Sri Sivasubramaniya Nadar College of Engineering, Chennai

(An Autonomous Institution Affiliated to Anna University)

Degree & Branch: Integrated M.Tech. Computer Science & Engineering
Semester: V

Course Code & Title: ICS1512 - Machine Learning Algorithms Laboratory

Academic Year: 2025-2026 (Odd) **Batch:** 2023-2028

Name: Vishwajith L K **Reg. No.:** 3122237001061

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

1. 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.

2. Libraries Used

- **NumPy**: For numerical operations and matrix manipulations
- **Pandas**: For data cleaning, preprocessing, and analysis
- **Scikit-learn**: For implementing Linear Regression and evaluating metrics
- **Matplotlib & Seaborn**: For data visualization and statistical plots

3. Objectives

- Perform preprocessing and cleaning of data
- Conduct EDA to better understand relationships between variables
- Implement three different classification models along with their variants for classifying email as spam or ham
- Evaluate every model's performance using error metrics
- Interpret the results.

4. Code Implementation

1. Loading the Dataset

```
import pandas as pd
import numpy as np
import sklearn as sk
from sklearn.linear_model import LinearRegression
import kagglehub

path = kagglehub.dataset_download("somesh24/spambase")

training_df = pd.read_csv(f"{path}/spambase_csv.csv")
target_col = training_df.columns[-1]
training_df.head(10)
```

	word_freq_make	word_freq_address	word_freq_all	word_freq_3d	word_freq_our	word_freq_over	word_freq_remove	word_freq_internet	word_freq_order	word_freq_mail
0	0.00	0.64	0.64	0.0	0.32	0.00	0.00	0.00	0.00	0.00
1	0.21	0.28	0.50	0.0	0.14	0.28	0.21	0.07	0.00	0.94
2	0.06	0.00	0.71	0.0	1.23	0.19	0.19	0.12	0.64	0.25
3	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63
4	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63
5	0.00	0.00	0.00	0.0	1.85	0.00	0.00	1.85	0.00	0.00
6	0.00	0.00	0.00	0.0	1.92	0.00	0.00	0.00	0.00	0.64
7	0.00	0.00	0.00	0.0	1.88	0.00	0.00	1.88	0.00	0.00
8	0.15	0.00	0.46	0.0	0.61	0.00	0.30	0.00	0.92	0.76
9	0.06	0.12	0.77	0.0	0.19	0.32	0.38	0.00	0.06	0.00

10 rows × 58 columns

Figure 1: Dataset

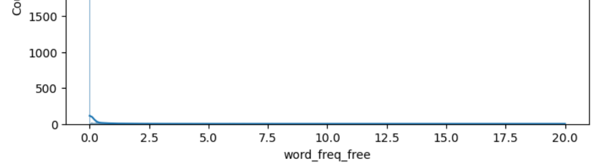
2. Data Preprocessing

```
import seaborn as sns
# Numerical features
num_features = training_df.select_dtypes(include=['int64', 'float64']).columns.tolist
num_features = [col for col in num_features if col != target_col]

# Categorical features
cat_features = training_df.select_dtypes(include=['object', 'category']).columns.toli

from sklearn.preprocessing import MinMaxScaler,StandardScaler

for col in features:
  if col in relevant_features:
    if features[col] == 1:
        scaler = MinMaxScaler()
    else:
        scaler = StandardScaler()
```

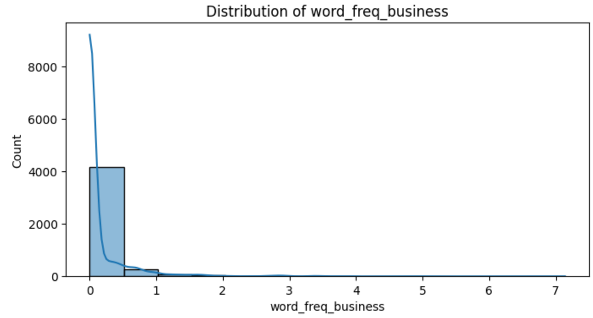
```
    # Reshape the column data to 2D array
    training_df[col] = scaler.fit_transform(training_df[col].values.reshape(-1, 1))
```

