Sri Sivasubramaniya Nadar College of Engineering, Chennai

(An autonomous Institution affiliated to Anna University)

Degree & Branch	M.Tech Computer Science & Engineering Integrated (5 Years)	**Semester** V
Subject Code & Name	ICS1512 & Machine Learning Algorithms Laboratory	
Academic year	2025–2026 (Odd)	**Batch:** 2023–2028

Name: I.S.Rajesh **Register No.: 3122237001042**

Experiment #3: Email Ham and Spam Classification using kNN, SVM and Naive Bayes and their variations

Aim

To build and compare different classification algorithms for email spam detection (Ham vs Spam) using classical ML models: BernoulliNB, MultinomialNB, GaussianNB, Support Vector Classifier (4 kernels), and k-Nearest Neighbors (kNN) with KD-Tree and Ball-Tree options.

Libraries used:
`numpy, pandas, sklearn, matplotlib, seaborn`

Objective:
Apply Linear Regression and Support Vector Regression to predict loan amount sanctioned to users using the provided dataset. Use K-Fold Cross Validation to validate models after effective splitting and compare performance across different SVR kernels.

Python Codes for All Models

Listing 1: GaussianNB

```
# (i) Import necessary libraries
```

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

from sklearn.model_selection import train_test_split, KFold,
    ↪ cross_val_score
from sklearn.preprocessing import MinMaxScaler, StandardScaler,
    ↪ LabelEncoder, OneHotEncoder
from sklearn.metrics import mean_squared_error, root_mean_squared_error
    ↪ , mean_absolute_error, r2_score, accuracy_score, precision_score,
    ↪  recall_score, f1_score, classification_report, confusion_matrix,
    ↪  roc_auc_score, roc_curve, ConfusionMatrixDisplay
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB

# (ii) Import dataset
df = pd.read_csv('spambase_csv.xls')
target = 'class'
# (iii) EDA and Preprocessing

def is_normal(series):
    skew = series.skew()
    return -0.5 <= skew <= 0.5

def has_outliers(series):
    Q1 = series.quantile(0.25)
    Q3 = series.quantile(0.75)
    IQR = Q3 - Q1
    outliers = ((series < (Q1 - 1.5 * IQR)) | (series > (Q3 + 1.5 * IQR
        ↪ ))).sum()
    return outliers > 0
# Separate types
numerical_cols = df.select_dtypes(include=['int64', 'float64']).columns
    ↪ .tolist()
categorical_cols = df.select_dtypes(include=['object', 'category']).
    ↪ columns.tolist()
numerical_cols = [col for col in numerical_cols if col != target]

# --- Missing Values Handling ---
for col in df.columns:
    if df[col].isnull().sum() > 0:
        if col in numerical_cols:
            if is_normal(df[col]):
                if has_outliers(df[col]):
                    df[col].fillna(df[col].median(), inplace=True)
                else:
                    df[col].fillna(df[col].mean(), inplace=True)
            else:
                df[col].fillna(df[col].median(), inplace=True)
        else:
            df[col].fillna(df[col].mode()[0], inplace=True)  #
                ↪ categorical

# --- Outlier Removal (for numerical columns only) ---
for col in numerical_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
```

```
    #df = df[~((df[col] < Q1 - 1.5 * IQR) | (df[col] > Q3 + 1.5 * IQR))]
# --- Drop rows where target is missing ---
df.dropna(subset=[target], inplace=True)
# --- Encoding categorical features ---
is_classification = True

if is_classification:
    le = LabelEncoder()
    df[target] = le.fit_transform(df[target])  # encode target
    for col in categorical_cols:
        df = pd.get_dummies(df, columns=categorical_cols)  # label encoding for classification
else:
    # Regression: target guided ordinal encoding
    for col in categorical_cols:
        ordered_labels = df.groupby(col)[target].mean().sort_values().index
        mapping = {k: i for i, k in enumerate(ordered_labels)}
        df[col] = df[col].map(mapping)
for col in numerical_cols:
    if is_normal(df[col]):
        scaler = StandardScaler()
    elif has_outliers(df[col]):
        scaler = StandardScaler()
    else:
        scaler = MinMaxScaler()

    df[[col]] = scaler.fit_transform(df[[col]])
# --- Histogram Subplots ---
n_cols = 5  # Number of plots per row
n_rows = int(np.ceil(len(numerical_cols) / n_cols))

fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 4 * n_rows))
axes = axes.flatten()

for i, col in enumerate(numerical_cols):
    df[col].hist(ax=axes[i], bins=30)
    axes[i].set_title(col)

# Turn off unused subplots
for j in range(i+1, len(axes)):
    axes[j].axis('off')

fig.suptitle("Histogram of Features", fontsize=20)
plt.tight_layout()
plt.show()

# --- Boxplot Subplots ---
fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 4 * n_rows))
axes = axes.flatten()

for i, col in enumerate(numerical_cols):
    axes[i].boxplot(df[col])
    axes[i].set_title(col)

for j in range(i+1, len(axes)):
    axes[j].axis('off')
```

```python
fig.suptitle("Boxplot for Outlier Detection", fontsize=20)
plt.tight_layout()
plt.show()

# --- Correlation Heatmap ---
plt.figure(figsize=(20, 16))
sns.heatmap(df.corr(), annot=False, fmt=".2f", cmap='coolwarm', linewidths=0.5)
plt.title("Feature Correlation Heatmap", fontsize=18)
plt.xticks(rotation=45, ha='right', fontsize=8)
plt.yticks(fontsize=8)
plt.tight_layout()
plt.show()

# (iv) Splitting dataset

X = df.drop(columns=[target])
y = df[target]

# Splitting: Train (60%), Validation (20%), Test (20%)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4, 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)


print(f'Train: {X_train.shape}, Validation: {X_val.shape}, Test: {X_test.shape}')

df.to_csv('updated_spam.csv')
# (v) Model Training

model = GaussianNB()

model.fit(X_train, y_train)

# Predictions
y_val_pred = model.predict(X_val)
y_test_pred = model.predict(X_test)
# (vi) Evaluation

# Evaluating Model using Performance Metrics

def evaluate_model(y_true, y_pred, is_classification, X, model, dataset_name):
    print(f"\n Evaluation - {dataset_name}")
    if is_classification:
        print("Accuracy :", round(accuracy_score(y_true, y_pred), 4))
        print("Precision:", round(precision_score(y_true, y_pred, average='weighted'), 4))
        print("Recall   :", round(recall_score(y_true, y_pred, average='weighted'), 4))
        print("F1 Score :", round(f1_score(y_true, y_pred, average='weighted'), 4))
        print("\nClassification Report:\n", classification_report(y_true, y_pred))
```

```
        # ROC Curve: Only for binary classification
        if len(np.unique(y_true)) == 2 and model is not None and
            ↪ hasattr(model, "predict_proba"):
            y_probs = model.predict_proba(X)[:, 1]
            fpr, tpr, _ = roc_curve(y_true, y_probs)
            auc_score = roc_auc_score(y_true, y_probs)
            print("ROC AUC Score:", round(auc_score, 4))

            # Plot ROC
            plt.figure(figsize=(6, 4))
            plt.plot(fpr, tpr, label=f"AUC = {auc_score:.4f}")
            plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
            plt.xlabel("False Positive Rate")
            plt.ylabel("True Positive Rate")
            plt.title(f"ROC Curve - {dataset_name}")
            plt.legend()
            plt.grid(True)
            plt.tight_layout()
            plt.show()
    else:
        n, p = X.shape
        r2 = r2_score(y_true, y_pred)
        adjusted_r2 = 1 - (1 - r2) * (n - 1) / (n - p - 1)
        print("Mean Squared Error:", mean_squared_error(y_true, y_pred)
            ↪ )
        print("Root Mean Squared Error:", root_mean_squared_error(
            ↪ y_true, y_pred))
        print("Mean Absolute Error:", mean_absolute_error(y_true,
            ↪ y_pred))
        print("R2 Score:", r2)
        print("Adjusted R2 Score:", adjusted_r2)

# For validation set
evaluate_model(y_val, y_val_pred, is_classification, X_val, model, "
    ↪ Validation Set")

# For test set
evaluate_model(y_test, y_test_pred, is_classification, X_test, model, "
    ↪ Test Set")
# Evaluating Model on Test and Validation Sets (Without Performance
    ↪ Metrics)

def plot_actual_vs_predicted(y_true, y_pred, title):
    plt.figure(figsize=(6, 4))
    plt.scatter(y_true, y_pred, alpha=0.5, edgecolor='k')
    plt.plot([y_true.min(), y_true.max()], [y_true.min(), y_true.max()
        ↪ ], 'r--')
    plt.xlabel("Actual")
    plt.ylabel("Predicted")
    plt.title(f"Actual vs Predicted - {title}")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

def plot_residuals(y_true, y_pred, title):
    residuals = y_true - y_pred
    plt.figure(figsize=(6, 4))
    plt.scatter(y_pred, residuals, alpha=0.5, edgecolor='k')
```

```
    plt.axhline(y=0, color='red', linestyle='--')
    plt.xlabel("Predicted")
    plt.ylabel("Residuals")
    plt.title(f"Residual Plot - {title}")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

def plot_residual_distribution(y_true, y_pred, title):
    residuals = y_true - y_pred
    plt.figure(figsize=(6, 4))
    sns.histplot(residuals, kde=True, color='skyblue')
    plt.title(f"Residual Distribution - {title}")
    plt.xlabel("Residuals")
    plt.ylabel("Frequency")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

def plot_confusion_matrix(y_true, y_pred, title):
    ConfusionMatrixDisplay.from_predictions(y_true, y_pred)
    plt.title(title)
    plt.show()

if not is_classification:
  plot_actual_vs_predicted(y_val, y_val_pred, "Validation Set")
  plot_residuals(y_val, y_val_pred, "Validation Set")
  plot_residual_distribution(y_val, y_val_pred, "Validation Set")

  plot_actual_vs_predicted(y_test, y_test_pred, "Test Set")
  plot_residuals(y_test, y_test_pred, "Test Set")
  plot_residual_distribution(y_test, y_test_pred, "Test Set")
else:
  plot_confusion_matrix(y_test, y_test_pred, "Test Set")
  plot_confusion_matrix(y_val, y_val_pred, "Validation Set")
# (vii) K-Fold Cross Validation

kfold = KFold(n_splits=5, shuffle=True, random_state=42)
if is_classification:
    score = 'accuracy'
else:
    score = 'r2'
cv_results = cross_val_score(model, X, y, cv=kfold, scoring=score)

print("Cross Validation Scores:", cv_results)
print("Average CV Score:", np.mean(cv_results))
```

Listing 2: MultinomialNB

```
# (i) Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, KFold,
   ↪ cross_val_score
from sklearn.preprocessing import MinMaxScaler, StandardScaler,
```

```
    ↪ LabelEncoder, Binarizer
from sklearn.metrics import mean_squared_error, root_mean_squared_error
    ↪ , mean_absolute_error, r2_score, accuracy_score, precision_score,
    ↪  recall_score, f1_score, classification_report, confusion_matrix,
    ↪  roc_auc_score, roc_curve, ConfusionMatrixDisplay
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB

# (ii) Import dataset
df = pd.read_csv('spambase_csv.xls')
target = 'class'
# (iii) EDA and Preprocessing

def is_normal(series):
    skew = series.skew()
    return -0.5 <= skew <= 0.5

def has_outliers(series):
    Q1 = series.quantile(0.25)
    Q3 = series.quantile(0.75)
    IQR = Q3 - Q1
    outliers = ((series < (Q1 - 1.5 * IQR)) | (series > (Q3 + 1.5 * IQR
        ↪ ))).sum()
    return outliers > 0
# Separate types
numerical_cols = df.select_dtypes(include=['int64', 'float64']).columns
    ↪ .tolist()
categorical_cols = df.select_dtypes(include=['object', 'category']).
    ↪ columns.tolist()
numerical_cols = [col for col in numerical_cols if col != target]

# --- Missing Values Handling ---
for col in df.columns:
    if df[col].isnull().sum() > 0:
        if col in numerical_cols:
            if is_normal(df[col]):
                if has_outliers(df[col]):
                    df[col].fillna(df[col].median(), inplace=True)
                else:
                    df[col].fillna(df[col].mean(), inplace=True)
            else:
                df[col].fillna(df[col].median(), inplace=True)
        else:
            df[col].fillna(df[col].mode()[0], inplace=True)  #
                ↪ categorical

# --- Outlier Removal (for numerical columns only) ---
for col in numerical_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    #df = df[~((df[col] < Q1 - 1.5 * IQR) | (df[col] > Q3 + 1.5 * IQR))
        ↪ ]
# --- Drop rows where target is missing ---
df.dropna(subset=[target], inplace=True)
# --- Encoding categorical features ---
is_classification = True

if is_classification:
```

```
    le = LabelEncoder()
    df[target] = le.fit_transform(df[target])  # encode target
    df = pd.get_dummies(df, columns=categorical_cols, drop_first=False)
else:
    # Regression: target guided ordinal encoding
    for col in categorical_cols:
        ordered_labels = df.groupby(col)[target].mean().sort_values().
            ↪ index
        mapping = {k: i for i, k in enumerate(ordered_labels)}
        df[col] = df[col].map(mapping)
# --- Scaling ---
for col in numerical_cols:
    scaler=MinMaxScaler()
    df[[col]] = scaler.fit_transform(df[[col]])

print(df)
# --- Histogram Subplots ---
n_cols = 5  # Number of plots per row
n_rows = int(np.ceil(len(numerical_cols) / n_cols))

fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 4 * n_rows))
axes = axes.flatten()

for i, col in enumerate(numerical_cols):
    df[col].hist(ax=axes[i], bins=30)
    axes[i].set_title(col)

# Turn off unused subplots
for j in range(i+1, len(axes)):
    axes[j].axis('off')

fig.suptitle("Histogram of Features", fontsize=20)
plt.tight_layout()
plt.show()

# --- Boxplot Subplots ---
fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 4 * n_rows))
axes = axes.flatten()

for i, col in enumerate(numerical_cols):
    axes[i].boxplot(df[col])
    axes[i].set_title(col)

for j in range(i+1, len(axes)):
    axes[j].axis('off')

fig.suptitle("Boxplot for Outlier Detection", fontsize=20)
plt.tight_layout()
plt.show()

# --- Correlation Heatmap ---
plt.figure(figsize=(20, 16))
sns.heatmap(df.corr(), annot=False, fmt=".2f", cmap='coolwarm',
   ↪ linewidths=0.5)
plt.title("Feature Correlation Heatmap", fontsize=18)
plt.xticks(rotation=45, ha='right', fontsize=8)
plt.yticks(fontsize=8)
plt.tight_layout()
```

```python
plt.show()

# Optional: Clustered Heatmap (if too many features)
# sns.clustermap(df.corr(), cmap='coolwarm', figsize=(18, 16))
# plt.title("Clustered Correlation Heatmap")
# plt.show()
# (iv) Splitting dataset

X = df.drop(columns=[target])
y = df[target]

# Splitting: Train (60%), Validation (20%), Test (20%)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4, 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)


print(f'Train: {X_train.shape}, Validation: {X_val.shape}, Test: {X_test.shape}')

df.to_csv('updated_spam.csv')
# (v) Model Training

model = MultinomialNB()

model.fit(X_train, y_train)

