Experiment 3: Email Spam or Ham Classification

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:**

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

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

Theoretical Description

1. Naive Bayes Classifier

Naive Bayes is a **probabilistic classifier** based on **Bayes' Theorem**, assuming **feature independence**.

$$P(C_k \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid C_k) \cdot P(C_k)}{P(\mathbf{x})}$$

$$P(\mathbf{x} \mid C_k) = \prod_{i=1}^{n} P(x_i \mid C_k) \quad \Rightarrow \quad P(C_k \mid \mathbf{x}) \propto P(C_k) \prod_{i=1}^{n} P(x_i \mid C_k)$$

Types:

- Gaussian NB (continuous)
- Multinomial NB (discrete counts)
- Bernoulli NB (binary features)

2. K-Nearest Neighbors (KNN)

KNN is a non-parametric, instance-based algorithm. It classifies data by majority vote among the k nearest neighbors using a distance metric like Euclidean distance.

$$d(\mathbf{x}, \mathbf{x}^{(i)}) = \sqrt{\sum_{j=1}^{n} (x_j - x_j^{(i)})^2}$$

$$\hat{C} = \arg\max_{C_k} \sum_{\mathbf{x}^{(i)} \in \mathcal{N}_k(\mathbf{x})} \mathbb{I}(y^{(i)} = C_k)$$

Python Code (Bernoulli Naive Bayes)

```
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,cross_val_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_

from google.colab import drive
drive.mount('/content/drive')

df = pd.read_csv("/content/spambase_csv.csv")

df.head()

for col in df.select_dtypes(include='object'):
    df[col].fillna(df[col].mode()[0], inplace=True)

for col in df.select_dtypes(include=['float64', 'int64']):
    df[col].fillna(df[col].mean(), inplace=True)

label_encoder = LabelEncoder()
binary_cols = [col for col in df.select_dtypes(include='object') if df[col].nunique() == 2]
for col in binary_cols:
    df[col] = label_encoder.fit_transform(df[col])

multi_cols = [col for col in df.select_dtypes(include='object') if col not in binary_cols]
```

```
df = pd.get_dummies(df, columns=multi_cols, drop_first=True)

target_col = 'class'  # CHANGE THIS
X = df.drop(target_col, axis=1)
y = df[target_col]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_temp, y_train, y_temp = train_test_split(X_scaled, y, test_size=0.3, random_state=42
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

from sklearn.naive_bayes import BernoulliNB
model = BernoulliNB()
model.fit(X_train, y_train)

def evaluate_classifier(model, X_data, y_true, name="Set"):
    y_pred = model.predict(X_data)
    print(f"--- {name} ---")
    print("Accuracy :", accuracy_score(y_true, y_pred))
    print("Precision:", precision_score(y_true, y_pred, average='weighted'))
    print("Recall   :", recall_score(y_true, y_pred, average='weighted'))
    print("F1 Score :", f1_score(y_true, y_pred, average='weighted'))
    print("\nConfusion Matrix:\n", confusion_matrix(y_true, y_pred))
    print("\nClassification Report:\n", classification_report(y_true, y_pred))

evaluate_classifier(model, X_val, y_val, "Validation")
evaluate_classifier(model, X_test, y_test, "Test")

scores = cross_val_score(model, X, y, cv=5, scoring='accuracy')

# Print results
print("Cross-validation scores:", scores)
print("Mean accuracy:", scores.mean())

from sklearn.metrics import ConfusionMatrixDisplay
ConfusionMatrixDisplay.from_estimator(model, X_test, y_test, cmap='Blues')
plt.title("Confusion Matrix - Test Set")
plt.show()

from sklearn.metrics import roc_curve, auc

# Predict probabilities
y_pred_proba = model.predict_proba(X_test)[:, 1]

# Compute ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)
```

```
# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```

Python Code (Multinomial Naive Bayes)