3. Exploratory Data Analysis

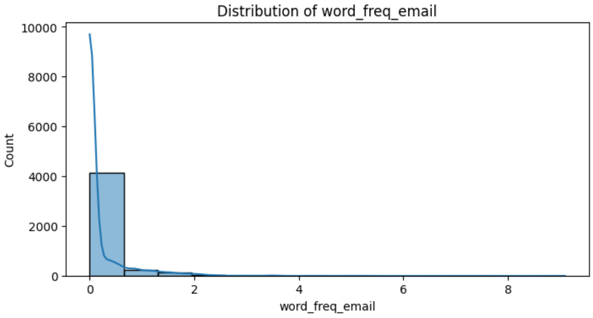
```
from scipy.stats import shapiro

fig, axes = plt.subplots(nrows=len(num_features), ncols=1, figsize=(8, len(num_featur
fig.tight_layout(pad=5.0)
features = {}
for i, column in enumerate(num_features):
    # Plot histogram with KDE
    sns.histplot(training_df[column], kde=True, ax=axes[i])
    axes[i].set_title(f'Distribution of {column}')

    # Shapiro-Wilk normality test
    stat, p = shapiro(training_df[column].dropna())

    # Add test result as text on plot
    result = 1 if p > 0.05 else 0
    features[column]=result

plt.show()


#Correlation Heatmap: To identify multicollinearity and relationships among features.

target_corr = training_df[num_features + [target_col]].corr()[target_col].drop(target
threshold = 0.2
relevant_features = target_corr[abs(target_corr) >= threshold].index.tolist()

print(f"Selected features with correlation >= {threshold}:")
print(relevant_features)
relevant_features.append(target_col)
corr = training_df[relevant_features].corr()
plt.figure(figsize=(20,20))
# Plot heatmap
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap of Relevant Numerical Features')

plt.show()
```