# Predictions
y_val_pred = model.predict(X_val)
y_test_pred = model.predict(X_test)
# (vi) Evaluation

# Evaluating Model using Performance Metrics

def evaluate_model(y_true, y_pred, is_classification, X, model, dataset_name):
    print(f"\n Evaluation - {dataset_name}")
    if is_classification:
        print("Accuracy :", round(accuracy_score(y_true, y_pred), 4))
        print("Precision:", round(precision_score(y_true, y_pred, average='weighted'), 4))
        print("Recall   :", round(recall_score(y_true, y_pred, average='weighted'), 4))
        print("F1 Score :", round(f1_score(y_true, y_pred, average='weighted'), 4))
        print("\nClassification Report:\n", classification_report(y_true, y_pred))

        # ROC Curve: Only for binary classification
        if len(np.unique(y_true)) == 2 and model is not None and hasattr(model, "predict_proba"):
            y_probs = model.predict_proba(X)[:, 1]
            fpr, tpr, _ = roc_curve(y_true, y_probs)
            auc_score = roc_auc_score(y_true, y_probs)
            print("ROC AUC Score:", round(auc_score, 4))

            # Plot ROC
```

```
            plt.figure(figsize=(6, 4))
            plt.plot(fpr, tpr, label=f"AUC = {auc_score:.4f}")
            plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
            plt.xlabel("False Positive Rate")
            plt.ylabel("True Positive Rate")
            plt.title(f"ROC Curve - {dataset_name}")
            plt.legend()
            plt.grid(True)
            plt.tight_layout()
            plt.show()
    else:
        n, p = X.shape
        r2 = r2_score(y_true, y_pred)
        adjusted_r2 = 1 - (1 - r2) * (n - 1) / (n - p - 1)
        print("Mean Squared Error:", mean_squared_error(y_true, y_pred)
            )
        print("Root Mean Squared Error:", root_mean_squared_error(
            y_true, y_pred))
        print("Mean Absolute Error:", mean_absolute_error(y_true,
            y_pred))
        print("R2 Score:", r2)
        print("Adjusted R2 Score:", adjusted_r2)

# For validation set
evaluate_model(y_val, y_val_pred, is_classification, X_val, model, "
    Validation Set")

# For test set
evaluate_model(y_test, y_test_pred, is_classification, X_test, model, "
    Test Set")
# Evaluating Model on Test and Validation Sets (Without Performance
    Metrics)

def plot_actual_vs_predicted(y_true, y_pred, title):
    plt.figure(figsize=(6, 4))
    plt.scatter(y_true, y_pred, alpha=0.5, edgecolor='k')
    plt.plot([y_true.min(), y_true.max()], [y_true.min(), y_true.max()
        ], 'r--')
    plt.xlabel("Actual")
    plt.ylabel("Predicted")
    plt.title(f"Actual vs Predicted - {title}")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

def plot_residuals(y_true, y_pred, title):
    residuals = y_true - y_pred
    plt.figure(figsize=(6, 4))
    plt.scatter(y_pred, residuals, alpha=0.5, edgecolor='k')
    plt.axhline(y=0, color='red', linestyle='--')
    plt.xlabel("Predicted")
    plt.ylabel("Residuals")
    plt.title(f"Residual Plot - {title}")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

def plot_residual_distribution(y_true, y_pred, title):
```

```
    residuals = y_true - y_pred
    plt.figure(figsize=(6, 4))
    sns.histplot(residuals, kde=True, color='skyblue')
    plt.title(f"Residual Distribution - {title}")
    plt.xlabel("Residuals")
    plt.ylabel("Frequency")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

def plot_confusion_matrix(y_true, y_pred, title):
    ConfusionMatrixDisplay.from_predictions(y_true, y_pred)
    plt.title(title)
    plt.show()

if not is_classification:
  plot_actual_vs_predicted(y_val, y_val_pred, "Validation Set")
  plot_residuals(y_val, y_val_pred, "Validation Set")
  plot_residual_distribution(y_val, y_val_pred, "Validation Set")

  plot_actual_vs_predicted(y_test, y_test_pred, "Test Set")
  plot_residuals(y_test, y_test_pred, "Test Set")
  plot_residual_distribution(y_test, y_test_pred, "Test Set")
else:
  plot_confusion_matrix(y_test, y_test_pred, "Test Set")
  plot_confusion_matrix(y_val, y_val_pred, "Validation Set")
# (vii) K-Fold Cross Validation

kfold = KFold(n_splits=5, shuffle=True, random_state=42)
if is_classification:
    score = 'accuracy'
else:
    score = 'r2'
cv_results = cross_val_score(model, X, y, cv=kfold, scoring=score)

print("Cross Validation Scores:", cv_results)
print("Average CV Score:", np.mean(cv_results))
```

Listing 3: BernoulliNB

```
# (i) Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, KFold,
   ↪ cross_val_score
from sklearn.preprocessing import MinMaxScaler, StandardScaler,
   ↪ LabelEncoder, Binarizer
from sklearn.metrics import mean_squared_error, root_mean_squared_error
   ↪ , mean_absolute_error, r2_score, accuracy_score, precision_score,
   ↪  recall_score, f1_score, classification_report, confusion_matrix,
   ↪  roc_auc_score, roc_curve, ConfusionMatrixDisplay
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB

# (ii) Import dataset
df = pd.read_csv('spambase_csv.xls')
```

```
target = 'class'
# (iii) EDA and Preprocessing

def is_normal(series):
    skew = series.skew()
    return -0.5 <= skew <= 0.5

def has_outliers(series):
    Q1 = series.quantile(0.25)
    Q3 = series.quantile(0.75)
    IQR = Q3 - Q1
    outliers = ((series < (Q1 - 1.5 * IQR)) | (series > (Q3 + 1.5 * IQR
        ↪ ))).sum()
    return outliers > 0
# Separate types
numerical_cols = df.select_dtypes(include=['int64', 'float64']).columns
    ↪ .tolist()
categorical_cols = df.select_dtypes(include=['object', 'category']).
    ↪ columns.tolist()
numerical_cols = [col for col in numerical_cols if col != target]

# --- Missing Values Handling ---
for col in df.columns:
    if df[col].isnull().sum() > 0:
        if col in numerical_cols:
            if is_normal(df[col]):
                if has_outliers(df[col]):
                    df[col].fillna(df[col].median(), inplace=True)
                else:
                    df[col].fillna(df[col].mean(), inplace=True)
            else:
                df[col].fillna(df[col].median(), inplace=True)
        else:
            df[col].fillna(df[col].mode()[0], inplace=True)  #
                ↪ categorical

# --- Outlier Removal (for numerical columns only) ---
for col in numerical_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    #df = df[~((df[col] < Q1 - 1.5 * IQR) | (df[col] > Q3 + 1.5 * IQR))
        ↪ ]
# --- Drop rows where target is missing ---
df.dropna(subset=[target], inplace=True)
# --- Encoding categorical features ---
is_classification = True

if is_classification:
    le = LabelEncoder()
    df[target] = le.fit_transform(df[target])  # encode target
    df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)
        ↪ # Similar to one-hot encoding as BernoulliNB requires binary
        ↪ values
else:
    # Regression: target guided ordinal encoding
    for col in categorical_cols:
        ordered_labels = df.groupby(col)[target].mean().sort_values().
```

```
        ↪ index
        mapping = {k: i for i, k in enumerate(ordered_labels)}
        df[col] = df[col].map(mapping)
# --- Scaling ---
for col in numerical_cols:
    binarizer = Binarizer(threshold=0.0)
    df[[col]] = binarizer.fit_transform(df[[col]])

print(df)
# --- Histogram Subplots ---
n_cols = 5  # Number of plots per row
n_rows = int(np.ceil(len(numerical_cols) / n_cols))

fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 4 * n_rows))
axes = axes.flatten()

for i, col in enumerate(numerical_cols):
    df[col].hist(ax=axes[i], bins=30)
    axes[i].set_title(col)

# Turn off unused subplots
for j in range(i+1, len(axes)):
    axes[j].axis('off')

fig.suptitle("Histogram␣of␣Features", fontsize=20)
plt.tight_layout()
plt.show()

# --- Boxplot Subplots ---
fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 4 * n_rows))
axes = axes.flatten()

for i, col in enumerate(numerical_cols):
    axes[i].boxplot(df[col])
    axes[i].set_title(col)

for j in range(i+1, len(axes)):
    axes[j].axis('off')

fig.suptitle("Boxplot␣for␣Outlier␣Detection", fontsize=20)
plt.tight_layout()
plt.show()

# --- Correlation Heatmap ---
plt.figure(figsize=(20, 16))
sns.heatmap(df.corr(), annot=False, fmt=".2f", cmap='coolwarm',
   ↪ linewidths=0.5)
plt.title("Feature␣Correlation␣Heatmap", fontsize=18)
plt.xticks(rotation=45, ha='right', fontsize=8)
plt.yticks(fontsize=8)
plt.tight_layout()
plt.show()

# Optional: Clustered Heatmap (if too many features)
# sns.clustermap(df.corr(), cmap='coolwarm', figsize=(18, 16))
# plt.title("Clustered Correlation Heatmap")
# plt.show()
# (iv) Splitting dataset
```

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

# Splitting: Train (60%), Validation (20%), Test (20%)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size
    ↪ =0.4, 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)


print(f'Train:␣{X_train.shape},␣Validation:␣{X_val.shape},␣Test:␣{
    ↪ X_test.shape}')

df.to_csv('updated_spam.csv')
# (v) Model Training

model = BernoulliNB()

model.fit(X_train, y_train)

# Predictions
y_val_pred = model.predict(X_val)
y_test_pred = model.predict(X_test)
# (vi) Evaluation

# Evaluating Model using Performance Metrics

def evaluate_model(y_true, y_pred, is_classification, X, model,
    ↪ dataset_name):
    print(f"\n␣Evaluation␣-␣{dataset_name}")
    if is_classification:
        print("Accuracy␣:", round(accuracy_score(y_true, y_pred), 4))
        print("Precision:", round(precision_score(y_true, y_pred,
            ↪ average='weighted'), 4))
        print("Recall␣␣␣:", round(recall_score(y_true, y_pred, average=
            ↪ 'weighted'), 4))
        print("F1␣Score␣:", round(f1_score(y_true, y_pred, average='
            ↪ weighted'), 4))
        print("\nClassification␣Report:\n", classification_report(
            ↪ y_true, y_pred))

        # ROC Curve: Only for binary classification
        if len(np.unique(y_true)) == 2 and model is not None and
            ↪ hasattr(model, "predict_proba"):
            y_probs = model.predict_proba(X)[:, 1]
            fpr, tpr, _ = roc_curve(y_true, y_probs)
            auc_score = roc_auc_score(y_true, y_probs)
            print("ROC␣AUC␣Score:", round(auc_score, 4))

            # Plot ROC
            plt.figure(figsize=(6, 4))
            plt.plot(fpr, tpr, label=f"AUC␣=␣{auc_score:.4f}")
            plt.plot([0, 1], [0, 1], 'k--', label='Random␣Guess')
            plt.xlabel("False␣Positive␣Rate")
            plt.ylabel("True␣Positive␣Rate")
            plt.title(f"ROC␣Curve␣-␣{dataset_name}")
            plt.legend()
```

```
            plt.grid(True)
            plt.tight_layout()
            plt.show()
    else:
        n, p = X.shape
        r2 = r2_score(y_true, y_pred)
        adjusted_r2 = 1 - (1 - r2) * (n - 1) / (n - p - 1)
        print("Mean␣Squared␣Error:", mean_squared_error(y_true, y_pred)
            ↪ )
        print("Root␣Mean␣Squared␣Error:", root_mean_squared_error(
            ↪ y_true, y_pred))
        print("Mean␣Absolute␣Error:", mean_absolute_error(y_true,
            ↪ y_pred))
        print("R2␣Score:", r2)
        print("Adjusted␣R2␣Score:", adjusted_r2)

# For validation set
evaluate_model(y_val, y_val_pred, is_classification, X_val, model, "
    ↪ Validation␣Set")

# For test set
evaluate_model(y_test, y_test_pred, is_classification, X_test, model, "
    ↪ Test␣Set")
# Evaluating Model on Test and Validation Sets (Without Performance
    ↪ Metrics)

def plot_actual_vs_predicted(y_true, y_pred, title):
    plt.figure(figsize=(6, 4))
    plt.scatter(y_true, y_pred, alpha=0.5, edgecolor='k')
    plt.plot([y_true.min(), y_true.max()], [y_true.min(), y_true.max()
        ↪ ], 'r--')
    plt.xlabel("Actual")
    plt.ylabel("Predicted")
    plt.title(f"Actual␣vs␣Predicted␣-␣{title}")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

def plot_residuals(y_true, y_pred, title):
    residuals = y_true - y_pred
    plt.figure(figsize=(6, 4))
    plt.scatter(y_pred, residuals, alpha=0.5, edgecolor='k')
    plt.axhline(y=0, color='red', linestyle='--')
    plt.xlabel("Predicted")
    plt.ylabel("Residuals")
    plt.title(f"Residual␣Plot␣-␣{title}")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

def plot_residual_distribution(y_true, y_pred, title):
    residuals = y_true - y_pred
    plt.figure(figsize=(6, 4))
    sns.histplot(residuals, kde=True, color='skyblue')
    plt.title(f"Residual␣Distribution␣-␣{title}")
    plt.xlabel("Residuals")
    plt.ylabel("Frequency")
    plt.grid(True)
```

```
    plt.tight_layout()
    plt.show()

def plot_confusion_matrix(y_true, y_pred, title):
    ConfusionMatrixDisplay.from_predictions(y_true, y_pred)
    plt.title(title)
    plt.show()

if not is_classification:
  plot_actual_vs_predicted(y_val, y_val_pred, "Validation␣Set")
  plot_residuals(y_val, y_val_pred, "Validation␣Set")
  plot_residual_distribution(y_val, y_val_pred, "Validation␣Set")

  plot_actual_vs_predicted(y_test, y_test_pred, "Test␣Set")
  plot_residuals(y_test, y_test_pred, "Test␣Set")
  plot_residual_distribution(y_test, y_test_pred, "Test␣Set")
else:
  plot_confusion_matrix(y_test, y_test_pred, "Test␣Set")
  plot_confusion_matrix(y_val, y_val_pred, "Validation␣Set")
# (vii) K-Fold Cross Validation

kfold = KFold(n_splits=5, shuffle=True, random_state=42)
if is_classification:
    score = 'accuracy'
else:
    score = 'r2'
cv_results = cross_val_score(model, X, y, cv=kfold, scoring=score)

print("Cross␣Validation␣Scores:", cv_results)
print("Average␣CV␣Score:", np.mean(cv_results))
```