```
    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,cross_val_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_

from google.colab import drive
#drive.mount('/content/drive')

df = pd.read_csv("/content/spambase_csv.csv")

df.head()

for col in df.select_dtypes(include='object'):
    df[col].fillna(df[col].mode()[0], inplace=True)

for col in df.select_dtypes(include=['float64', 'int64']):
    df[col].fillna(df[col].mean(), inplace=True)

label_encoder = LabelEncoder()
binary_cols = [col for col in df.select_dtypes(include='object') if df[col].nunique() == 2]
for col in binary_cols:
    df[col] = label_encoder.fit_transform(df[col])

multi_cols = [col for col in df.select_dtypes(include='object') if col not in binary_cols]
df = pd.get_dummies(df, columns=multi_cols, drop_first=True)

target_col = "class"  # CHANGE THIS
X = df.drop(target_col, axis=1)
y = df[target_col]
```

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_temp, y_train, y_temp = train_test_split(X_scaled, y, test_size=0.3, random_state=42
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

from sklearn.naive_bayes import MultinomialNB
model = MultinomialNB()
model.fit(X_train, y_train)

def evaluate_classifier(model, X_data, y_true, name="Set"):
    y_pred = model.predict(X_data)
    print(f"--- {name} ---")
    print("Accuracy :", accuracy_score(y_true, y_pred))
    print("Precision:", precision_score(y_true, y_pred, average='weighted'))
    print("Recall   :", recall_score(y_true, y_pred, average='weighted'))
    print("F1 Score :", f1_score(y_true, y_pred, average='weighted'))
    print("\nConfusion Matrix:\n", confusion_matrix(y_true, y_pred))
    print("\nClassification Report:\n", classification_report(y_true, y_pred))

evaluate_classifier(model, X_val, y_val, "Validation")
evaluate_classifier(model, X_test, y_test, "Test")
evaluate_classifier(model, X_train, y_train, "Train")

scores = cross_val_score(model, X, y, cv=5, scoring='accuracy')

# Print results
print("Cross-validation scores:", scores)
print("Mean accuracy:", scores.mean())

from sklearn.metrics import ConfusionMatrixDisplay
ConfusionMatrixDisplay.from_estimator(model, X_test, y_test, cmap='Blues')
plt.title("Confusion Matrix - Test Set")
plt.show()

from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Get predicted probabilities for the test set
y_pred_proba = model.predict_proba(X_test)[:, 1]

# Compute ROC curve and AUC
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
```

```
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

Python code Gaussian Naive Bayes

```
    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,cross_val_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_

df = pd.read_csv("/content/spambase_csv.csv")

for col in df.select_dtypes(include='object'):
    df[col].fillna(df[col].mode()[0], inplace=True)

for col in df.select_dtypes(include=['float64', 'int64']):
    df[col].fillna(df[col].mean(), inplace=True)

label_encoder = LabelEncoder()
binary_cols = [col for col in df.select_dtypes(include='object') if df[col].nunique() == 2]
for col in binary_cols:
    df[col] = label_encoder.fit_transform(df[col])

multi_cols = [col for col in df.select_dtypes(include='object') if col not in binary_cols]
df = pd.get_dummies(df, columns=multi_cols, drop_first=True)

target_col = 'class'  # CHANGE THIS
X = df.drop(target_col, axis=1)
y = df[target_col]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_temp, y_train, y_temp = train_test_split(X_scaled, y, test_size=0.3, random_state=42
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
```

```
from sklearn.Naive_Bayes import GaussianNb
model = GaussianNb()
model.fit(X_train, y_train)

def evaluate_classifier(model, X_data, y_true, name="Set"):
    y_pred = model.predict(X_data)
    print(f"--- {name} ---")
    print("Accuracy :", accuracy_score(y_true, y_pred))
    print("Precision:", precision_score(y_true, y_pred, average='weighted'))
    print("Recall   :", recall_score(y_true, y_pred, average='weighted'))
    print("F1 Score :", f1_score(y_true, y_pred, average='weighted'))
    print("\nConfusion Matrix:\n", confusion_matrix(y_true, y_pred))
    print("\nClassification Report:\n", classification_report(y_true, y_pred))

evaluate_classifier(model, X_val, y_val, "Validation")
evaluate_classifier(model, X_test, y_test, "Test")

# Apply 5-fold cross-validation
scores = cross_val_score(model, X, y, cv=5, scoring='accuracy')

# Print results
print("Cross-validation scores:", scores)
print("Mean accuracy:", scores.mean())

from sklearn.metrics import ConfusionMatrixDisplay
ConfusionMatrixDisplay.from_estimator(model, X_test, y_test, cmap='Blues')
plt.title("Confusion Matrix - Test Set")
plt.show()