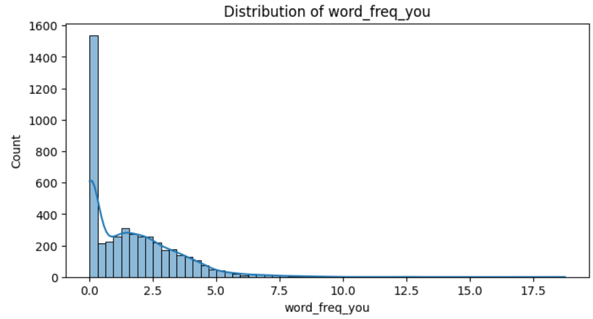
Distribution of word_freq_business



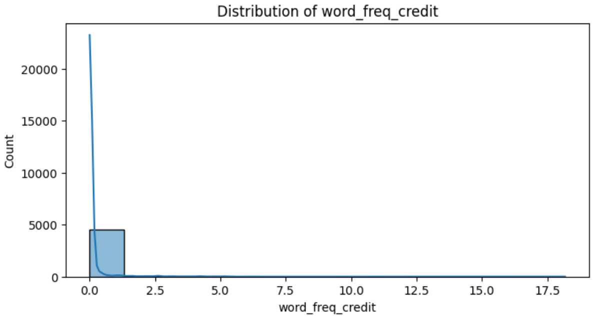
Distribution of word_freq_email



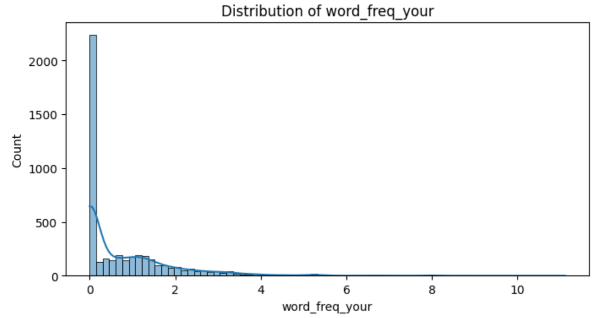
Distribution of word_freq_you



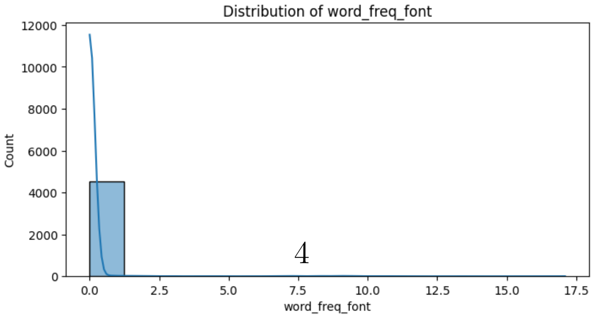
Distribution of word_freq_credit



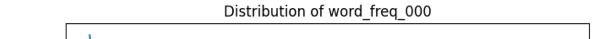
Distribution of word_freq_your



Distribution of word_freq_font



Distribution of word_freq_000



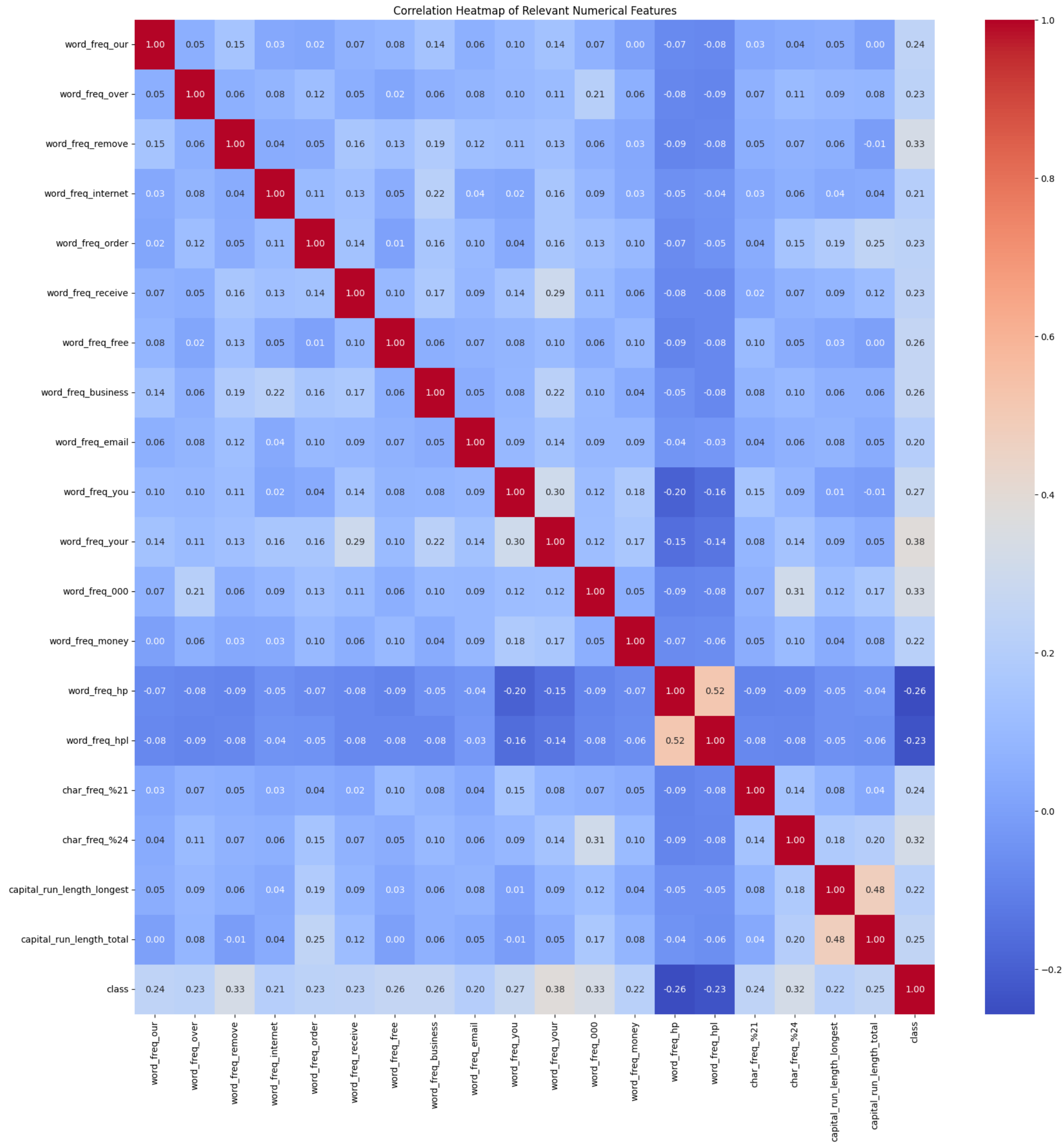


Figure 3: Scatter Plot and Correlation Heatmap

4. Split the dataset

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

X = training_df.drop(columns=target_col)
Y = training_df[target_col]
x_train,x_test, y_train, y_test = train_test_split(X,Y,test_size=0.2)
```

5. Hyperparameter Tuning

Naive Bayes : Gaussian