Listing 4: kNN

```
# (i) Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, KFold,
   ↪ cross_val_score
from sklearn.preprocessing import MinMaxScaler, StandardScaler,
   ↪ LabelEncoder, OneHotEncoder
from sklearn.metrics import mean_squared_error, root_mean_squared_error
   ↪ , mean_absolute_error, r2_score, accuracy_score, precision_score,
   ↪  recall_score, f1_score, classification_report, confusion_matrix,
   ↪  roc_auc_score, roc_curve, ConfusionMatrixDisplay
from sklearn.neighbors import KNeighborsClassifier

import time

# (ii) Import dataset
df = pd.read_csv('spambase_csv.xls')
target = 'class'
# (iii) EDA and Preprocessing

def is_normal(series):
    skew = series.skew()
```

```
    return -0.5 <= skew <= 0.5

def has_outliers(series):
    Q1 = series.quantile(0.25)
    Q3 = series.quantile(0.75)
    IQR = Q3 - Q1
    outliers = ((series < (Q1 - 1.5 * IQR)) | (series > (Q3 + 1.5 * IQR))).sum()
    return outliers > 0
# Separate types
numerical_cols = df.select_dtypes(include=['int64', 'float64']).columns.tolist()
categorical_cols = df.select_dtypes(include=['object', 'category']).columns.tolist()
numerical_cols = [col for col in numerical_cols if col != target]

# --- Missing Values Handling ---
for col in df.columns:
    if df[col].isnull().sum() > 0:
        if col in numerical_cols:
            if is_normal(df[col]):
                if has_outliers(df[col]):
                    df[col].fillna(df[col].median(), inplace=True)
                else:
                    df[col].fillna(df[col].mean(), inplace=True)
            else:
                df[col].fillna(df[col].median(), inplace=True)
        else:
            df[col].fillna(df[col].mode()[0], inplace=True)  # categorical

# --- Outlier Removal (for numerical columns only) ---
for col in numerical_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    #df = df[~((df[col] < Q1 - 1.5 * IQR) | (df[col] > Q3 + 1.5 * IQR))]
# --- Drop rows where target is missing ---
df.dropna(subset=[target], inplace=True)
# --- Encoding categorical features ---
is_classification = True

if is_classification:
    le = LabelEncoder()
    df[target] = le.fit_transform(df[target])  # encode target
    for col in categorical_cols:
        df = pd.get_dummies(df, columns=categorical_cols, drop_first=False)  # Similar to one-hot encoding
else:
    # Regression: target guided ordinal encoding
    for col in categorical_cols:
        ordered_labels = df.groupby(col)[target].mean().sort_values().index
        mapping = {k: i for i, k in enumerate(ordered_labels)}
        df[col] = df[col].map(mapping)
# --- Scaling ---
for col in numerical_cols:
```

```
    if is_normal(df[col]):
        scaler = StandardScaler()
    elif has_outliers(df[col]):
        scaler = StandardScaler()
    else:
        scaler = MinMaxScaler()

    df[[col]] = scaler.fit_transform(df[[col]])
# --- Histogram Subplots ---
n_cols = 5  # Number of plots per row
n_rows = int(np.ceil(len(numerical_cols) / n_cols))

fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 4 * n_rows))
axes = axes.flatten()

for i, col in enumerate(numerical_cols):
    df[col].hist(ax=axes[i], bins=30)
    axes[i].set_title(col)

# Turn off unused subplots
for j in range(i+1, len(axes)):
    axes[j].axis('off')

fig.suptitle("Histogram of Features", fontsize=20)
plt.tight_layout()
plt.show()

# --- Boxplot Subplots ---
fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 4 * n_rows))
axes = axes.flatten()

for i, col in enumerate(numerical_cols):
    axes[i].boxplot(df[col])
    axes[i].set_title(col)

for j in range(i+1, len(axes)):
    axes[j].axis('off')

fig.suptitle("Boxplot for Outlier Detection", fontsize=20)
plt.tight_layout()
plt.show()

# --- Correlation Heatmap ---
plt.figure(figsize=(20, 16))
sns.heatmap(df.corr(), annot=True, fmt=".2f", cmap='coolwarm',
    ↪ linewidths=0.5)
plt.title("Feature Correlation Heatmap", fontsize=18)
plt.xticks(rotation=45, ha='right', fontsize=8)
plt.yticks(fontsize=8)
plt.tight_layout()
plt.show()

# (iv) Splitting dataset

X = df.drop(columns=[target])
y = df[target]

# Splitting: Train (60%), Validation (20%), Test (20%)
```

```python
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4, 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)


print(f'Train: {X_train.shape}, Validation: {X_val.shape}, Test: {X_test.shape}')

df.to_csv('updated_spam.csv')
# (v) Model Training

model = KNeighborsClassifier(n_neighbors=5, metric='minkowski')

model.fit(X_train, y_train)

# Predictions
y_val_pred = model.predict(X_val)
y_test_pred = model.predict(X_test)
# (vi) Evaluation

# Evaluating Model using Performance Metrics

def evaluate_model(y_true, y_pred, is_classification, X, model, dataset_name):
    print(f"\n Evaluation - {dataset_name}")
    if is_classification:
        print("Accuracy :", round(accuracy_score(y_true, y_pred), 4))
        print("Precision:", round(precision_score(y_true, y_pred, average='weighted'), 4))
        print("Recall   :", round(recall_score(y_true, y_pred, average='weighted'), 4))
        print("F1 Score :", round(f1_score(y_true, y_pred, average='weighted'), 4))
        print("\nClassification Report:\n", classification_report(y_true, y_pred))

        # ROC Curve: Only for binary classification
        if len(np.unique(y_true)) == 2 and model is not None and hasattr(model, "predict_proba"):
            y_probs = model.predict_proba(X)[:, 1]
            fpr, tpr, _ = roc_curve(y_true, y_probs)
            auc_score = roc_auc_score(y_true, y_probs)
            print("ROC AUC Score:", round(auc_score, 4))

            # Plot ROC
            plt.figure(figsize=(6, 4))
            plt.plot(fpr, tpr, label=f"AUC = {auc_score:.4f}")
            plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
            plt.xlabel("False Positive Rate")
            plt.ylabel("True Positive Rate")
            plt.title(f"ROC Curve - {dataset_name}")
            plt.legend()
            plt.grid(True)
            plt.tight_layout()
            plt.show()
    else:
        n, p = X.shape
```

```
        r2 = r2_score(y_true, y_pred)
        adjusted_r2 = 1 - (1 - r2) * (n - 1) / (n - p - 1)
        print("Mean Squared Error:", mean_squared_error(y_true, y_pred)
            ↪ )
        print("Root Mean Squared Error:", root_mean_squared_error(
            ↪ y_true, y_pred))
        print("Mean Absolute Error:", mean_absolute_error(y_true,
            ↪ y_pred))
        print("R2 Score:", r2)
        print("Adjusted R2 Score:", adjusted_r2)

# For validation set
evaluate_model(y_val, y_val_pred, is_classification, X_val, model, "
    ↪ Validation Set")

# For test set
evaluate_model(y_test, y_test_pred, is_classification, X_test, model, "
    ↪ Test Set")
# Evaluating Model on Test and Validation Sets (Without Performance
    ↪ Metrics)

def plot_actual_vs_predicted(y_true, y_pred, title):
    plt.figure(figsize=(6, 4))
    plt.scatter(y_true, y_pred, alpha=0.5, edgecolor='k')
    plt.plot([y_true.min(), y_true.max()], [y_true.min(), y_true.max()
        ↪ ], 'r--')
    plt.xlabel("Actual")
    plt.ylabel("Predicted")
    plt.title(f"Actual vs Predicted - {title}")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

def plot_residuals(y_true, y_pred, title):
    residuals = y_true - y_pred
    plt.figure(figsize=(6, 4))
    plt.scatter(y_pred, residuals, alpha=0.5, edgecolor='k')
    plt.axhline(y=0, color='red', linestyle='--')
    plt.xlabel("Predicted")
    plt.ylabel("Residuals")
    plt.title(f"Residual Plot - {title}")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

def plot_residual_distribution(y_true, y_pred, title):
    residuals = y_true - y_pred
    plt.figure(figsize=(6, 4))
    sns.histplot(residuals, kde=True, color='skyblue')
    plt.title(f"Residual Distribution - {title}")
    plt.xlabel("Residuals")
    plt.ylabel("Frequency")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

def plot_confusion_matrix(y_true, y_pred, title):
    ConfusionMatrixDisplay.from_predictions(y_true, y_pred)
```

```
    plt.title(title)
    plt.show()

if not is_classification:
  plot_actual_vs_predicted(y_val, y_val_pred, "Validation Set")
  plot_residuals(y_val, y_val_pred, "Validation Set")
  plot_residual_distribution(y_val, y_val_pred, "Validation Set")

  plot_actual_vs_predicted(y_test, y_test_pred, "Test Set")
  plot_residuals(y_test, y_test_pred, "Test Set")
  plot_residual_distribution(y_test, y_test_pred, "Test Set")
else:
  plot_confusion_matrix(y_test, y_test_pred, "Test Set")
  plot_confusion_matrix(y_val, y_val_pred, "Validation Set")
# (vii) K-Fold Cross Validation

kfold = KFold(n_splits=5, shuffle=True, random_state=42)
if is_classification:
    score = 'accuracy'
else:
    score = 'r2'
cv_results = cross_val_score(model, X, y, cv=kfold, scoring=score)

print("Cross Validation Scores:", cv_results)
print("Average CV Score:", np.mean(cv_results))
results = []

# --- KNN with different k values ---
for k in [1, 3, 5, 7]:
    start = time.time()
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    train_time = time.time() - start

    results.append({
        'Model': f'KNN (k={k})',
        'Accuracy': accuracy_score(y_test, y_pred),
        'Precision': precision_score(y_test, y_pred, average='weighted'
           ↪ ),
        'Recall': recall_score(y_test, y_pred, average='weighted'),
        'F1-Score': f1_score(y_test, y_pred, average='weighted'),
        'Train Time (s)': round(train_time, 4)
    })


# --- KDTree (k=5) ---
start = time.time()
model_kdtree = KNeighborsClassifier(n_neighbors=5, algorithm='kd_tree')
model_kdtree.fit(X_train, y_train)
y_pred_kdtree = model_kdtree.predict(X_test)
train_time = time.time() - start

results.append({
    'Model': 'KDTree (k=5)',
    'Accuracy': accuracy_score(y_test, y_pred_kdtree),
    'Precision': precision_score(y_test, y_pred_kdtree, average='
       ↪ weighted'),
```

```
    'Recall': recall_score(y_test, y_pred_kdtree, average='weighted'),
    'F1-Score': f1_score(y_test, y_pred_kdtree, average='weighted'),
    'Train␣Time␣(s)': round(train_time, 4)
})


# --- BallTree (k=5) ---
start = time.time()
model_balltree = KNeighborsClassifier(n_neighbors=5, algorithm='
   ↪ ball_tree')
model_balltree.fit(X_train, y_train)
y_pred_balltree = model_balltree.predict(X_test)
train_time = time.time() - start

results.append({
    'Model': 'BallTree␣(k=5)',
    'Accuracy': accuracy_score(y_test, y_pred_balltree),
    'Precision': precision_score(y_test, y_pred_balltree, average='
       ↪ weighted'),
    'Recall': recall_score(y_test, y_pred_balltree, average='weighted')
       ↪ ,
    'F1-Score': f1_score(y_test, y_pred_balltree, average='weighted'),
    'Train␣Time␣(s)': round(train_time, 4)
})

# --- Display Results ---
df_results = pd.DataFrame(results)
print(df_results)
```