from sklearn.metrics import RocCurveDisplay

RocCurveDisplay.from_estimator(model, X_test, y_test)
plt.title("ROC Curve - Test Set")
plt.show()
```

Python Code KNN

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

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_
```

```
from sklearn.neighbors import KNeighborsClassifier

# Read dataset
df = pd.read_csv('/content/spambase_csv.csv')

# Fill missing values
for col in df.select_dtypes(include='object'):
    df[col].fillna(df[col].mode()[0], inplace=True)

for col in df.select_dtypes(include=['float64', 'int64']):
    df[col].fillna(df[col].mean(), inplace=True)

# Encode binary columns
label_encoder = LabelEncoder()
binary_cols = [col for col in df.select_dtypes(include='object') if df[col].nunique() == 2]
for col in binary_cols:
    df[col] = label_encoder.fit_transform(df[col])

# One-hot encode remaining categorical columns
multi_cols = [col for col in df.select_dtypes(include='object') if col not in binary_cols]
df = pd.get_dummies(df, columns=multi_cols, drop_first=True)

# Split features and target
target_col = 'class'
X = df.drop(target_col, axis=1)
y = df[target_col]

# Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-Validation-Test split
X_train, X_temp, y_train, y_temp = train_test_split(X_scaled, y, test_size=0.3, random_state=42
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# Evaluation function
def evaluate_classifier(model, X_data, y_true, name="Set"):
    y_pred = model.predict(X_data)
    print(f"--- {name} ---")
    print("Accuracy :", accuracy_score(y_true, y_pred))
    print("Precision:", precision_score(y_true, y_pred, average='weighted'))
    print("Recall   :", recall_score(y_true, y_pred, average='weighted'))
    print("F1 Score :", f1_score(y_true, y_pred, average='weighted'))
    print("\nConfusion Matrix:\n", confusion_matrix(y_true, y_pred))

# Set k values
k_values = [1, 3, 5, 7, 9, 11]
```

```
# ------------------------------
# Algorithm = 'auto'
# ------------------------------
print(f"Training KNeighborsClassifier with algorithm = 'auto'")
for k in k_values:
    print(f"\nTraining with k = {k}")
    model = KNeighborsClassifier(n_neighbors=k, algorithm='auto', metric='manhattan')
    model.fit(X_train, y_train)
    evaluate_classifier(model, X_val, y_val, f"Validation (k={k}, algorithm='auto')")

# Confusion matrix on test set
ConfusionMatrixDisplay.from_estimator(model, X_test, y_test, cmap='Blues')
plt.title("Confusion Matrix - Test Set")
plt.show()

# ------------------------------
# Cross-Validation: 'ball_tree'
# ------------------------------
print(f"\nCross-validation with algorithm = 'ball_tree'")
for k in k_values:
    print(f"\nTraining with k = {k}")
    model = KNeighborsClassifier(n_neighbors=k, algorithm='ball_tree')

    start_time = time.time()
    scores = cross_val_score(model, X_scaled, y, cv=5, scoring='accuracy')
    end_time = time.time()

    print("Cross-validation scores:", scores)
    print("Mean accuracy:", scores.mean())
    print("Training + Cross-validation time: {:.4f} seconds".format(end_time - start_time))

# ------------------------------
# Cross-Validation: 'kd_tree'
# ------------------------------
print(f"\nCross-validation with algorithm = 'kd_tree'")
for k in k_values:
    print(f"\nTraining with k = {k}")
    model = KNeighborsClassifier(n_neighbors=k, algorithm='kd_tree')

    start_time = time.time()
    scores = cross_val_score(model, X_scaled, y, cv=5, scoring='accuracy')
    end_time = time.time()

    print("Cross-validation scores:", scores)
    print("Mean accuracy:", scores.mean())
    print("Training + Cross-validation time: {:.4f} seconds".format(end_time - start_time))
```

```
    param_grid = {
    'n_neighbors': [1, 3, 5, 7, 9, 11],
    'algorithm': ['auto', 'ball_tree', 'kd_tree']
}

knn = KNeighborsClassifier()
grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy')



grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

print(" Best Hyperparameters:", best_params)
print(" Best Model:", best_model)



def evaluate_classifier(model, X_data, y_true, name="Set"):
    y_pred = model.predict(X_data)
    print(f"--- {name} ---")
    print("Accuracy :", accuracy_score(y_true, y_pred))
    print("Precision:", precision_score(y_true, y_pred, average='weighted'))
    print("Recall   :", recall_score(y_true, y_pred, average='weighted'))
    print("F1 Score :", f1_score(y_true, y_pred, average='weighted'))
    print("\nConfusion Matrix:\n", confusion_matrix(y_true, y_pred))
    ConfusionMatrixDisplay.from_estimator(model, X_data, y_true, cmap='Blues')
    plt.title(f"Confusion Matrix - {name}")
    plt.show()


evaluate_classifier(best_model, X_test, y_test, name="Test Set (Best Model)")
```