```
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

param_grid = {
    'var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6, 1e-5],  # valid for GaussianNB
    'priors': [None]  # can also try custom priors, but None is common
}

grid_search = GridSearchCV(GaussianNB(), param_grid, cv=kfold, scoring='accuracy', n_
grid_search.fit(x_train, y_train)

# Best parameters and accuracy
print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation Accuracy:", grid_search.best_score_)

# Evaluate on test data
best_model = grid_search.best_estimator_
y_pred = best_model.predict(x_test)
test_acc = accuracy_score(y_test, y_pred)
print("Test Accuracy:", test_acc)
```

Naive Bayes : Multinominal

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score

param_grid = {
    'alpha': [0.01, 0.1, 0.2, 0.5, 1.0, 2.0],
    'fit_prior': [True, False]
}

grid_search = GridSearchCV(MultinomialNB(), param_grid, cv=kfold, scoring='accuracy')
grid_search.fit(x_train, y_train)

print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation Accuracy:", grid_search.best_score_)

best_model = grid_search.best_estimator_
y_pred = best_model.predict(x_test)
test_acc = accuracy_score(y_test, y_pred)
print("Test Accuracy:", test_acc)
```

Naive Bayes : Bernoulli

```
from sklearn.model_selection import GridSearchCV
param_grid = {
    'alpha': [0.01, 0.1, 0.5, 1.0, 2.0],
    'binarize': [0.0, 0.1, 0.5, 1.0],
    'fit_prior': [True, False]
}

# Set up GridSearchCV with KFold
grid_search = GridSearchCV(BernoulliNB(), param_grid, cv=kfold, scoring='accuracy', n
grid_search.fit(x_train, y_train)

# Best parameters and accuracy
print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation Accuracy:", grid_search.best_score_)

# Evaluate on test data
best_model = grid_search.best_estimator_
y_pred = best_model.predict(x_test)
test_acc = accuracy_score(y_test, y_pred)
print("Test Accuracy:", test_acc)
```

KNN : Vary k

```
#for k=1,3,5,7
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score

# Example KNN hyperparameter grid
param_grid = {
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan', 'minkowski']
}

grid_search = GridSearchCV(KNeighborsClassifier(n_neighbors=k), param_grid, cv=kfold,
grid_search.fit(x_train, y_train)

print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation Accuracy:", grid_search.best_score_)

best_model = grid_search.best_estimator_
y_pred = best_model.predict(x_test)
test_acc = accuracy_score(y_test, y_pred)
print("Test Accuracy:", test_acc)
```

KNN : Ball Tree

```
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
import time
param_grid = {
    'n_neighbors': [1,3,5,7],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan', 'minkowski']
}
start_train = time.time()
grid_search = GridSearchCV(KNeighborsClassifier(algorithm='ball_tree'), param_grid, c
grid_search.fit(x_train, y_train)
end_train = time.time()
print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation Accuracy:", grid_search.best_score_)

best_model = grid_search.best_estimator_
start_pred = time.time()
y_pred = best_model.predict(x_test)
test_acc = accuracy_score(y_test, y_pred)
print("Test Accuracy:", test_acc)
end_pred = time.time()
train_time = end_train - start_train
print(f"GridSearch + Training Time: {train_time:.4f} seconds")
pred_time = end_pred - start_pred
print(f"Prediction Time: {pred_time:.4f} seconds")
```

KNN: KD Tree

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
import time
param_grid = {
    'n_neighbors': [1,3,5,7],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan', 'minkowski']
}
start_train = time.time()
grid_search = GridSearchCV(KNeighborsClassifier(algorithm='kd_tree'), param_grid, cv=
grid_search.fit(x_train, y_train)
end_train = time.time()
print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation Accuracy:", grid_search.best_score_)

best_model = grid_search.best_estimator_
start_pred = time.time()
y_pred = best_model.predict(x_test)
test_acc = accuracy_score(y_test, y_pred)
```