Listing 5: kDTree

```
# (i) Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, KFold,
   ↪ cross_val_score
from sklearn.preprocessing import MinMaxScaler, StandardScaler,
   ↪ LabelEncoder, OneHotEncoder
from sklearn.metrics import mean_squared_error, root_mean_squared_error
   ↪ , mean_absolute_error, r2_score, accuracy_score, precision_score,
   ↪  recall_score, f1_score, classification_report, confusion_matrix,
   ↪  roc_auc_score, roc_curve, ConfusionMatrixDisplay
from sklearn.neighbors import KNeighborsClassifier

import time

# (ii) Import dataset
df = pd.read_csv('spambase_csv.xls')
target = 'class'
# (iii) EDA and Preprocessing

def is_normal(series):
    skew = series.skew()
    return -0.5 <= skew <= 0.5
```

```
def has_outliers(series):
    Q1 = series.quantile(0.25)
    Q3 = series.quantile(0.75)
    IQR = Q3 - Q1
    outliers = ((series < (Q1 - 1.5 * IQR)) | (series > (Q3 + 1.5 * IQR
        ↪ ))).sum()
    return outliers > 0
# Separate types
numerical_cols = df.select_dtypes(include=['int64', 'float64']).columns
    ↪ .tolist()
categorical_cols = df.select_dtypes(include=['object', 'category']).
    ↪ columns.tolist()
numerical_cols = [col for col in numerical_cols if col != target]

# --- Missing Values Handling ---
for col in df.columns:
    if df[col].isnull().sum() > 0:
        if col in numerical_cols:
            if is_normal(df[col]):
                if has_outliers(df[col]):
                    df[col].fillna(df[col].median(), inplace=True)
                else:
                    df[col].fillna(df[col].mean(), inplace=True)
            else:
                df[col].fillna(df[col].median(), inplace=True)
        else:
            df[col].fillna(df[col].mode()[0], inplace=True)  #
                ↪ categorical

# --- Outlier Removal (for numerical columns only) ---
for col in numerical_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    #df = df[~((df[col] < Q1 - 1.5 * IQR) | (df[col] > Q3 + 1.5 * IQR))
        ↪ ]
# --- Drop rows where target is missing ---
df.dropna(subset=[target], inplace=True)
# --- Encoding categorical features ---
is_classification = True

if is_classification:
    le = LabelEncoder()
    df[target] = le.fit_transform(df[target])  # encode target
    for col in categorical_cols:
        df = pd.get_dummies(df, columns=categorical_cols, drop_first=
            ↪ False)  # Similar to one-hot encoding
else:
    # Regression: target guided ordinal encoding
    for col in categorical_cols:
        ordered_labels = df.groupby(col)[target].mean().sort_values().
            ↪ index
        mapping = {k: i for i, k in enumerate(ordered_labels)}
        df[col] = df[col].map(mapping)
# --- Scaling ---
for col in numerical_cols:
    if is_normal(df[col]):
        scaler = StandardScaler()
```

```
    elif has_outliers(df[col]):
        scaler = StandardScaler()
    else:
        scaler = MinMaxScaler()

    df[[col]] = scaler.fit_transform(df[[col]])
# --- Histogram Subplots ---
n_cols = 5  # Number of plots per row
n_rows = int(np.ceil(len(numerical_cols) / n_cols))

fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 4 * n_rows))
axes = axes.flatten()

for i, col in enumerate(numerical_cols):
    df[col].hist(ax=axes[i], bins=30)
    axes[i].set_title(col)

# Turn off unused subplots
for j in range(i+1, len(axes)):
    axes[j].axis('off')

fig.suptitle("Histogram␣of␣Features", fontsize=20)
plt.tight_layout()
plt.show()

# --- Boxplot Subplots ---
fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 4 * n_rows))
axes = axes.flatten()

for i, col in enumerate(numerical_cols):
    axes[i].boxplot(df[col])
    axes[i].set_title(col)

for j in range(i+1, len(axes)):
    axes[j].axis('off')

fig.suptitle("Boxplot␣for␣Outlier␣Detection", fontsize=20)
plt.tight_layout()
plt.show()

# --- Correlation Heatmap ---
plt.figure(figsize=(20, 16))
sns.heatmap(df.corr(), annot=True, fmt=".2f", cmap='coolwarm',
   ↪ linewidths=0.5)
plt.title("Feature␣Correlation␣Heatmap", fontsize=18)
plt.xticks(rotation=45, ha='right', fontsize=8)
plt.yticks(fontsize=8)
plt.tight_layout()
plt.show()

# (iv) Splitting dataset

X = df.drop(columns=[target])
y = df[target]

# Splitting: Train (60%), Validation (20%), Test (20%)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size
   ↪ =0.4, 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)


print(f'Train: {X_train.shape}, Validation: {X_val.shape}, Test: {X_test.shape}')

df.to_csv('updated_spam.csv')
# (v) Model Training

model = KNeighborsClassifier(n_neighbors=5, algorithm='kd_tree', metric='minkowski')

model.fit(X_train, y_train)

# Predictions
y_val_pred = model.predict(X_val)
y_test_pred = model.predict(X_test)
# (vi) Evaluation

# Evaluating Model using Performance Metrics

def evaluate_model(y_true, y_pred, is_classification, X, model, dataset_name):
    print(f"\n Evaluation - {dataset_name}")
    if is_classification:
        print("Accuracy :", round(accuracy_score(y_true, y_pred), 4))
        print("Precision:", round(precision_score(y_true, y_pred, average='weighted'), 4))
        print("Recall   :", round(recall_score(y_true, y_pred, average='weighted'), 4))
        print("F1 Score :", round(f1_score(y_true, y_pred, average='weighted'), 4))
        print("\nClassification Report:\n", classification_report(y_true, y_pred))

        # ROC Curve: Only for binary classification
        if len(np.unique(y_true)) == 2 and model is not None and hasattr(model, "predict_proba"):
            y_probs = model.predict_proba(X)[:, 1]
            fpr, tpr, _ = roc_curve(y_true, y_probs)
            auc_score = roc_auc_score(y_true, y_probs)
            print("ROC AUC Score:", round(auc_score, 4))

            # Plot ROC
            plt.figure(figsize=(6, 4))
            plt.plot(fpr, tpr, label=f"AUC = {auc_score:.4f}")
            plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
            plt.xlabel("False Positive Rate")
            plt.ylabel("True Positive Rate")
            plt.title(f"ROC Curve - {dataset_name}")
            plt.legend()
            plt.grid(True)
            plt.tight_layout()
            plt.show()
    else:
        n, p = X.shape
        r2 = r2_score(y_true, y_pred)
```

```
        adjusted_r2 = 1 - (1 - r2) * (n - 1) / (n - p - 1)
        print("Mean␣Squared␣Error:", mean_squared_error(y_true, y_pred)
           ↪ )
        print("Root␣Mean␣Squared␣Error:", root_mean_squared_error(
           ↪ y_true, y_pred))
        print("Mean␣Absolute␣Error:", mean_absolute_error(y_true,
           ↪ y_pred))
        print("R2␣Score:", r2)
        print("Adjusted␣R2␣Score:", adjusted_r2)

# For validation set
evaluate_model(y_val, y_val_pred, is_classification, X_val, model, "
   ↪ Validation␣Set")

# For test set
evaluate_model(y_test, y_test_pred, is_classification, X_test, model, "
   ↪ Test␣Set")
# Evaluating Model on Test and Validation Sets (Without Performance
   ↪ Metrics)

def plot_actual_vs_predicted(y_true, y_pred, title):
    plt.figure(figsize=(6, 4))
    plt.scatter(y_true, y_pred, alpha=0.5, edgecolor='k')
    plt.plot([y_true.min(), y_true.max()], [y_true.min(), y_true.max()
       ↪ ], 'r--')
    plt.xlabel("Actual")
    plt.ylabel("Predicted")
    plt.title(f"Actual␣vs␣Predicted␣-␣{title}")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

def plot_residuals(y_true, y_pred, title):
    residuals = y_true - y_pred
    plt.figure(figsize=(6, 4))
    plt.scatter(y_pred, residuals, alpha=0.5, edgecolor='k')
    plt.axhline(y=0, color='red', linestyle='--')
    plt.xlabel("Predicted")
    plt.ylabel("Residuals")
    plt.title(f"Residual␣Plot␣-␣{title}")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

def plot_residual_distribution(y_true, y_pred, title):
    residuals = y_true - y_pred
    plt.figure(figsize=(6, 4))
    sns.histplot(residuals, kde=True, color='skyblue')
    plt.title(f"Residual␣Distribution␣-␣{title}")
    plt.xlabel("Residuals")
    plt.ylabel("Frequency")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

def plot_confusion_matrix(y_true, y_pred, title):
    ConfusionMatrixDisplay.from_predictions(y_true, y_pred)
    plt.title(title)
```

```
    plt.show()

if not is_classification:
  plot_actual_vs_predicted(y_val, y_val_pred, "Validation Set")
  plot_residuals(y_val, y_val_pred, "Validation Set")
  plot_residual_distribution(y_val, y_val_pred, "Validation Set")

  plot_actual_vs_predicted(y_test, y_test_pred, "Test Set")
  plot_residuals(y_test, y_test_pred, "Test Set")
  plot_residual_distribution(y_test, y_test_pred, "Test Set")
else:
  plot_confusion_matrix(y_test, y_test_pred, "Test Set")
  plot_confusion_matrix(y_val, y_val_pred, "Validation Set")
# (vii) K-Fold Cross Validation

kfold = KFold(n_splits=5, shuffle=True, random_state=42)
if is_classification:
    score = 'accuracy'
else:
    score = 'r2'
cv_results = cross_val_score(model, X, y, cv=kfold, scoring=score)

print("Cross Validation Scores:", cv_results)
print("Average CV Score:", np.mean(cv_results))
```

Listing 6: BallTree

```
# (i) Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, KFold,
    ↪ cross_val_score
from sklearn.preprocessing import MinMaxScaler, StandardScaler,
    ↪ LabelEncoder, OneHotEncoder
from sklearn.metrics import mean_squared_error, root_mean_squared_error
    ↪ , mean_absolute_error, r2_score, accuracy_score, precision_score,
    ↪  recall_score, f1_score, classification_report, confusion_matrix,
    ↪  roc_auc_score, roc_curve, ConfusionMatrixDisplay
from sklearn.neighbors import KNeighborsClassifier

import time

# (ii) Import dataset
df = pd.read_csv('spambase_csv.xls')
target = 'class'
# (iii) EDA and Preprocessing

def is_normal(series):
    skew = series.skew()
    return -0.5 <= skew <= 0.5

def has_outliers(series):
    Q1 = series.quantile(0.25)
    Q3 = series.quantile(0.75)
    IQR = Q3 - Q1
```

```
    outliers = ((series < (Q1 - 1.5 * IQR)) | (series > (Q3 + 1.5 * IQR
        ↪ ))).sum()
    return outliers > 0
# Separate types
numerical_cols = df.select_dtypes(include=['int64', 'float64']).columns
    ↪ .tolist()
categorical_cols = df.select_dtypes(include=['object', 'category']).
    ↪ columns.tolist()
numerical_cols = [col for col in numerical_cols if col != target]

# --- Missing Values Handling ---
for col in df.columns:
    if df[col].isnull().sum() > 0:
        if col in numerical_cols:
            if is_normal(df[col]):
                if has_outliers(df[col]):
                    df[col].fillna(df[col].median(), inplace=True)
                else:
                    df[col].fillna(df[col].mean(), inplace=True)
            else:
                df[col].fillna(df[col].median(), inplace=True)
        else:
            df[col].fillna(df[col].mode()[0], inplace=True)  #
                ↪ categorical

# --- Outlier Removal (for numerical columns only) ---
for col in numerical_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    #df = df[~((df[col] < Q1 - 1.5 * IQR) | (df[col] > Q3 + 1.5 * IQR))
        ↪ ]
# --- Drop rows where target is missing ---
df.dropna(subset=[target], inplace=True)
# --- Encoding categorical features ---
is_classification = True

if is_classification:
    le = LabelEncoder()
    df[target] = le.fit_transform(df[target])  # encode target
    for col in categorical_cols:
        df = pd.get_dummies(df, columns=categorical_cols, drop_first=
            ↪ False)  # Similar to one-hot encoding
else:
    # Regression: target guided ordinal encoding
    for col in categorical_cols:
        ordered_labels = df.groupby(col)[target].mean().sort_values().
            ↪ index
        mapping = {k: i for i, k in enumerate(ordered_labels)}
        df[col] = df[col].map(mapping)
# --- Scaling ---
for col in numerical_cols:
    if is_normal(df[col]):
        scaler = StandardScaler()
    elif has_outliers(df[col]):
        scaler = StandardScaler()
    else:
        scaler = MinMaxScaler()
```

```
    df[[col]] = scaler.fit_transform(df[[col]])
# --- Histogram Subplots ---
n_cols = 5  # Number of plots per row
n_rows = int(np.ceil(len(numerical_cols) / n_cols))

fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 4 * n_rows))
axes = axes.flatten()

for i, col in enumerate(numerical_cols):
    df[col].hist(ax=axes[i], bins=30)
    axes[i].set_title(col)

# Turn off unused subplots
for j in range(i+1, len(axes)):
    axes[j].axis('off')

fig.suptitle("Histogram␣of␣Features", fontsize=20)
plt.tight_layout()
plt.show()

# --- Boxplot Subplots ---
fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 4 * n_rows))
axes = axes.flatten()

for i, col in enumerate(numerical_cols):
    axes[i].boxplot(df[col])
    axes[i].set_title(col)

for j in range(i+1, len(axes)):
    axes[j].axis('off')

fig.suptitle("Boxplot␣for␣Outlier␣Detection", fontsize=20)
plt.tight_layout()
plt.show()

# --- Correlation Heatmap ---
plt.figure(figsize=(20, 16))
sns.heatmap(df.corr(), annot=True, fmt=".2f", cmap='coolwarm',
   ↪ linewidths=0.5)
plt.title("Feature␣Correlation␣Heatmap", fontsize=18)
plt.xticks(rotation=45, ha='right', fontsize=8)
plt.yticks(fontsize=8)
plt.tight_layout()
plt.show()

# (iv) Splitting dataset

X = df.drop(columns=[target])
y = df[target]

# Splitting: Train (60%), Validation (20%), Test (20%)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size
   ↪ =0.4, 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)
```

```
print(f'Train:␣{X_train.shape},␣Validation:␣{X_val.shape},␣Test:␣{
    ↪ X_test.shape}')

df.to_csv('updated_spam.csv')
# (v) Model Training

model = KNeighborsClassifier(n_neighbors=5, algorithm='ball_tree',
    ↪ metric='minkowski')

model.fit(X_train, y_train)

# Predictions
y_val_pred = model.predict(X_val)
y_test_pred = model.predict(X_test)
# (vi) Evaluation

# Evaluating Model using Performance Metrics

def evaluate_model(y_true, y_pred, is_classification, X, model,
    ↪ dataset_name):
    print(f"\n␣Evaluation␣-␣{dataset_name}")
    if is_classification:
        print("Accuracy␣:", round(accuracy_score(y_true, y_pred), 4))
        print("Precision:", round(precision_score(y_true, y_pred,
            ↪ average='weighted'), 4))
        print("Recall␣␣␣:", round(recall_score(y_true, y_pred, average=
            ↪ 'weighted'), 4))
        print("F1␣Score␣:", round(f1_score(y_true, y_pred, average='
            ↪ weighted'), 4))
        print("\nClassification␣Report:\n", classification_report(
            ↪ y_true, y_pred))

        # ROC Curve: Only for binary classification
        if len(np.unique(y_true)) == 2 and model is not None and
            ↪ hasattr(model, "predict_proba"):
            y_probs = model.predict_proba(X)[:, 1]
            fpr, tpr, _ = roc_curve(y_true, y_probs)
            auc_score = roc_auc_score(y_true, y_probs)
            print("ROC␣AUC␣Score:", round(auc_score, 4))

            # Plot ROC
            plt.figure(figsize=(6, 4))
            plt.plot(fpr, tpr, label=f"AUC␣=␣{auc_score:.4f}")
            plt.plot([0, 1], [0, 1], 'k--', label='Random␣Guess')
            plt.xlabel("False␣Positive␣Rate")
            plt.ylabel("True␣Positive␣Rate")
            plt.title(f"ROC␣Curve␣-␣{dataset_name}")
            plt.legend()
            plt.grid(True)
            plt.tight_layout()
            plt.show()
    else:
        n, p = X.shape
        r2 = r2_score(y_true, y_pred)
        adjusted_r2 = 1 - (1 - r2) * (n - 1) / (n - p - 1)
        print("Mean␣Squared␣Error:", mean_squared_error(y_true, y_pred)
            ↪ )
        print("Root␣Mean␣Squared␣Error:", root_mean_squared_error(
```

```
            ↪ y_true, y_pred))
        print("Mean␣Absolute␣Error:", mean_absolute_error(y_true,
            ↪ y_pred))
        print("R2␣Score:", r2)
        print("Adjusted␣R2␣Score:", adjusted_r2)

# For validation set
evaluate_model(y_val, y_val_pred, is_classification, X_val, model, "
    ↪ Validation␣Set")

# For test set
evaluate_model(y_test, y_test_pred, is_classification, X_test, model, "
    ↪ Test␣Set")
# Evaluating Model on Test and Validation Sets (Without Performance
    ↪ Metrics)

def plot_actual_vs_predicted(y_true, y_pred, title):
    plt.figure(figsize=(6, 4))
    plt.scatter(y_true, y_pred, alpha=0.5, edgecolor='k')
    plt.plot([y_true.min(), y_true.max()], [y_true.min(), y_true.max()
        ↪ ], 'r--')
    plt.xlabel("Actual")
    plt.ylabel("Predicted")
    plt.title(f"Actual␣vs␣Predicted␣-␣{title}")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

def plot_residuals(y_true, y_pred, title):
    residuals = y_true - y_pred
    plt.figure(figsize=(6, 4))
    plt.scatter(y_pred, residuals, alpha=0.5, edgecolor='k')
    plt.axhline(y=0, color='red', linestyle='--')
    plt.xlabel("Predicted")
    plt.ylabel("Residuals")
    plt.title(f"Residual␣Plot␣-␣{title}")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

def plot_residual_distribution(y_true, y_pred, title):
    residuals = y_true - y_pred
    plt.figure(figsize=(6, 4))
    sns.histplot(residuals, kde=True, color='skyblue')
    plt.title(f"Residual␣Distribution␣-␣{title}")
    plt.xlabel("Residuals")
    plt.ylabel("Frequency")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

def plot_confusion_matrix(y_true, y_pred, title):
    ConfusionMatrixDisplay.from_predictions(y_true, y_pred)
    plt.title(title)
    plt.show()

if not is_classification:
  plot_actual_vs_predicted(y_val, y_val_pred, "Validation␣Set")
```

```
  plot_residuals(y_val, y_val_pred, "Validation␣Set")
  plot_residual_distribution(y_val, y_val_pred, "Validation␣Set")

  plot_actual_vs_predicted(y_test, y_test_pred, "Test␣Set")
  plot_residuals(y_test, y_test_pred, "Test␣Set")
  plot_residual_distribution(y_test, y_test_pred, "Test␣Set")
else:
  plot_confusion_matrix(y_test, y_test_pred, "Test␣Set")
  plot_confusion_matrix(y_val, y_val_pred, "Validation␣Set")
# (vii) K-Fold Cross Validation

kfold = KFold(n_splits=5, shuffle=True, random_state=42)
if is_classification:
    score = 'accuracy'
else:
    score = 'r2'
cv_results = cross_val_score(model, X, y, cv=kfold, scoring=score)

print("Cross␣Validation␣Scores:", cv_results)
print("Average␣CV␣Score:", np.mean(cv_results))
```