Python Code (SVM)

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import time
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_
```

```
from sklearn.svm import SVC
\begin{figure}
    \centering
    \includegraphics[width=0.5\linewidth]{svmlinear.png}
    \caption{Enter Caption}
    \label{fig:placeholder}
\end{figure}
# Read dataset
df = pd.read_csv('/content/spambase_csv.csv')

# Handle missing values
for col in df.select_dtypes(include='object'):
    df[col].fillna(df[col].mode()[0], inplace=True)
for col in df.select_dtypes(include=['float64', 'int64']):
    df[col].fillna(df[col].mean(), inplace=True)

# Encode categorical columns
label_encoder = LabelEncoder()
binary_cols = [col for col in df.select_dtypes(include='object') if df[col].nunique() == 2]
for col in binary_cols:
    df[col] = label_encoder.fit_transform(df[col])
multi_cols = [col for col in df.select_dtypes(include='object') if col not in binary_cols]
df = pd.get_dummies(df, columns=multi_cols, drop_first=True)

# Features and target
target_col = 'class'
X = df.drop(target_col, axis=1)
y = df[target_col]

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

# Split
X_train, X_temp, y_train, y_temp = train_test_split(X_scaled, y, test_size=0.3, random_state=42
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# Kernels to evaluate
kernels = {
    "Linear": {"kernel": "linear", "C": 1},
    "Polynomial": {"kernel": "poly", "degree": 3, "C": 1, "gamma": "scale"},
    "RBF": {"kernel": "rbf", "C": 1, "gamma": "scale"},
    "Sigmoid": {"kernel": "sigmoid", "C": 1, "gamma": "scale"}
}

# Evaluation function
def evaluate_classifier(model, X_data, y_true, name="Set"):
    y_pred = model.predict(X_data)
```

```
    print(f"--- {name} ---")
    print("Accuracy :", accuracy_score(y_true, y_pred))
    print("Precision:", precision_score(y_true, y_pred, average='weighted'))
    print("Recall   :", recall_score(y_true, y_pred, average='weighted'))
    print("F1 Score :", f1_score(y_true, y_pred, average='weighted'))
    print("\nConfusion Matrix:\n", confusion_matrix(y_true, y_pred))

# Train and evaluate SVM models
for name, params in kernels.items():
    print(f"\n----- {name} Kernel -----")
    model = SVC(**params, probability=True, random_state=42)

    start = time.time()
    model.fit(X_train, y_train)
    end = time.time()

    evaluate_classifier(model, X_val, y_val, f"Validation ({name})")
    evaluate_classifier(model, X_test, y_test, f"Test ({name})")

    # Cross-validation
    scores = cross_val_score(model, X_scaled, y, cv=5, scoring='accuracy')
    print("Cross-validation scores:", scores)
    print("Mean accuracy:", scores.mean())
    print(f"Training Time: {end - start:.2f} seconds")