```
print("Test Accuracy:", test_acc)
end_pred = time.time()
train_time = end_train - start_train
print(f"GridSearch + Training Time: {train_time:.4f} seconds")
pred_time = end_pred - start_pred
print(f"Prediction Time: {pred_time:.4f} seconds")
```

SVM

```
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix, roc_curve, auc, classification_report
)
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
import numpy as np
param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf', 'poly', 'sigmoid'],
    'gamma': ['scale', 'auto']
}

kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

grid_search = GridSearchCV(
    estimator=SVC(),
    param_grid=param_grid,
    cv=kfold,
    scoring='accuracy',
    verbose=1,
    n_jobs=-1
)
grid_search.fit(x_train, y_train)

# Extract all fold scores
results_df = pd.DataFrame(grid_search.cv_results_)

# Display only parameters + fold scores
fold_cols = [f'split{i}_test_score' for i in range(kfold.n_splits)]
print(results_df[['params'] + fold_cols])
```

6. Model Training

Naive Bayes : Gaussian

```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(x_train, y_train)
```

Naive Bayes : Multinominal

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

Naive Bayes : Bernoulli

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

KNN: Vary k

```
from sklearn.neighbors import KNeighborsClassifier
#for k=1,3,5,7
model = KNeighborsClassifier(n_neighbors=k)
model.fit(x_train, y_train)
```

KNN: KD Tree

```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=5, algorithm='kd_tree')
model.fit(x_train, y_train)
```

KNN: Ball Tree

```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=k, algorithm='ball_tree')
model.fit(x_train, y_train)
```

SVM: Linear

```
import time
st = time.time()
model = SVC(C= 10, gamma='auto', kernel= 'linear')
model.fit(x_train, y_train)
edt = time.time()
print("Training time: ", edt-st)
```

SVM : Poly

```
import time
st = time.time()
model = SVC(C= 10, gamma='auto', kernel='poly')
model.fit(x_train, y_train)
edt = time.time()
print("Training time: ", edt-st)
```

SVM: RBF

```
import time
st = time.time()
model = SVC(C= 1, gamma='auto', kernel='rbf')
model.fit(x_train, y_train)
edt = time.time()
print("Training time: ", edt-st)
```

SVM: Sigmoid

```
import time
st = time.time()
model = SVC(C= 0.1, gamma= 'auto', kernel= 'sigmoid')
model.fit(x_train, y_train)
edt = time.time()
print("Training time: ", edt-st)
```

7. Model Evaluation