Listing 7: SVM (SVC all 4)

```
# (i) Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, KFold,
   ↪ cross_val_score, GridSearchCV
from sklearn.preprocessing import MinMaxScaler, StandardScaler,
   ↪ LabelEncoder, OneHotEncoder, Binarizer
from sklearn.metrics import mean_squared_error, root_mean_squared_error
   ↪ , mean_absolute_error, r2_score, accuracy_score, precision_score,
   ↪  recall_score, f1_score, classification_report, confusion_matrix,
   ↪  roc_auc_score, roc_curve, ConfusionMatrixDisplay
from sklearn.svm import SVC

# (ii) Import dataset
df = pd.read_csv('spambase_csv.xls')
target = 'class'
# (iii) EDA and Preprocessing

def is_normal(series):
    skew = series.skew()
    return -0.5 <= skew <= 0.5

def has_outliers(series):
    Q1 = series.quantile(0.25)
    Q3 = series.quantile(0.75)
    IQR = Q3 - Q1
    outliers = ((series < (Q1 - 1.5 * IQR)) | (series > (Q3 + 1.5 * IQR
       ↪ ))).sum()
    return outliers > 0
# Separate types
numerical_cols = df.select_dtypes(include=['int64', 'float64']).columns
   ↪ .tolist()
```

```python
categorical_cols = df.select_dtypes(include=['object', 'category']).
    ↪ columns.tolist()
numerical_cols = [col for col in numerical_cols if col != target]

# --- Missing Values Handling ---
for col in df.columns:
    if df[col].isnull().sum() > 0:
        if col in numerical_cols:
            if is_normal(df[col]):
                if has_outliers(df[col]):
                    df[col].fillna(df[col].median(), inplace=True)
                else:
                    df[col].fillna(df[col].mean(), inplace=True)
            else:
                df[col].fillna(df[col].median(), inplace=True)
        else:
            df[col].fillna(df[col].mode()[0], inplace=True)  #
                ↪ categorical

# --- Outlier Removal (for numerical columns only) ---
for col in numerical_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    #df = df[~((df[col] < Q1 - 1.5 * IQR) | (df[col] > Q3 + 1.5 * IQR))
        ↪ ]
# --- Drop rows where target is missing ---
df.dropna(subset=[target], inplace=True)
# --- Encoding categorical features ---
is_classification = True

if is_classification:
    le = LabelEncoder()
    df[target] = le.fit_transform(df[target])  # encode target
    for col in categorical_cols:
        df[col] = le.fit_transform(df[col])  # label encoding for
            ↪ classification
else:
    # Regression: target guided ordinal encoding
    for col in categorical_cols:
        ordered_labels = df.groupby(col)[target].mean().sort_values().
            ↪ index
        mapping = {k: i for i, k in enumerate(ordered_labels)}
        df[col] = df[col].map(mapping)
for col in numerical_cols:
    if is_normal(df[col]):
        scaler = StandardScaler()
    elif has_outliers(df[col]):
        scaler = StandardScaler()
    else:
        scaler = MinMaxScaler()

    df[[col]] = scaler.fit_transform(df[[col]])
# --- Histogram Subplots ---
n_cols = 5  # Number of plots per row
n_rows = int(np.ceil(len(numerical_cols) / n_cols))

fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 4 * n_rows))
```

```
axes = axes.flatten()

for i, col in enumerate(numerical_cols):
    df[col].hist(ax=axes[i], bins=30)
    axes[i].set_title(col)

# Turn off unused subplots
for j in range(i+1, len(axes)):
    axes[j].axis('off')

fig.suptitle("Histogram of Features", fontsize=20)
plt.tight_layout()
plt.show()

# --- Boxplot Subplots ---
fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 4 * n_rows))
axes = axes.flatten()

for i, col in enumerate(numerical_cols):
    axes[i].boxplot(df[col])
    axes[i].set_title(col)

for j in range(i+1, len(axes)):
    axes[j].axis('off')

fig.suptitle("Boxplot for Outlier Detection", fontsize=20)
plt.tight_layout()
plt.show()

# --- Correlation Heatmap ---
plt.figure(figsize=(20, 16))
sns.heatmap(df.corr(), annot=False, fmt=".2f", cmap='coolwarm',
    ↪ linewidths=0.5)
plt.title("Feature Correlation Heatmap", fontsize=18)
plt.xticks(rotation=45, ha='right', fontsize=8)
plt.yticks(fontsize=8)
plt.tight_layout()
plt.show()

# (iv) Splitting dataset

X = df.drop(columns=[target])
y = df[target]

# Splitting: Train (60%), Validation (20%), Test (20%)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size
    ↪ =0.4, 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)


print(f'Train: {X_train.shape}, Validation: {X_val.shape}, Test: {
    ↪ X_test.shape}')

df.to_csv('updated_spam.csv')
# (v) Model Training

# ---------------- Linear Kernel ----------------
```

```
param_grid_l = {'C': [0.1, 1.0, 10]}
grid_l = GridSearchCV(SVC(kernel='linear'), param_grid_l, cv=5)
grid_l.fit(X_train, y_train)

model_l = grid_l.best_estimator_
y_val_pred_l = model_l.predict(X_val)
y_test_pred_l = model_l.predict(X_test)

# ---------------- Polynomial Kernel ----------------
param_grid_p = {
    'C': [0.1, 1.0, 10],
    'degree': [2, 3, 4],
    'gamma': ['scale', 'auto']
}
grid_p = GridSearchCV(SVC(kernel='poly'), param_grid_p, cv=5)
grid_p.fit(X_train, y_train)

model_p = grid_p.best_estimator_
y_val_pred_p = model_p.predict(X_val)
y_test_pred_p = model_p.predict(X_test)

# ---------------- RBF Kernel ----------------
param_grid_r = {
    'C': [0.1, 1.0, 10],
    'gamma': ['scale', 'auto']
}
grid_r = GridSearchCV(SVC(kernel='rbf'), param_grid_r, cv=5)
grid_r.fit(X_train, y_train)

model_r = grid_r.best_estimator_
y_val_pred_r = model_r.predict(X_val)
y_test_pred_r = model_r.predict(X_test)

# ---------------- Sigmoid Kernel ----------------
param_grid_s = {
    'C': [0.1, 1.0, 10],
    'gamma': ['scale', 'auto']
}
grid_s = GridSearchCV(SVC(kernel='sigmoid'), param_grid_s, cv=5)
grid_s.fit(X_train, y_train)

model_s = grid_s.best_estimator_
y_val_pred_s = model_s.predict(X_val)
y_test_pred_s = model_s.predict(X_test)
# (vi) Evaluation

# Evaluating Model using Performance Metrics

def evaluate_model(y_true, y_pred, is_classification, X, model,
   ↪ dataset_name):
    print(f"\n Evaluation - {dataset_name}")
    if is_classification:
        print("Accuracy :", round(accuracy_score(y_true, y_pred), 4))
        print("Precision:", round(precision_score(y_true, y_pred,
           ↪ average='weighted'), 4))
        print("Recall   :", round(recall_score(y_true, y_pred, average=
           ↪ 'weighted'), 4))
        print("F1 Score :", round(f1_score(y_true, y_pred, average='
```

```
            weighted'), 4))
        print("\nClassification Report:\n", classification_report(y_true, y_pred))

        # ROC Curve: Only for binary classification
        if len(np.unique(y_true)) == 2 and model is not None and hasattr(model, "predict_proba"):
            y_probs = model.predict_proba(X)[:, 1]
            fpr, tpr, _ = roc_curve(y_true, y_probs)
            auc_score = roc_auc_score(y_true, y_probs)
            print("ROC AUC Score:", round(auc_score, 4))

            # Plot ROC
            plt.figure(figsize=(6, 4))
            plt.plot(fpr, tpr, label=f"AUC = {auc_score:.4f}")
            plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
            plt.xlabel("False Positive Rate")
            plt.ylabel("True Positive Rate")
            plt.title(f"ROC Curve - {dataset_name}")
            plt.legend()
            plt.grid(True)
            plt.tight_layout()
            plt.show()
    else:
        n, p = X.shape
        r2 = r2_score(y_true, y_pred)
        adjusted_r2 = 1 - (1 - r2) * (n - 1) / (n - p - 1)
        print("Mean Squared Error:", mean_squared_error(y_true, y_pred))
        print("Root Mean Squared Error:", root_mean_squared_error(y_true, y_pred))
        print("Mean Absolute Error:", mean_absolute_error(y_true, y_pred))
        print("R2 Score:", r2)
        print("Adjusted R2 Score:", adjusted_r2)

evaluate_model(y_val, y_val_pred_l, True, X_val, model_l, "Validation Set - Linear SVM")
evaluate_model(y_test, y_test_pred_l, True, X_test, model_l, "Test Set - Linear SVM")

evaluate_model(y_val, y_val_pred_p, True, X_val, model_p, "Validation Set - Polynomial SVM")
evaluate_model(y_test, y_test_pred_p, True, X_test, model_p, "Test Set - Polynomial SVM")

evaluate_model(y_val, y_val_pred_r, True, X_val, model_r, "Validation Set - RBF SVM")
evaluate_model(y_test, y_test_pred_r, True, X_test, model_r, "Test Set - RBF SVM")

evaluate_model(y_val, y_val_pred_s, True, X_val, model_s, "Validation Set - Sigmoid SVM")
evaluate_model(y_test, y_test_pred_s, True, X_test, model_s, "Test Set - Sigmoid SVM")
# Evaluating Model on Test and Validation Sets (Without Performance Metrics)
```

```
def plot_actual_vs_predicted(y_true, y_pred, title):
    plt.figure(figsize=(6, 4))
    plt.scatter(y_true, y_pred, alpha=0.5, edgecolor='k')
    plt.plot([y_true.min(), y_true.max()], [y_true.min(), y_true.max()
        ↪ ], 'r--')
    plt.xlabel("Actual")
    plt.ylabel("Predicted")
    plt.title(f"Actual␣vs␣Predicted␣-␣{title}")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

def plot_residuals(y_true, y_pred, title):
    residuals = y_true - y_pred
    plt.figure(figsize=(6, 4))
    plt.scatter(y_pred, residuals, alpha=0.5, edgecolor='k')
    plt.axhline(y=0, color='red', linestyle='--')
    plt.xlabel("Predicted")
    plt.ylabel("Residuals")
    plt.title(f"Residual␣Plot␣-␣{title}")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

def plot_residual_distribution(y_true, y_pred, title):
    residuals = y_true - y_pred
    plt.figure(figsize=(6, 4))
    sns.histplot(residuals, kde=True, color='skyblue')
    plt.title(f"Residual␣Distribution␣-␣{title}")
    plt.xlabel("Residuals")
    plt.ylabel("Frequency")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

def plot_confusion_matrix(y_true, y_pred, title):
    ConfusionMatrixDisplay.from_predictions(y_true, y_pred)
    plt.title(title)
    plt.show()

if not is_classification:
  plot_actual_vs_predicted(y_val, y_val_pred, "Validation␣Set")
  plot_residuals(y_val, y_val_pred, "Validation␣Set")
  plot_residual_distribution(y_val, y_val_pred, "Validation␣Set")

  plot_actual_vs_predicted(y_test, y_test_pred, "Test␣Set")
  plot_residuals(y_test, y_test_pred, "Test␣Set")
  plot_residual_distribution(y_test, y_test_pred, "Test␣Set")
else:
  # --- Linear SVM ---
  plot_confusion_matrix(y_val, y_val_pred_l, "Validation␣Set␣-␣Linear␣
     ↪ SVM")
  plot_confusion_matrix(y_test, y_test_pred_l, "Test␣Set␣-␣Linear␣SVM")

  # --- Polynomial SVM ---
  plot_confusion_matrix(y_val, y_val_pred_p, "Validation␣Set␣-␣
     ↪ Polynomial␣SVM")
  plot_confusion_matrix(y_test, y_test_pred_p, "Test␣Set␣-␣Polynomial␣
```

```
      ↪ SVM")

  # --- RBF SVM ---
  plot_confusion_matrix(y_val, y_val_pred_r, "Validation␣Set␣-␣RBF␣SVM"
      ↪ )
  plot_confusion_matrix(y_test, y_test_pred_r, "Test␣Set␣-␣RBF␣SVM")

  # --- Sigmoid SVM ---
  plot_confusion_matrix(y_val, y_val_pred_s, "Validation␣Set␣-␣Sigmoid␣
      ↪ SVM")
  plot_confusion_matrix(y_test, y_test_pred_s, "Test␣Set␣-␣Sigmoid␣SVM"
      ↪ )
# (vii) K-Fold Cross Validation

kfold = KFold(n_splits=5, shuffle=True, random_state=42)
if is_classification:
    score = 'accuracy'
else:
    score = 'r2'
# --- Linear SVM ---
cv_l = cross_val_score(model_l, X, y, cv=kfold, scoring=score)
print("Linear␣SVM␣-␣CV␣Scores:", cv_l)
print("Linear␣SVM␣-␣Average␣CV␣Score:", np.mean(cv_l))

# --- Polynomial SVM ---
cv_p = cross_val_score(model_p, X, y, cv=kfold, scoring=score)
print("Polynomial␣SVM␣-␣CV␣Scores:", cv_p)
print("Polynomial␣SVM␣-␣Average␣CV␣Score:", np.mean(cv_p))

# --- RBF SVM ---
cv_r = cross_val_score(model_r, X, y, cv=kfold, scoring=score)
print("RBF␣SVM␣-␣CV␣Scores:", cv_r)
print("RBF␣SVM␣-␣Average␣CV␣Score:", np.mean(cv_r))

# --- Sigmoid SVM ---
cv_s = cross_val_score(model_s, X, y, cv=kfold, scoring=score)
print("Sigmoid␣SVM␣-␣CV␣Scores:", cv_s)
print("Sigmoid␣SVM␣-␣Average␣CV␣Score:", np.mean(cv_s))
print("Best␣params␣for␣linear:", grid_l.best_params_)
print("Best␣params␣for␣polynomial:", grid_p.best_params_)
print("Best␣params␣for␣rbf:", grid_r.best_params_)
print("Best␣params␣for␣sigmoid:", grid_s.best_params_)
```

Output for All Models

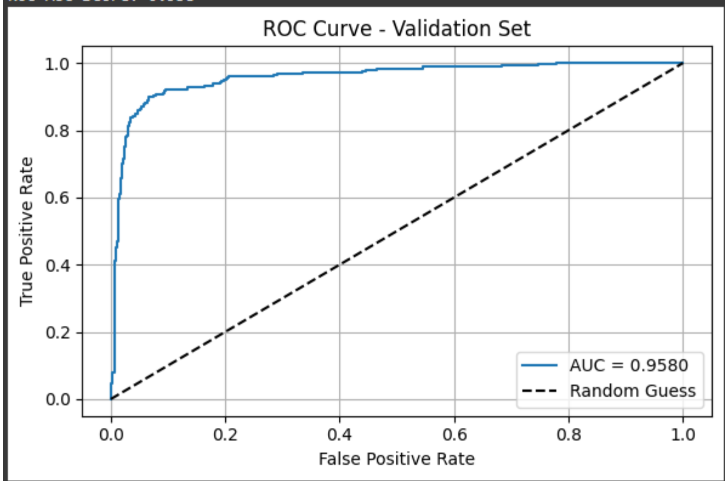
```
 Evaluation - Validation Set
Accuracy : 0.8728
Precision: 0.8842
Recall   : 0.8728
F1 Score : 0.8687

Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.98      0.90       552
           1       0.95      0.72      0.82       368

    accuracy                           0.87       920
   macro avg       0.90      0.85      0.86       920
weighted avg       0.88      0.87      0.87       920

ROC AUC Score: 0.958
```




```
 Evaluation - Test Set
Accuracy : 0.8784
Precision: 0.8842
Recall   : 0.8784
F1 Score : 0.8761

Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.96      0.90       537
           1       0.93      0.77      0.84       384

    accuracy                           0.88       921
   macro avg       0.89      0.86      0.87       921
weighted avg       0.88      0.88      0.88       921
```

