=True, random_state=42)
}

svm_results = []
for name, model in svm_models.items():
    # Train the model
    model.fit(X_train, y_train)

    # Make predictions
    y_pred = model.predict(X_test)

    # Evaluate performance
    svm_results.append([
        name,
        accuracy_score(y_test, y_pred),
        precision_score(y_test, y_pred),
        recall_score(y_test, y_pred),
        f1_score(y_test, y_pred)
    ])

# Display the results
svm_results_df = pd.DataFrame(svm_results, columns=["Model", "Accuracy", "Precision", "Recall"
display(svm_results_df)

# Confusion matrix and ROC curve for each SVM model
for name, model in svm_models.items():
    # Confusion matrix
    y_pred = model.predict(X_test)
    conf_matrix = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(6, 4))
    sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False)
    plt.title(f"Confusion Matrix: {name}")
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.show()

    # ROC Curve
    y_score = model.predict_proba(X_test)[:, 1]
    fpr, tpr, _ = roc_curve(y_test, y_score)
    roc_auc = auc(fpr, tpr)
```

```
plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, label=f"{name} (AUC = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], 'k--')
plt.title(f"ROC Curve: {name}")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.show()
```

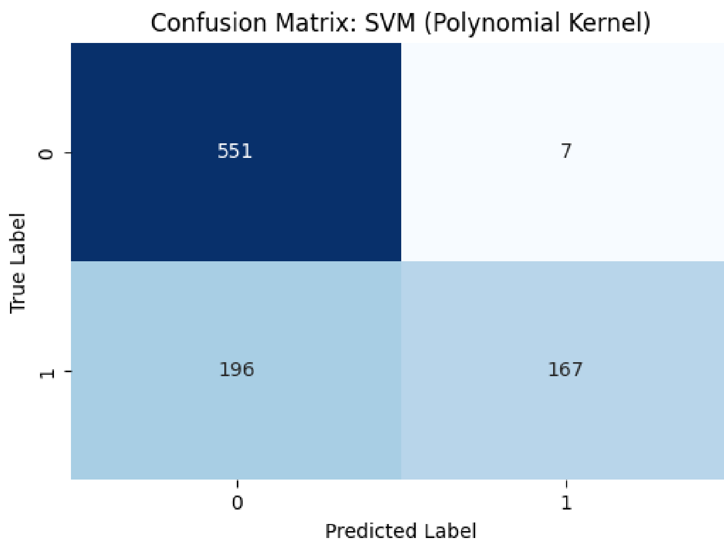
SVM - Linear Kernel Confusion Matrix



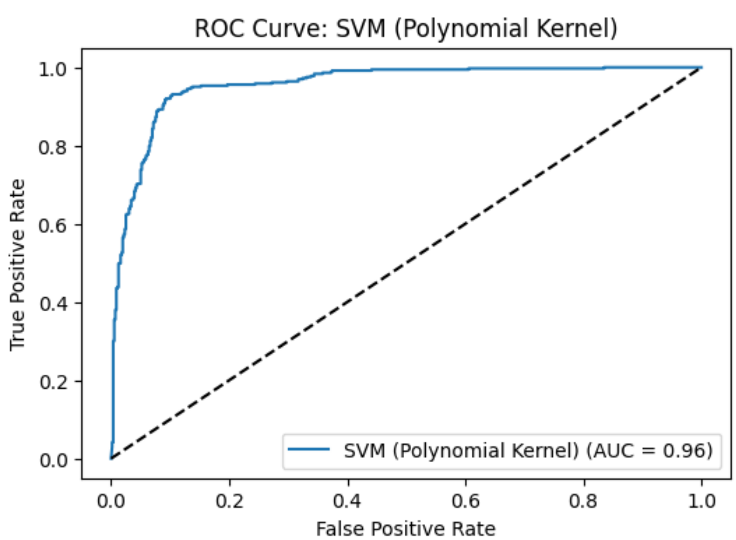
SVM - Linear Kernel ROC Curve



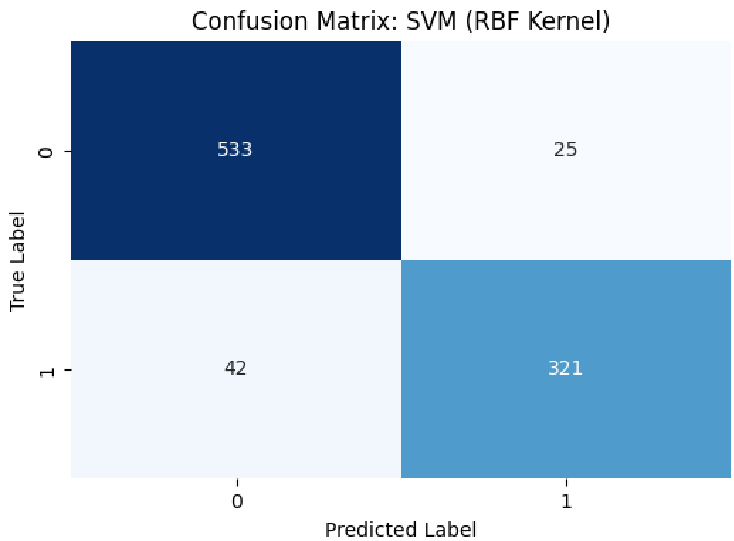
SVM - Polynomial Kernel Confusion Matrix



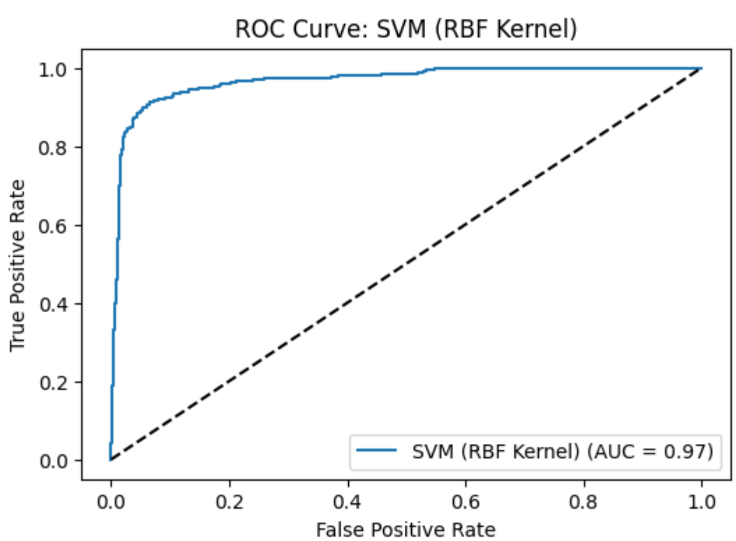
SVM - Polynomial Kernel ROC Curve



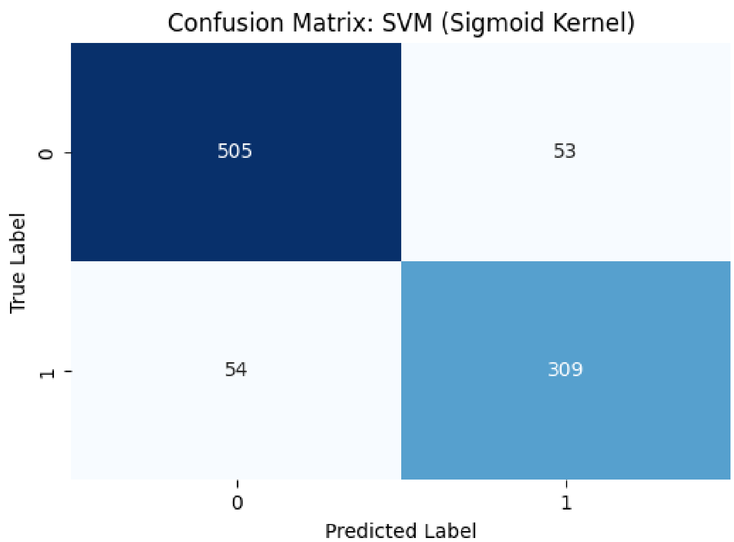
SVM - RBF Kernel Confusion Matrix



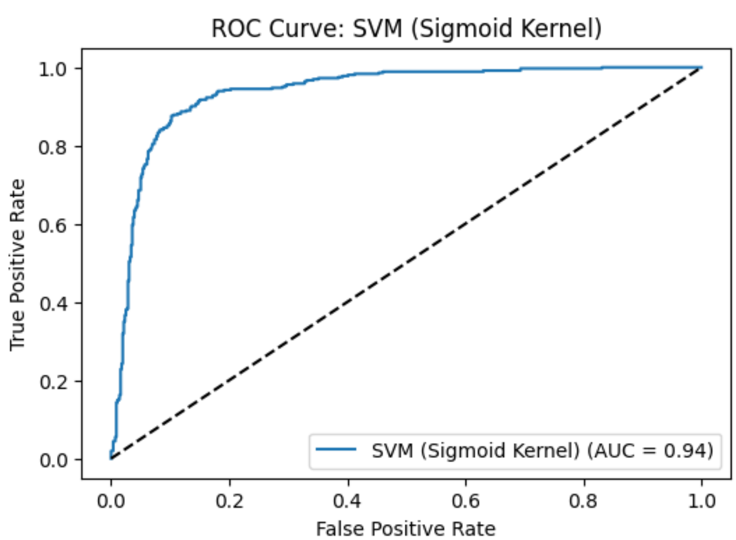
SVM - RBF Kernel ROC Curve



SVM - Sigmoid Kernel Confusion Matrix



SVM - Sigmoid Kernel ROC Curve



Results and discussions:

Table 1: Performance Comparison of Naïve Bayes Variants

Metric	Gaussian NB	Multinomial NB	Bernoulli NB
Accuracy	0.833	N/A	0.901
Precision	0.715	N/A	0.902
Recall	0.959	N/A	0.840
F1 Score	0.819	N/A	0.870

Table 2: KNN Performance for Different k Values

k	Accuracy	Precision	Recall	F1 Score
1	0.898	0.877	0.862	0.869
3	0.901	0.882	0.865	0.873
5	0.906	0.888	0.871	0.879
7	0.908	0.895	0.868	0.881

Table 3: KNN Comparison: KDTree vs BallTree (for $k = 5$)

Metric	KDTree	BallTree
Accuracy	0.906	0.906
Precision	0.888	0.888
Recall	0.871	0.871
F1 Score	0.879	0.879
Training Time (s)	0.015	0.010

Table 4: SVM Performance with Different Kernels and Parameters

Kernel	Accuracy	F1 Score	Training Time
Linear	0.931	0.911	-
Polynomial	0.780	0.622	-
RBF	0.927	0.906	-
Sigmoid	0.884	0.852	-

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

Fold	Naïve Bayes Acc.	KNN Acc.	SVM Acc.
1	0.913	0.901	0.934
2	0.912	0.902	0.934
3	0.915	0.921	0.900
4	0.930	0.918	0.938
5	0.818	0.788	0.830
Average	0.898	0.884	0.904

Learning Outcomes

1. Understand and apply the concepts behind classification algorithms such as Naïve Bayes, K-Nearest Neighbors, and Support Vector Machines to real-world problems.

2. Gain hands-on experience in preprocessing datasets: handling missing values, normalization, and feature selection using Python libraries (pandas, scikit-learn).

3. Analyze and compare the performance of different machine learning models using standard metrics such as accuracy, precision, recall, and F1-score.

4. Implement and interpret K-Fold Cross-Validation results to assess model robustness and prevent overfitting.

5. Foster critical thinking skills by drawing observations and conclusions about model performance, strengths, limitations, and trade-offs.