```
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix, roc_curve, auc, classification_report
)
import matplotlib.pyplot as plt
y_pred = model.predict(x_test)
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred, average='weighted')
rec = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print(classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
ax= plt.subplot()
sns.heatmap(cm, annot=True, fmt='g', ax=ax)
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
```

	precision	recall	f1-score	support
0	0.96	0.97	0.96	551
1	0.95	0.94	0.94	370
accuracy			0.96	921
macro avg	0.96	0.95	0.95	921
weighted avg	0.96	0.96	0.96	921

Figure 4: Evaluation Metrics on Test Set

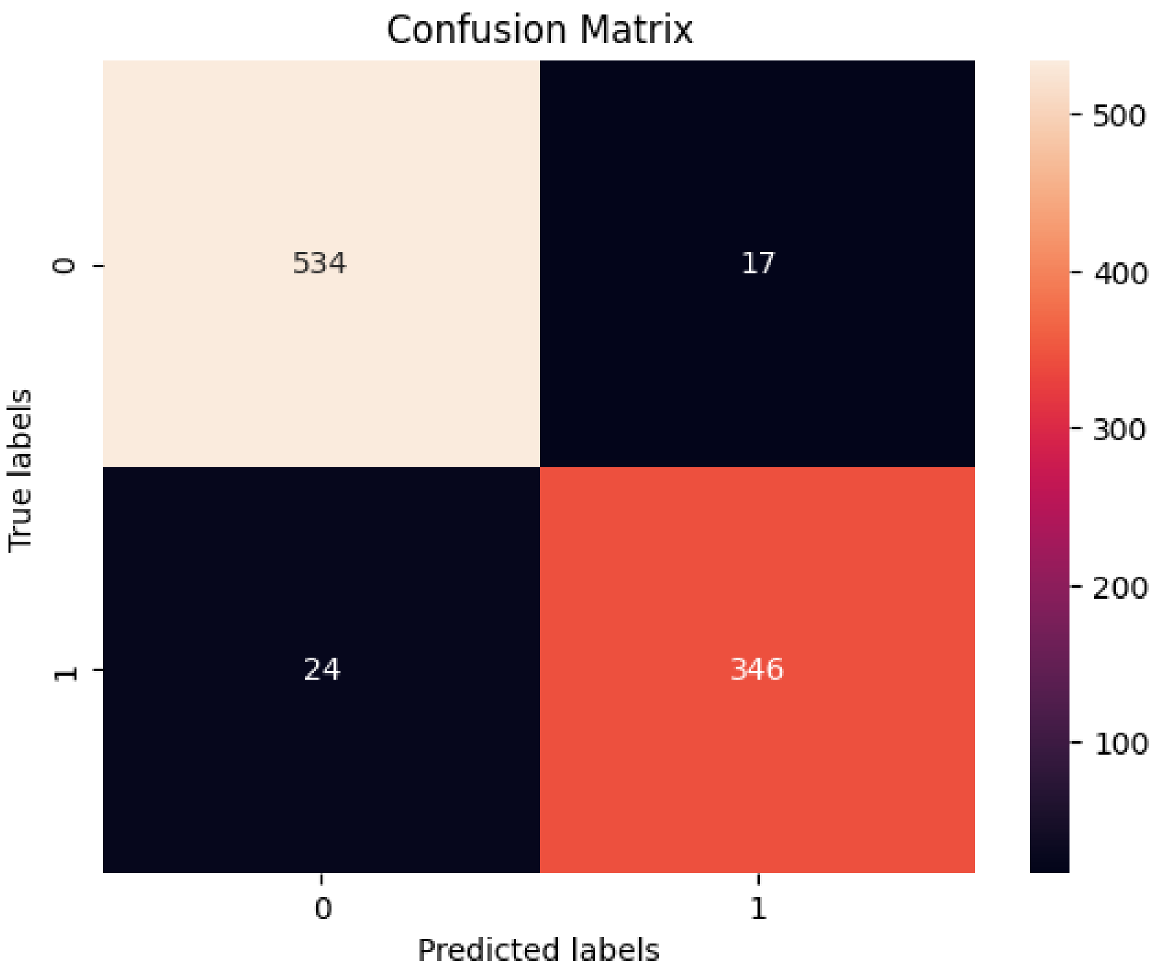


Figure 5: Confusion Matrix

8. Visualization of the Results

```
# ROC
y_scores = model.decision_function(x_test)

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

# Plot
plt.figure()
plt.plot(fpr, tpr, label=f"ROC curve (AUC = {roc_auc:.2f})", color='green')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver Operating Characteristic (ROC) Curve - SVM RBF")
plt.legend(loc="lower right")
plt.grid(True)
```