Figure 1: MultinomialNB Performance

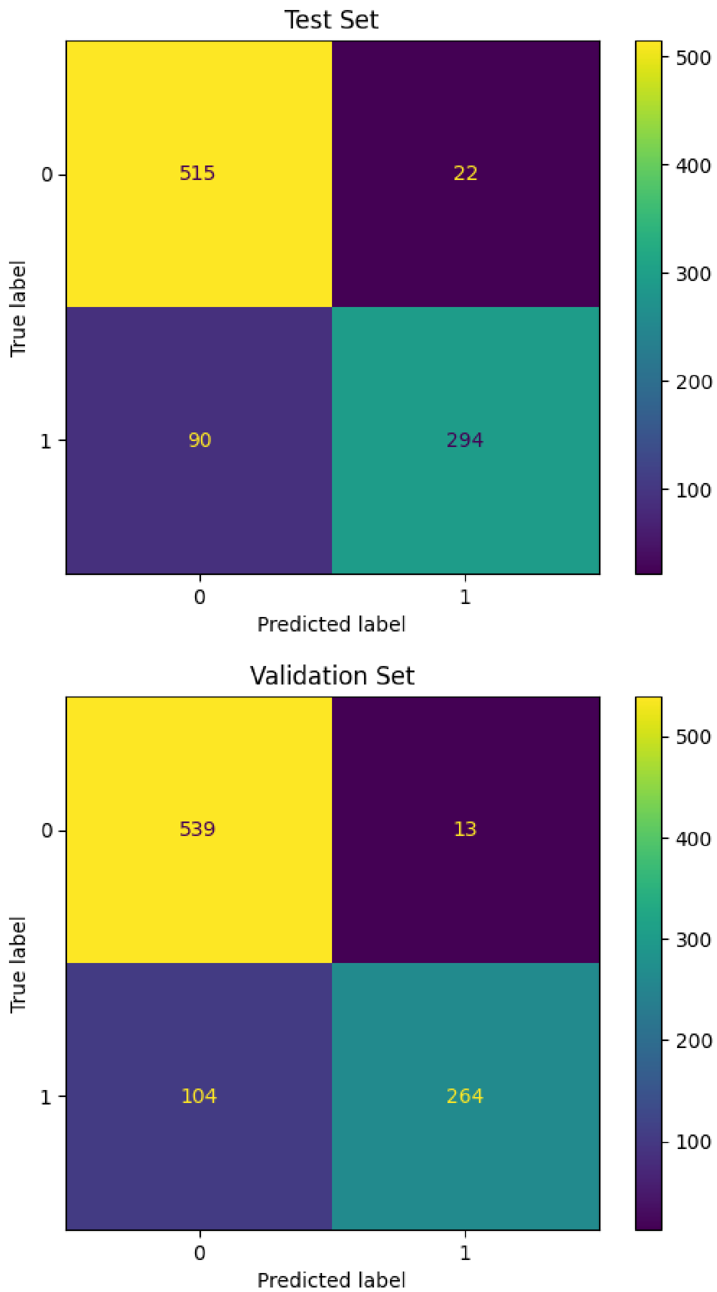


Figure 2: MultinomialNB Confusion Matrix

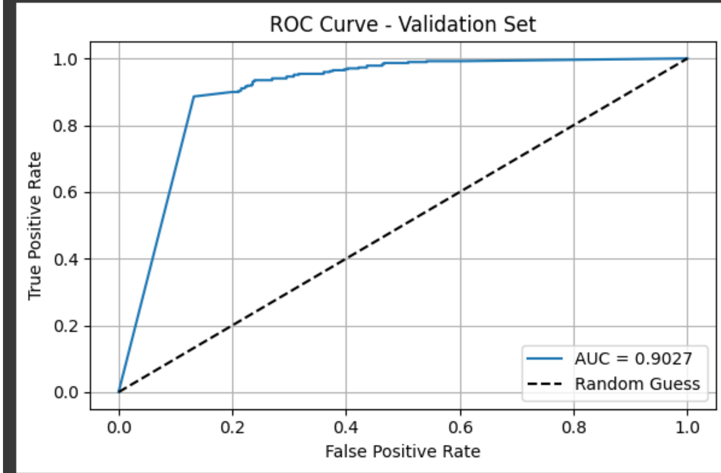
```
 Evaluation - Validation Set
Accuracy : 0.8
Precision: 0.8414
Recall   : 0.8
F1 Score : 0.8014

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.70      0.81       552
           1       0.68      0.94      0.79       368

    accuracy                           0.80       920
   macro avg       0.81      0.82      0.80       920
weighted avg       0.84      0.80      0.80       920

ROC AUC Score: 0.9027
```




```
 Evaluation - Test Set
Accuracy : 0.8274
Precision: 0.8625
Recall   : 0.8274
F1 Score : 0.8279

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.73      0.83       537
           1       0.72      0.96      0.82       384

    accuracy                           0.83       921
   macro avg       0.84      0.85      0.83       921
weighted avg       0.86      0.83      0.83       921
```

Figure 3: GaussianNB Performance

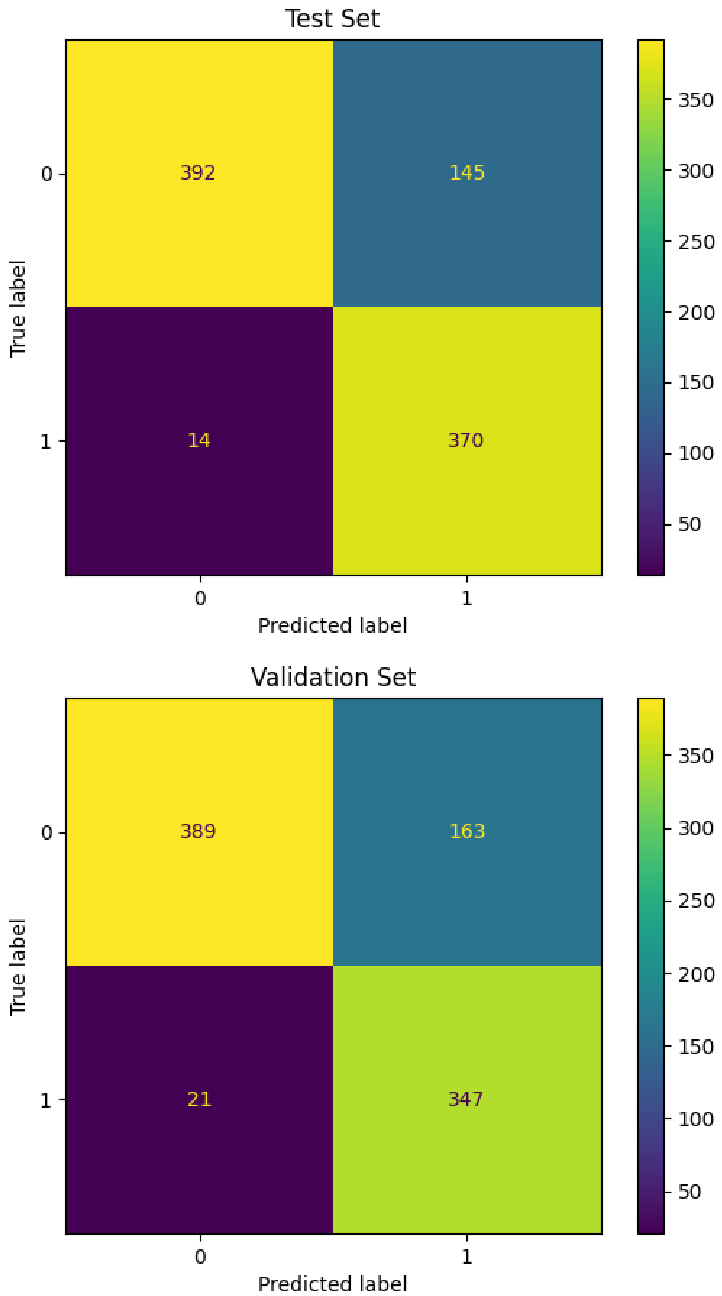


Figure 4: GaussianNB Confusion Matrix

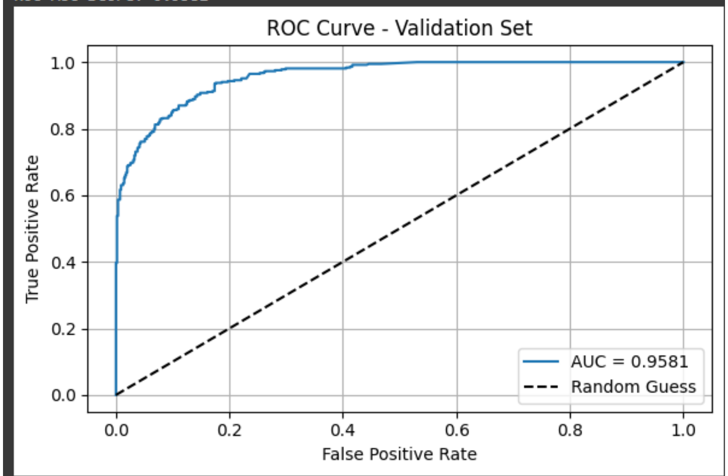
```
 Evaluation - Validation Set
Accuracy : 0.8815
Precision: 0.8818
Recall   : 0.8815
F1 Score : 0.8805

Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.93      0.90       552
           1       0.89      0.81      0.84       368

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

ROC AUC Score: 0.9581
```




```
 Evaluation - Test Set
Accuracy : 0.8827
Precision: 0.8836
Recall   : 0.8827
F1 Score : 0.8818

Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.93      0.90       537
           1       0.90      0.81      0.85       384

    accuracy                           0.88       921
   macro avg       0.89      0.87      0.88       921
weighted avg       0.88      0.88      0.88       921
```

Figure 5: BernoulliNB Performance

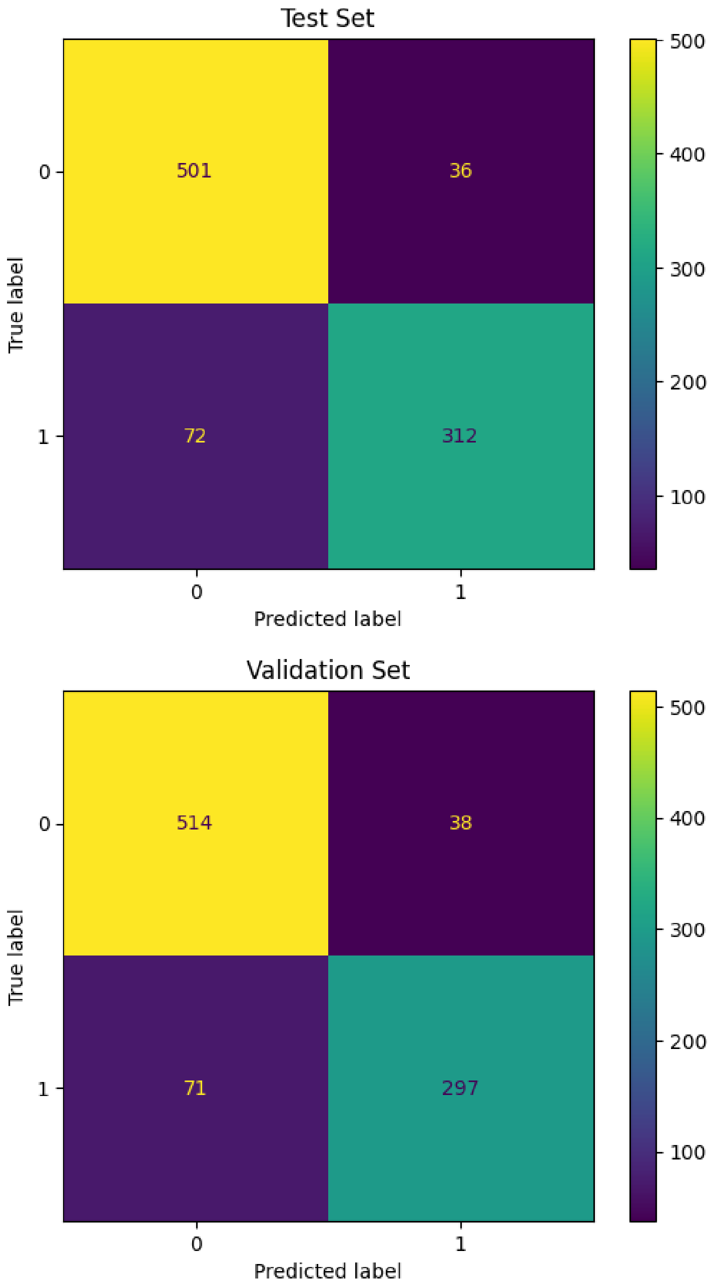


Figure 6: BernoulliNB Confusion Matrix

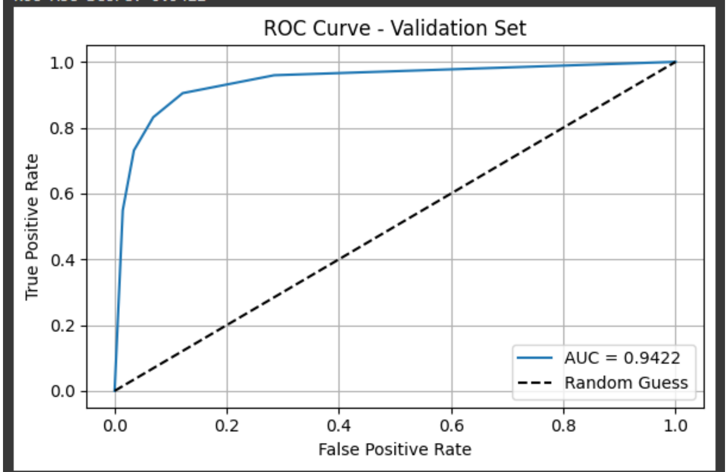
```
 Evaluation - Validation Set
Accuracy : 0.8913
Precision: 0.8912
Recall   : 0.8913
F1 Score : 0.8906

Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.93      0.91       552
           1       0.89      0.83      0.86       368

    accuracy                           0.89       920
   macro avg       0.89      0.88      0.89       920
weighted avg       0.89      0.89      0.89       920

ROC AUC Score: 0.9422
```




```
 Evaluation - Test Set
Accuracy : 0.9088
Precision: 0.9087
Recall   : 0.9088
F1 Score : 0.9086

Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.93      0.92       537
           1       0.90      0.88      0.89       384

    accuracy                           0.91       921
   macro avg       0.91      0.90      0.91       921
weighted avg       0.91      0.91      0.91       921
```