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

    plt.figure()
    plt.plot(fpr, tpr, lw=2, label=f'{name} (area = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve - {name} Kernel')
    plt.legend(loc="lower right")
    plt.show()
```

Table 1: SVM Performance with Different Kernels and Parameters

Kernel	Hyperparameters	Accuracy	F1 Score
Linear	C=1	0.9054	0.9021
Polynomial	C=1, degree=3, gamma=scale	0.8982	0.8955
RBF	C=1, gamma=scale	0.9126	0.9102
Sigmoid	C=1, gamma=scale	0.8723	0.8704

Table 2: Performance Comparison of Naïve Bayes Variants

Metric	Gaussian NB	Multinomial NB	Bernoulli NB
Accuracy	0.8899	0.8594	0.8130
Precision	0.8900	0.8746	0.8441
Recall	0.8899	0.8594	0.8130
F1 Score	0.8893	0.8549	0.8136

Table 3: KNN Performance for Different k Values

k	Accuracy	Precision	Recall	F1 Score
1	0.8898	0.8900	0.8898	0.8893
3	0.8942	0.8954	0.8942	0.8934
5	0.8884	0.8912	0.8884	0.8871
7	0.9014	0.9049	0.901	0.8999

SVM Performance with Different Kernels

Performance Comparison

KNN: Varying k Values

KNN: KDTree vs BallTree

Table 4: KNN Comparison: KDTree vs BallTree

Metric	KDTree	BallTree
Accuracy	0.8806	0.8837
Precision	0.8823	0.8853
Recall	0.8806	0.8837
F1 Score	0.8808	0.8836
Training Time (s)	5.6s	5.9s

Plots