```
plt.show()
```

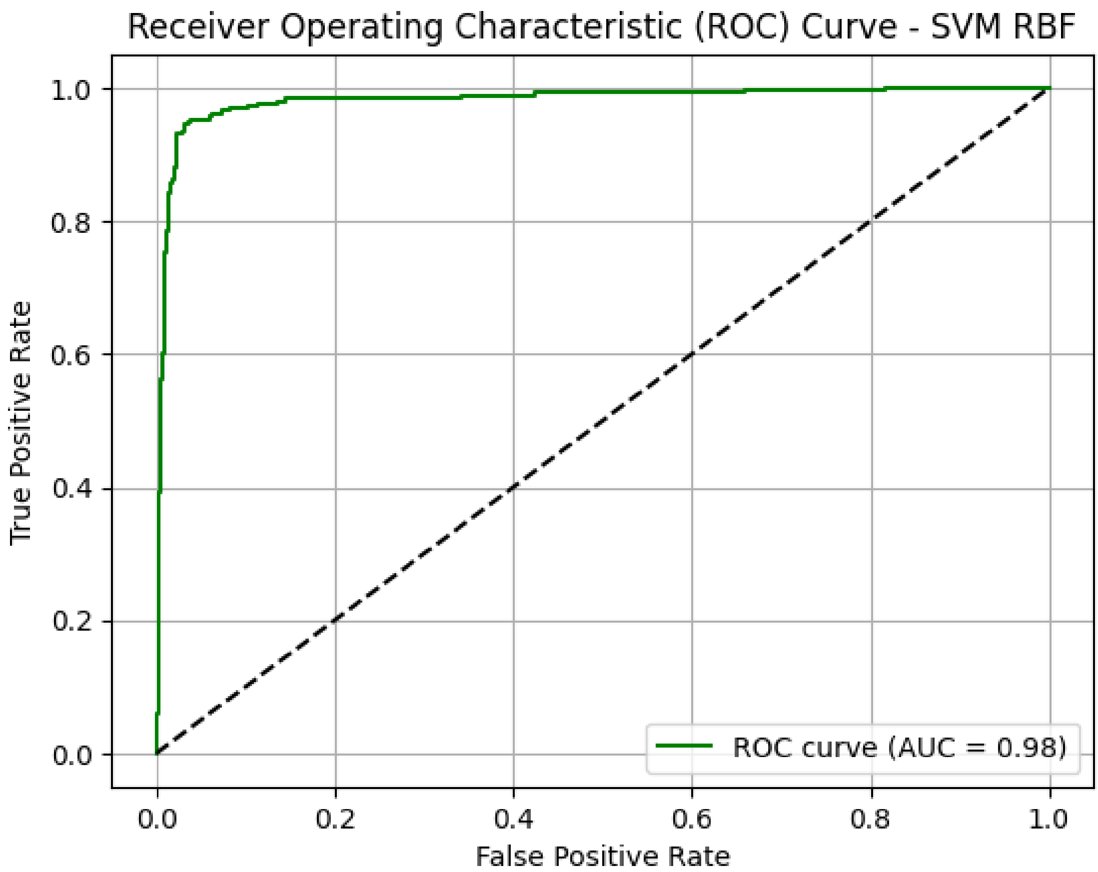


Figure 6: ROC plot

6. Results Tables

Table 1: Naïve Bayes Variant Comparison

Metric	Gaussian NB	Multinomial NB	Bernoulli NB
Accuracy	83%	81%	90%
Precision	86%	81%	89%
Recall	83%	81%	89%
F1 Score	83%	81%	89%

Table 2: KNN Performance for Different k Values

k	Accuracy	Precision	Recall	F1 Score
1	89%	91%	89%	90%
3	83%	85%	83%	83%
5	81%	84%	81%	82%
7	81%	83%	81%	82%

Table 3: KNN Comparison (KDTree vs BallTree)

Metric	KDTree	BallTree
Accuracy	85%	85%
Precision	87%	86%
Recall	85%	85%
F1 Score	85%	85%
GridSearch + Training Time (s)	7.2643 s	9.7231 s

Table 4: SVM Performance with Different Kernels

Kernel	Hyperparameters	Accuracy	F1 Score	Training Time
Linear	C = 10	94%	93%	5.9636 s
Polynomial	C = 10, degree = 3, gamma = 'auto'	93%	93%	0.4592 s
RBF	C = 1, gamma = 'auto'	96%	96%	0.30369 s
Sigmoid	C = 0.1, gamma = 'auto'	85%	85%	0.46634 s

Table 5: Cross-Validation Scores for Each Model

Fold	Naïve Bayes Accuracy	KNN Accuracy	SVM Accuracy
Fold 1	89.5%	83.7%	93.48%
Fold 2	88.9%	82.5%	94.84%
Fold 3	90.2%	81.4%	94.16%
Fold 4	90.4%	81.2%	92.53%
Fold 5	90.1%	79.6%	91.98%
Average	89.8%	81.7%	93.4%

7. Best Practices

- Performed thorough data cleaning and preprocessing
- Used train-test split to evaluate model performance
- Implemented feature scaling for better convergence
- Compared multiple evaluation metrics
- Documented all steps and interpretations

8. Learning Outcomes

- Understood how to implement classification algorithms—Naive Bayes, K-Nearest Neighbors (KNN), and Support Vector Machine (SVM) in a email spam or ham dataset
- Learned the role of data preprocessing and feature selection in model performance

- Developed skills in EDA, visualization, and interpreting regression results
- Realized the importance of hyper parameter tuning and cross-validation for robust models