Figure 7: kNN Performance

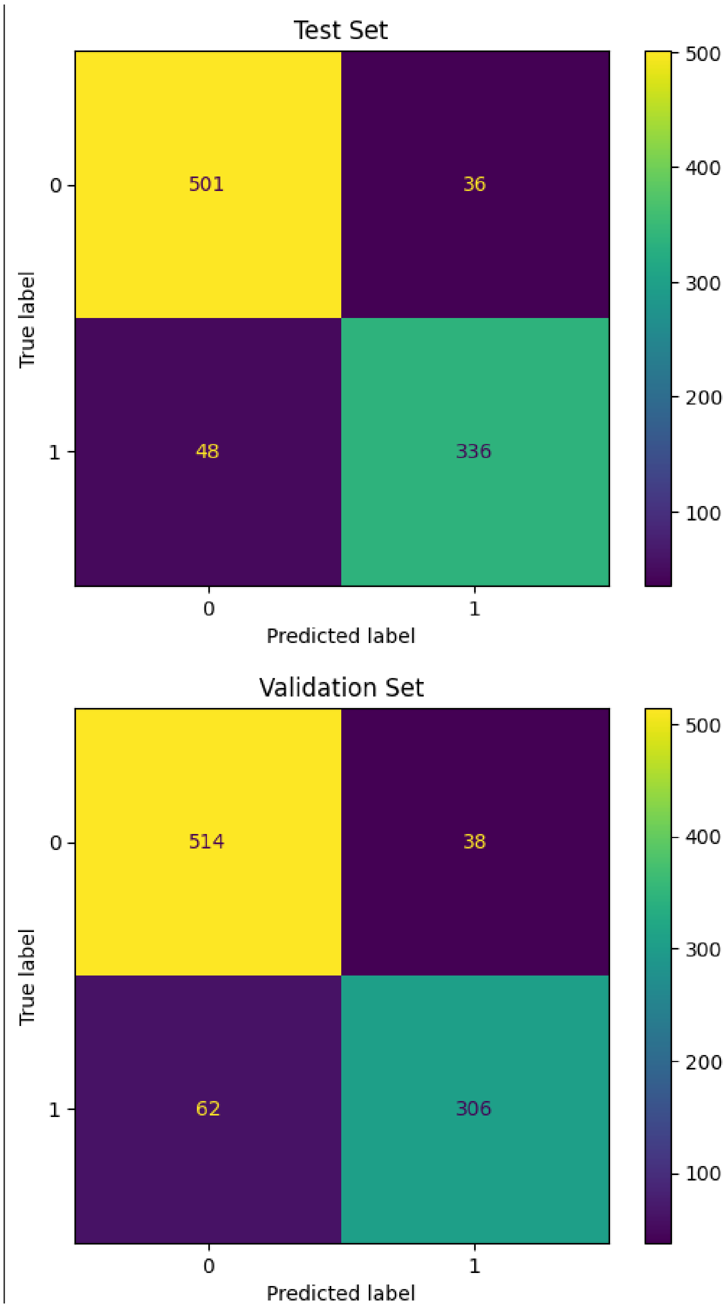


Figure 8: kNN Confusion Matrix

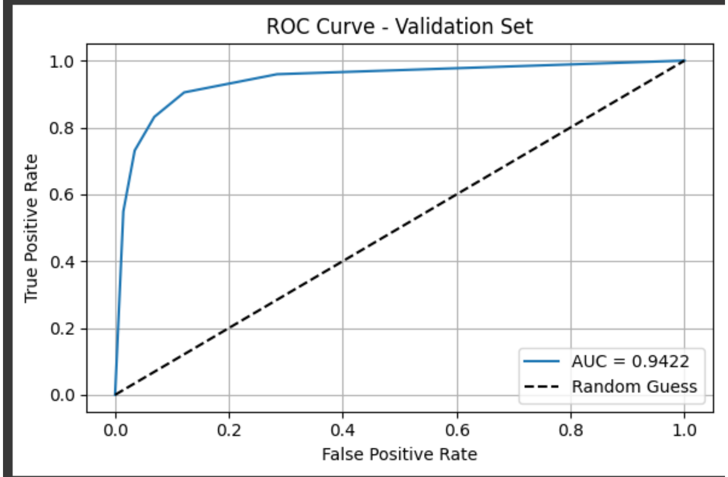
```
 Evaluation - Validation Set
Accuracy : 0.8913
Precision: 0.8912
Recall   : 0.8913
F1 Score : 0.8906

Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.93      0.91       552
           1       0.89      0.83      0.86       368

    accuracy                           0.89       920
   macro avg       0.89      0.88      0.89       920
weighted avg       0.89      0.89      0.89       920

ROC AUC Score: 0.9422
```




```
 Evaluation - Test Set
Accuracy : 0.9088
Precision: 0.9087
Recall   : 0.9088
F1 Score : 0.9086

Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.93      0.92       537
           1       0.90      0.88      0.89       384

    accuracy                           0.91       921
   macro avg       0.91      0.90      0.91       921
weighted avg       0.91      0.91      0.91       921
```

Figure 9: kDTree Performance

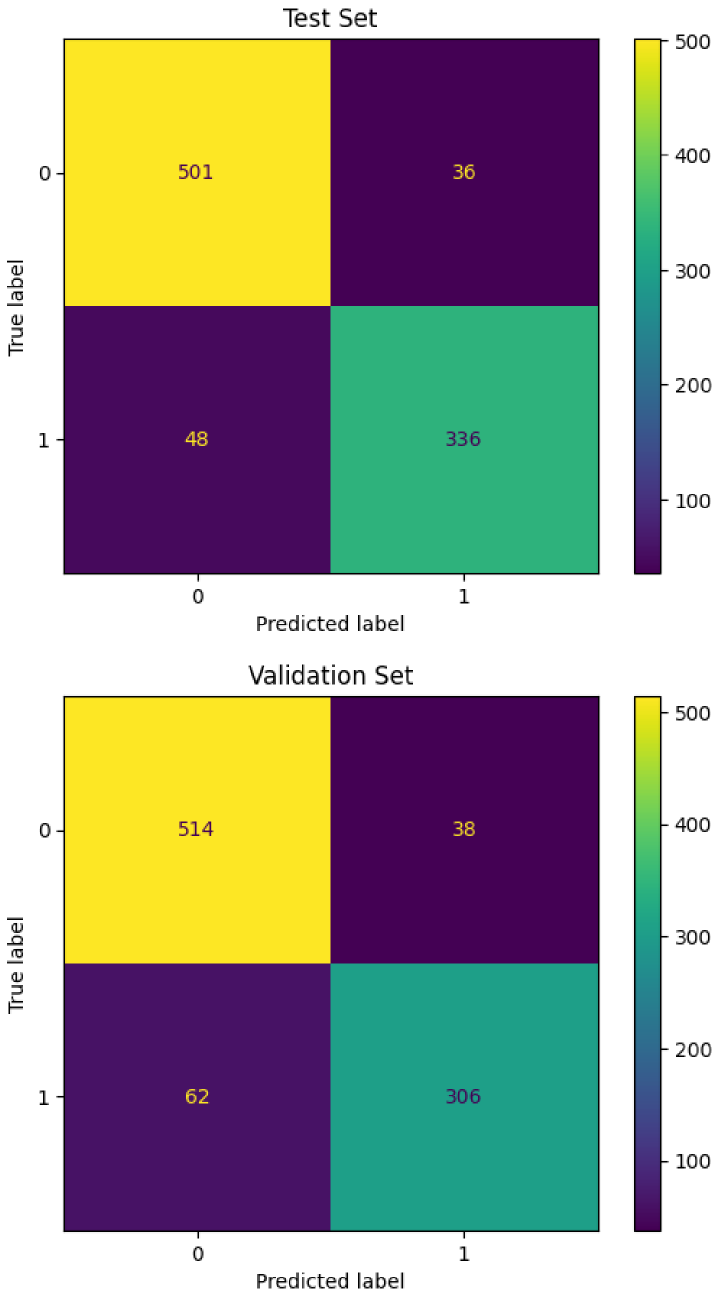


Figure 10: kDTree Confusion Matrix

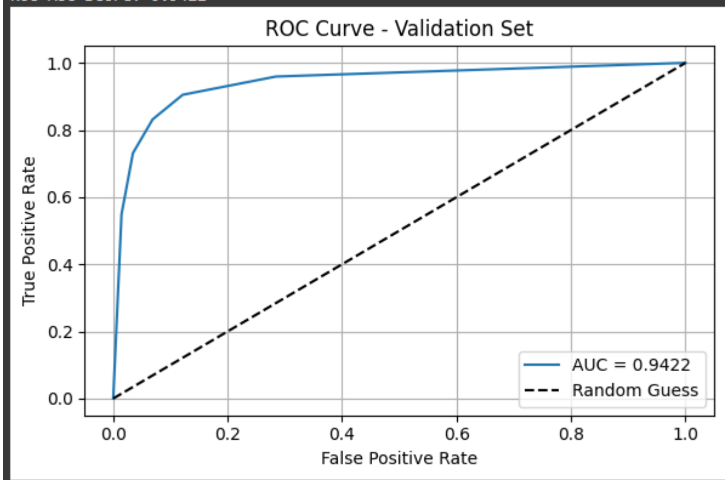
```
 Evaluation - Validation Set
Accuracy : 0.8913
Precision: 0.8912
Recall   : 0.8913
F1 Score : 0.8906

Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.93      0.91       552
           1       0.89      0.83      0.86       368

    accuracy                           0.89       920
   macro avg       0.89      0.88      0.89       920
weighted avg       0.89      0.89      0.89       920

ROC AUC Score: 0.9422
```




```
 Evaluation - Test Set
Accuracy : 0.9088
Precision: 0.9087
Recall   : 0.9088
F1 Score : 0.9086

Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.93      0.92       537
           1       0.90      0.88      0.89       384

    accuracy                           0.91       921
   macro avg       0.91      0.90      0.91       921
weighted avg       0.91      0.91      0.91       921
```

Figure 11: BallTree Performance

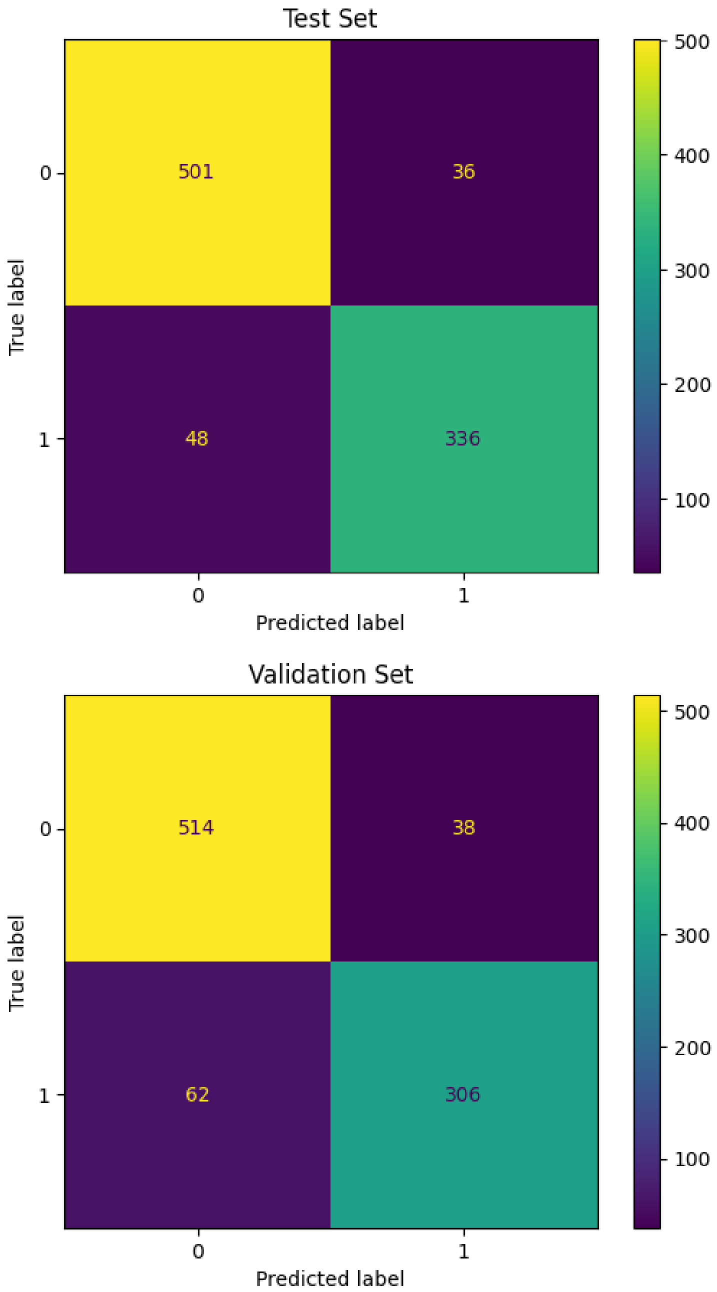


Figure 12: BallTree Confusion Matrix

```
 Evaluation - Validation Set - Linear SVM
Accuracy : 0.9348
Precision: 0.9352
Recall   : 0.9348
F1 Score : 0.9344

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.97      0.95       552
           1       0.95      0.89      0.92       368

    accuracy                           0.93       920
   macro avg       0.94      0.93      0.93       920
weighted avg       0.94      0.93      0.93       920


 Evaluation - Test Set - Linear SVM
Accuracy : 0.9229
Precision: 0.923
Recall   : 0.9229
F1 Score : 0.923

Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.93      0.93       537
           1       0.90      0.91      0.91       384

    accuracy                           0.92       921
   macro avg       0.92      0.92      0.92       921
weighted avg       0.92      0.92      0.92       921


 Evaluation - Validation Set - Polynomial SVM
Accuracy : 0.9261
Precision: 0.9272
Recall   : 0.9261
F1 Score : 0.9254

Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.97      0.94       552
           1       0.95      0.86      0.90       368

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


 Evaluation - Test Set - Polynomial SVM
Accuracy : 0.911
Precision: 0.9118
Recall   : 0.911
F1 Score : 0.9104
```

Figure 13: SVM Performance 1

```
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.95      0.93       537
           1       0.93      0.85      0.89       384

    accuracy                           0.91       921
   macro avg       0.91      0.90      0.91       921
weighted avg       0.91      0.91      0.91       921


 Evaluation - Validation Set - RBF SVM
Accuracy : 0.9391
Precision: 0.9393
Recall   : 0.9391
F1 Score : 0.9389

Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.97      0.95       552
           1       0.95      0.90      0.92       368

    accuracy                           0.94       920
   macro avg       0.94      0.93      0.94       920
weighted avg       0.94      0.94      0.94       920


 Evaluation - Test Set - RBF SVM
Accuracy : 0.9251
Precision: 0.9252
Recall   : 0.9251
F1 Score : 0.9248

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.95      0.94       537
           1       0.93      0.89      0.91       384

    accuracy                           0.93       921
   macro avg       0.93      0.92      0.92       921
weighted avg       0.93      0.93      0.92       921


 Evaluation - Validation Set - Sigmoid SVM
Accuracy : 0.8848
Precision: 0.8845
Recall   : 0.8848
F1 Score : 0.8841
```

Figure 14: SVM Performance 2

```
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.92      0.91       552
           1       0.88      0.83      0.85       368

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


 Evaluation - Test Set - Sigmoid SVM
Accuracy : 0.8882
Precision: 0.8883
Recall   : 0.8882
F1 Score : 0.8882

Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.90      0.90       537
           1       0.86      0.87      0.87       384

    accuracy                           0.89       921
   macro avg       0.88      0.89      0.89       921
weighted avg       0.89      0.89      0.89       921
```