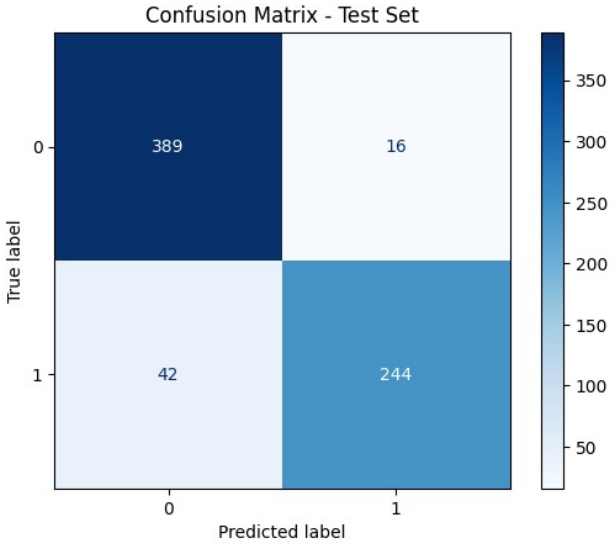

Figure 1: Bernoulli Confusion Matrix

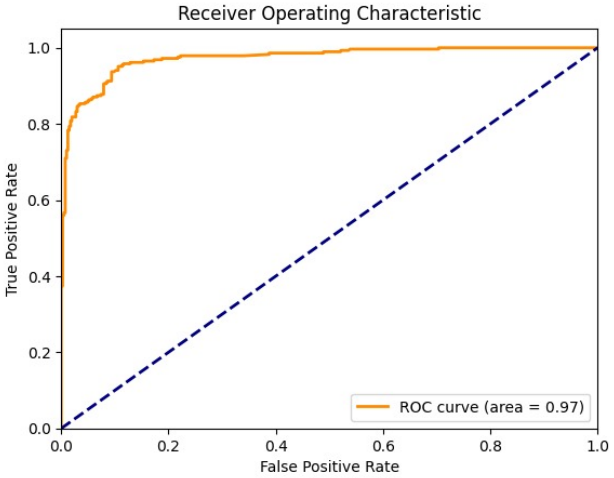

Figure 2: Bernoulli ROC Curve

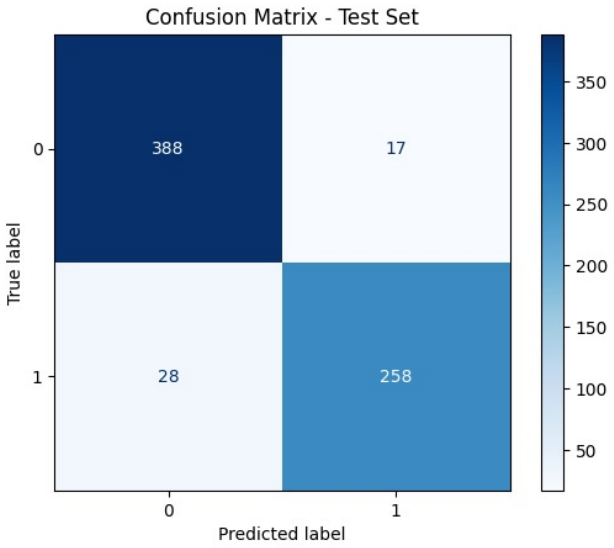

Figure 3: Gaussian Confusion Matrix

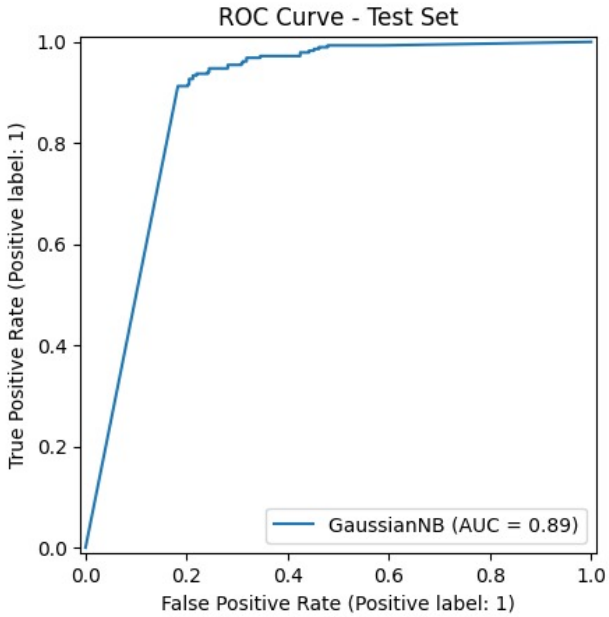

Figure 4: Gaussian ROC curve

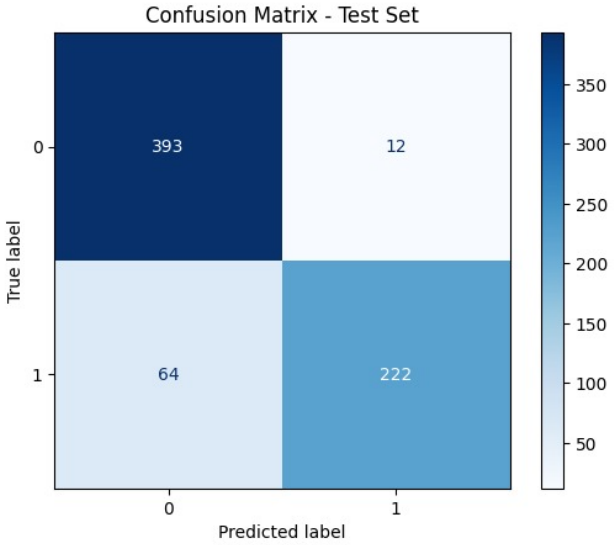

Figure 5: MultiNomial Confusion Matrix

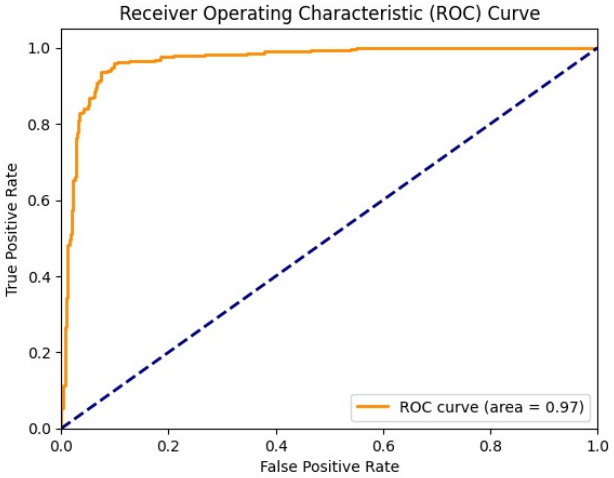

Figure 6: MultiNomial ROC Curve

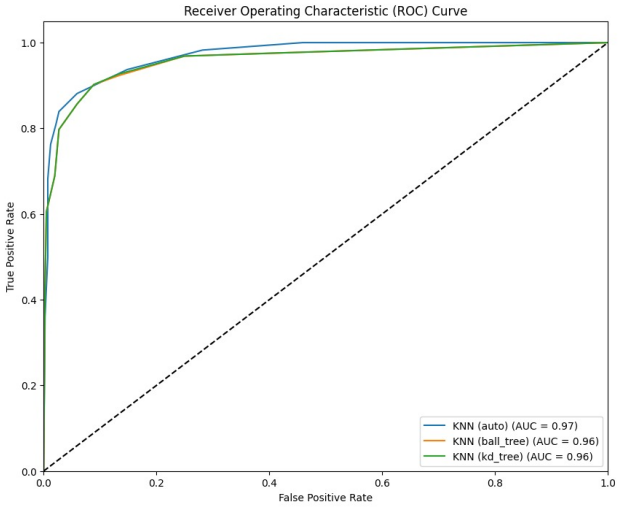

Figure 7: KNN ROC CURVES

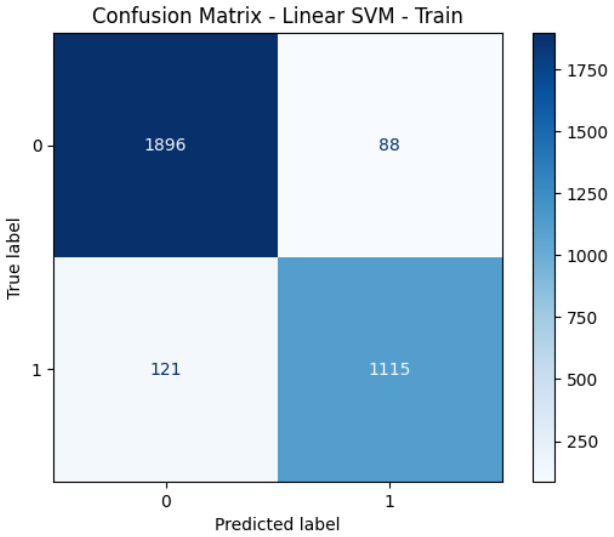

Figure 8: SVM (Linear) Confusion Matrix