Figure 15: SVM Performance 3

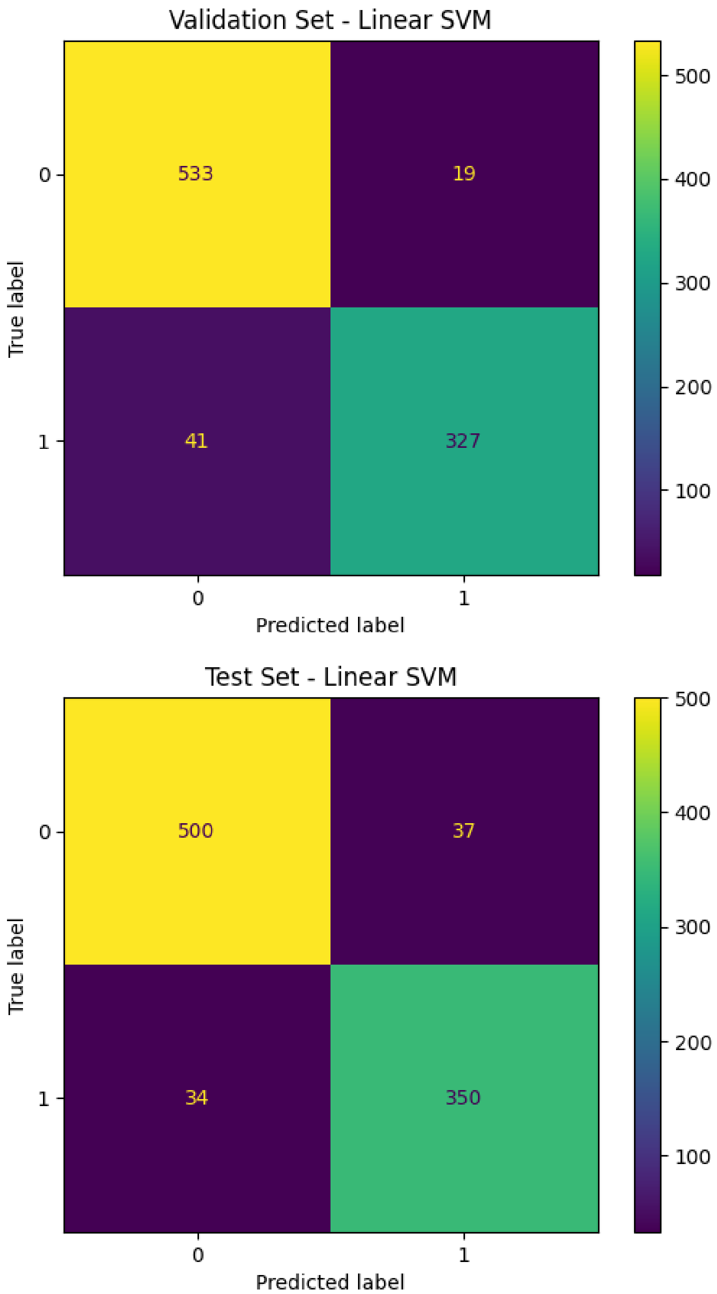


Figure 16: SVM Confusion Matrix 1

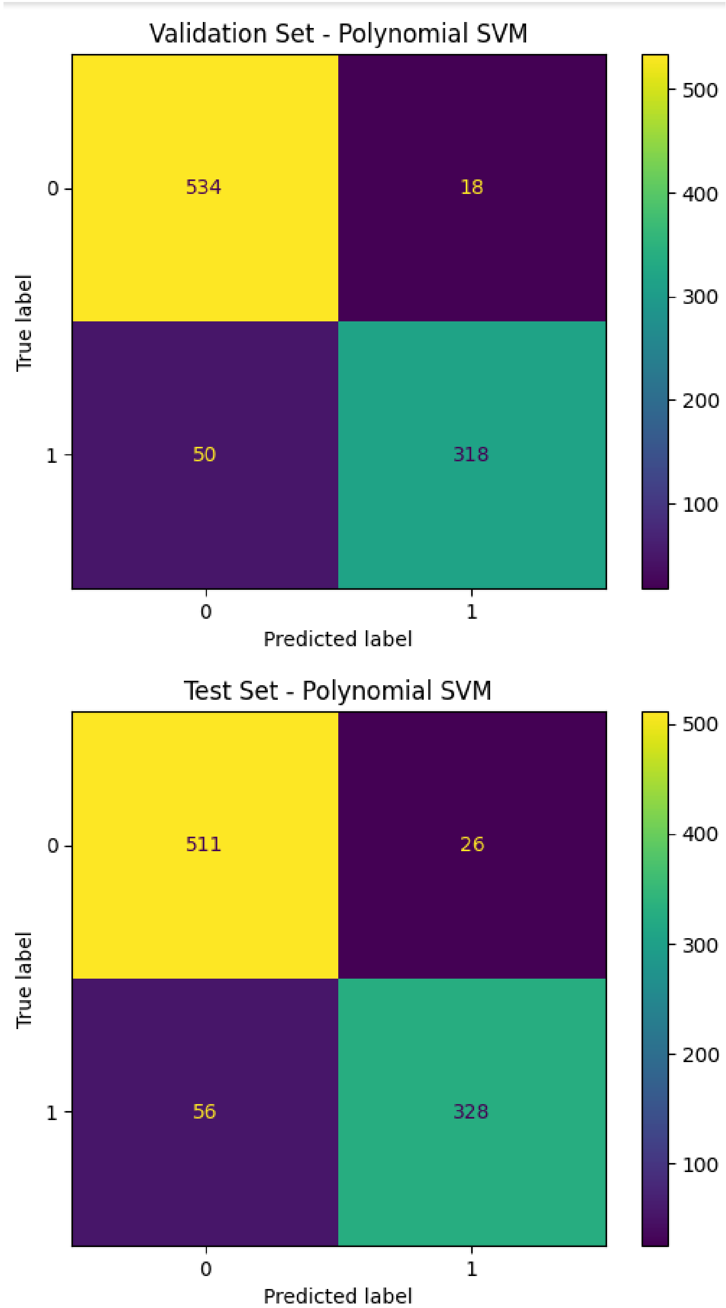


Figure 17: SVM Confusion Matrix 2

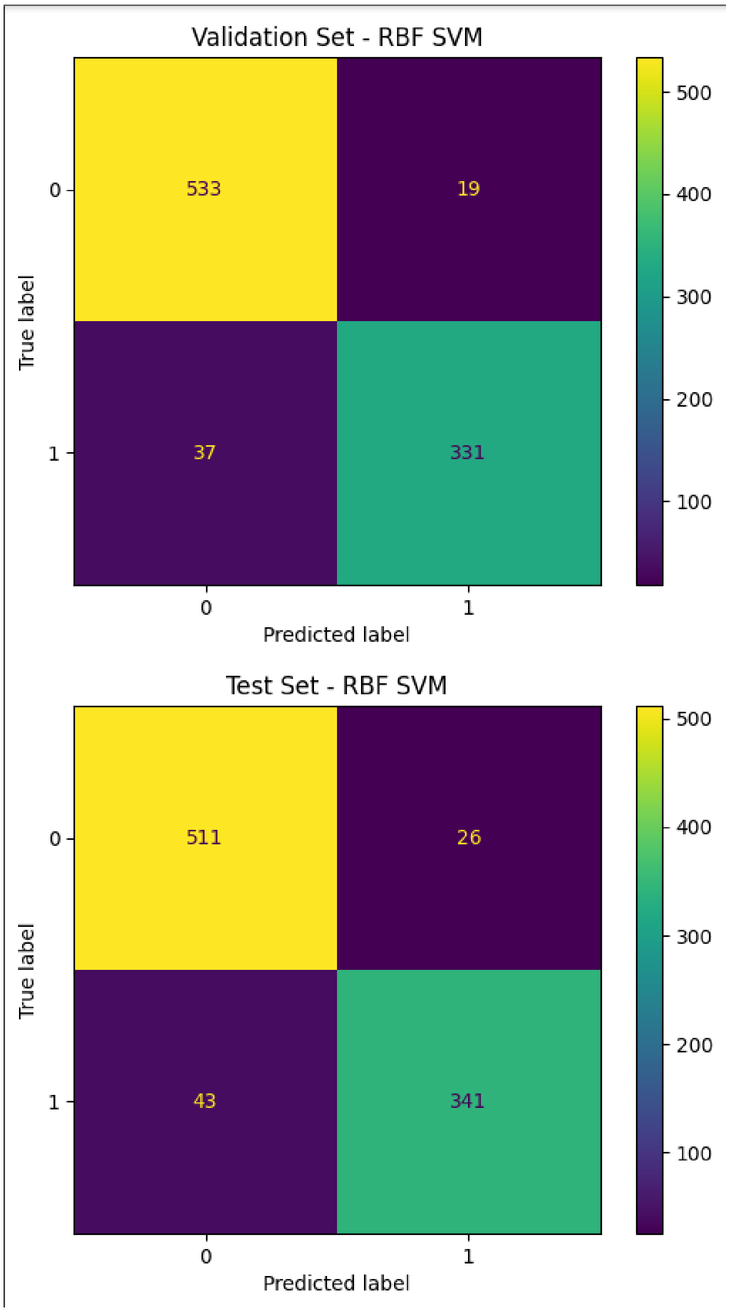


Figure 18: SVM Confusion Matrix 3

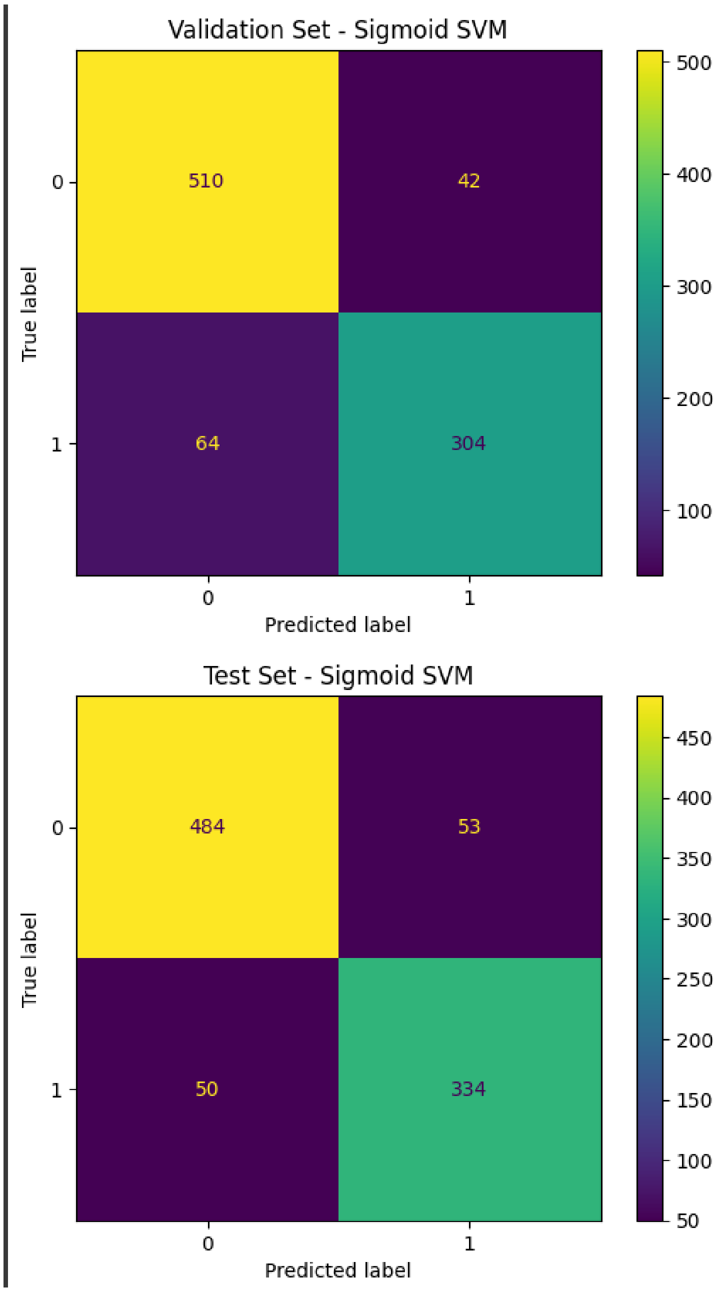


Figure 19: SVM Confusion Matrix 4

1 Results Tables

1.1 Naïve Bayes Variant Comparison

Table 1: Performance Comparison of Naïve Bayes Variants

Metric	Gaussian NB	Multinomial NB	Bernoulli NB
Accuracy	0.8724	0.8784	0.8827
Precision	0.8625	0.8842	0.8836
Recall	0.8274	0.8764	0.8827
F1 Score	0.8279	0.8761	0.8818

1.2 KNN: Varying k Values

Table 2: KNN Performance for Different k Values

k	Accuracy	Precision	Recall	F1 Score
1	0.889251	0.890009	0.889251	0.889477
3	0.893594	0.893388	0.893594	0.893383
5	0.908795	0.908673	0.908795	0.908575
7	0.912052	0.911934	0.912052	0.911859

Table 3: KNN Comparison: KDTree vs BallTree

Metric	KDTree	BallTree
Accuracy	0.908795	0.908795
Precision	0.908673	0.908673
Recall	0.908795	0.908795
F1 Score	0.908575	0.908575
Training Time (s)	0.4053	0.3473

1.3 SVM Performance with Different Kernels and Parameters

Table 4: SVM Performance with Different Kernels and Parameters

Kernel	Hyperparameters	Accuracy	F1 Score	Training Time
Linear	C = 10	0.9229	0.923	12.066
Polynomial	C = 10, degree = 2, gamma = scale	0.911	0.9104	27.869
RBF	C = 1, gamma = auto	0.9251	0.9248	6.716
Sigmoid	C = 1, gamma = auto	0.8882	0.8882	6.783

1.4 K-Fold Cross-Validation Results ($K = 5$)

Table 5: Cross-Validation Scores for Naïve Bayes Variants

Fold	Multinomial NB	Gaussian NB	Bernoulli NB
Fold 1	0.8719	0.8219	0.8806
Fold 2	0.8935	0.8033	0.8902
Fold 3	0.8891	0.7946	0.8837
Fold 4	0.8913	0.8228	0.8870
Fold 5	0.8859	0.8337	0.8902
Average	0.8863	0.8153	0.8863

Observations and Conclusions

- Relative to the other Naive Bayes variations, Gaussian NB is less desirable for this dataset, since Bernoulli and Multinomial NB are built to deal with text based datasets.

- kNN and its variations work at the same accuracy level, and are better than the Naive Bayes classifiers. This is because of the clustering property of kNN, which fits better for text/word vector spaces as compared to the independent feature assumption of Naive Bayes.
- kDTree and BallTree perform at a better time complexity than kNN due to dimensionality reduction.
- SVC's variations perform at higher accuracy than kNN, but take longer training time.
- Specifically, RBF kernel maps the data into an infinite-dimensional feature space, allowing the SVM to create curved decision boundaries that wrap around clusters of similar emails.

Best Practices

- Pre-process the dataset accordingly on the basis of the model used.
- Visualize classification results using appropriate performance metrics.
- Use GridSearch or RandomizedSearch for optimized hyperparameter tuning.
- Compare cross-validation results across multiple models to ensure efficient splitting of dataset.

Learning Outcomes

- Understood and applied classification models like kNN, SVM and Naive Bayes along with their variations on Email classification dataset.
- Evaluated model performance using performance metrics like Accuracy, Precision, Recall and F1-Score.
- Gained experience with hyperparameter tuning using GridSearch.
- Learned about advantages of variations in kNN (kDTree and BallTree), SVC (Linear, Polynomial, RBF, Sigmoid) and Naive Bayes (Multinomial, Gaussian, Bernoulli).
- Learned to compare and conclude why different models give better/worse outputs for a classification dataset.