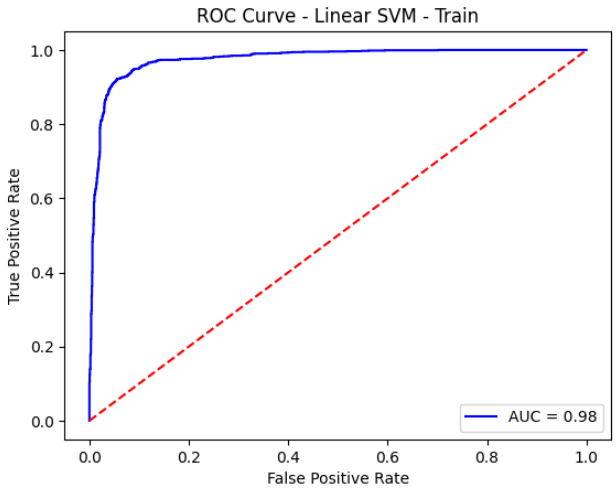

Figure 9: SVM (Linear) ROC Curve

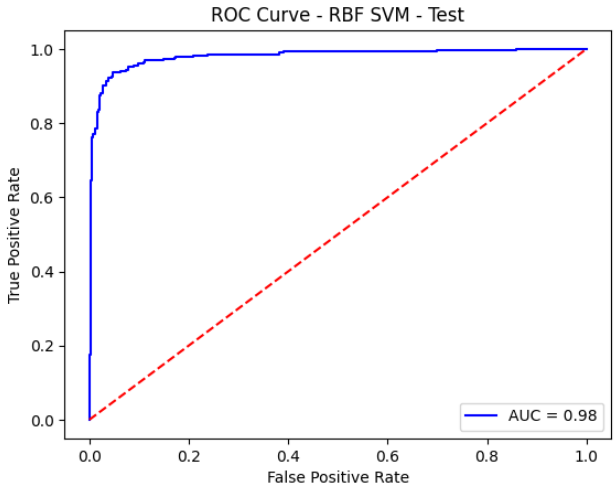

Figure 10: Enter Caption

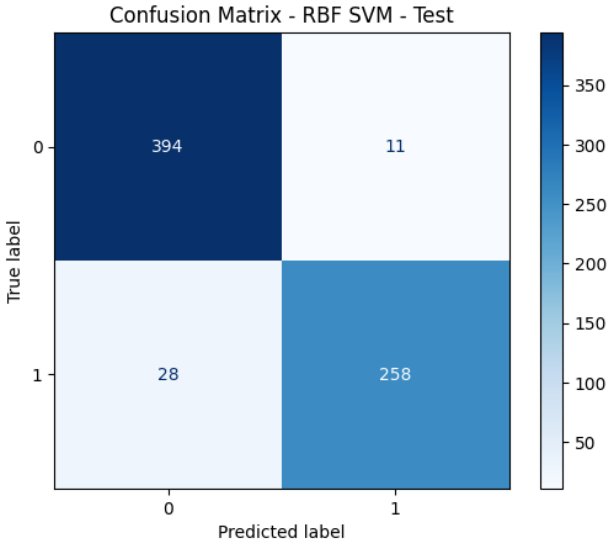

Figure 15: SVM (RBF) Confusion Matrix

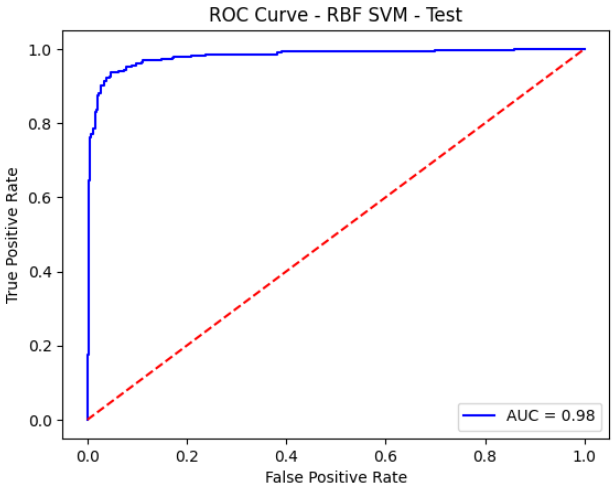

Figure 16: SVM (RBF) ROC Curve

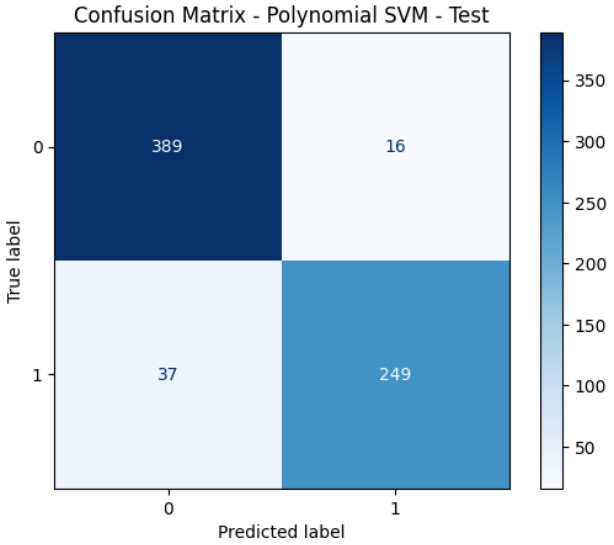

Figure 17: SVM (Polynomial) Confusion Matrix

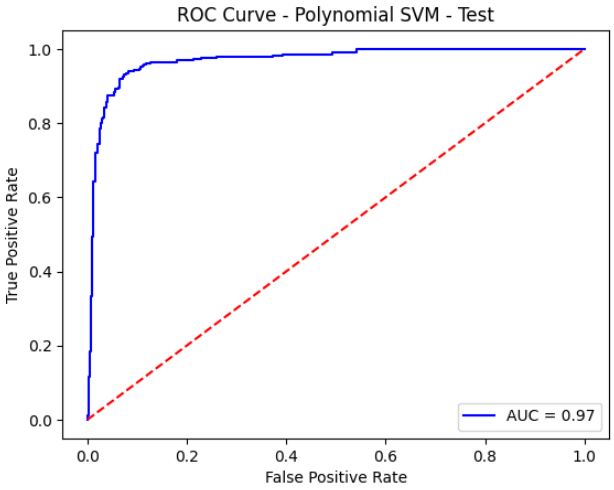

Figure 18: SVM (Polynomial) ROC Curve

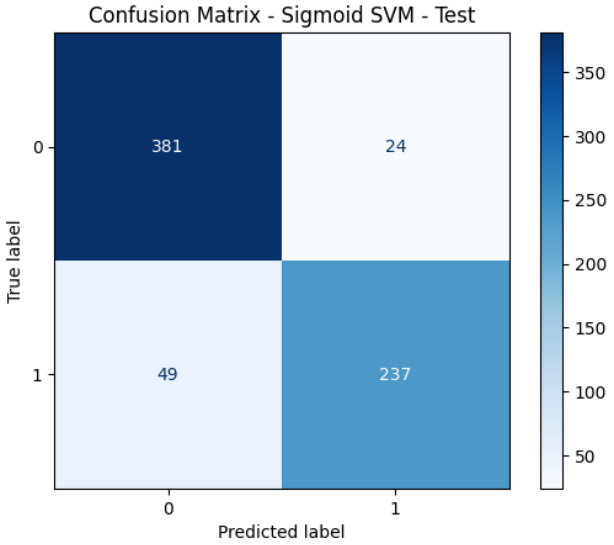

Figure 19: SVM (Sigmoid) Confusion Matrix

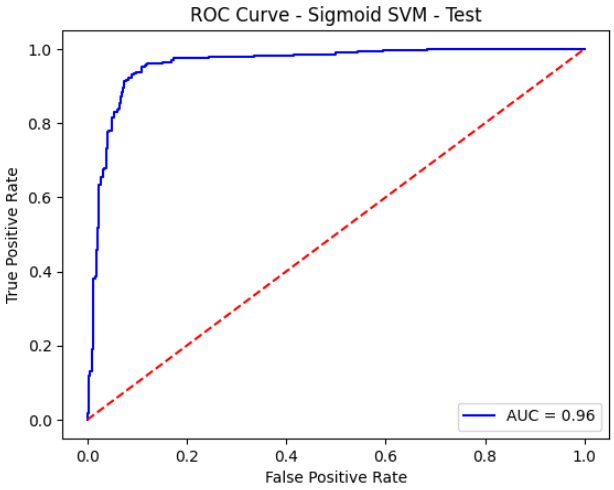

Figure 20: SVM (Sigmoid) ROC Curve