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

Moogambigai A

August 2025

1 Aim

To classify emails as spam or ham using Naive Bayes, K-Nearest Neighbors (KNN), and Support Vector Machine (SVM), and evaluate their performance using accuracy, precision, recall, F1-score, and K-fold cross-validation.

2 Libraries USed

- Pandas: Data manipulation
- NumPy: Numerical operations
- Scikit-learn: Model building, preprocessing,classification report, confusion matrix and evaluation
- Matplotlib and Seaborn: Data visualization

3 Objective

- The primary goal is to use three specific machine learning models—Naïve Bayes, K-Nearest Neighbors (KNN), and Support Vector Machine (SVM)—to categorize emails.
- To Evaluate Performance Metrics: The models' performance will be measured and compared using key metrics such as accuracy, precision, recall, and F1-score.
- To Use K-Fold Cross-Validation: This technique will be applied to get a more robust evaluation of the models' performance by testing them on different subsets of the data.
- To Compare Model Effectiveness: The assignment aims to determine which classifier and its specific hyperparameters are most effective for this email classification task.

4 Python Code

```
# -----------------------------------------
# 1. Load the Dataset
# -----------------------------------------
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
    , ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset
df = pd.read_csv("/content/drive/MyDrive/spambase.csv")

# Features and labels
X = df.drop("class", axis=1)
y = df["class"]

# Check for missing values
missing_values = df.isnull().sum().sum()

# Separate features and labels
X = df.drop(columns=['class'])
y = df['class']

# Normalize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

missing_values

# Class distribution (0 = ham, 1 = spam)
class_counts = y.value_counts()

# Plot class balance
plt.figure(figsize=(6, 4))
sns.barplot(x=class_counts.index, y=class_counts.values, palette='viridis')
plt.xticks([0, 1], ['Ham␣(0)', 'Spam␣(1)'])
plt.title('Class␣Distribution')
plt.ylabel('Number␣of␣Emails')
plt.xlabel('Email␣Type')
plt.tight_layout()
plt.show()

# -----------------------------------------
# Feature Distributions
# -----------------------------------------

# 1. Histogram for a few important features
selected_features = ['word_freq_free', 'word_freq_money', 'char_freq_%21', '
    capital_run_length_total']

plt.figure(figsize=(12, 8))
for i, feature in enumerate(selected_features):
    plt.subplot(2, 2, i + 1)
    sns.histplot(df[feature], bins=30, kde=True, color='teal')
    plt.title(f'Distribution␣of␣{feature}')
    plt.xlabel(feature)
    plt.ylabel("Frequency")

plt.tight_layout()
plt.show()

# -----------------------------------------
# 3. Splitting the Dataset
```

```
# ----------------------------------------

# Use the normalized dataset
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
)

print("Train␣set␣size:", X_train.shape)
print("Test␣set␣size:", X_test.shape)

# ----------------------------------------
# 4. Model Building: Na ve Bayes & KNN
# ----------------------------------------

import time

# ---------- 1. Prepare Scaled and Raw Versions ----------
X_raw = df.drop('class', axis=1)
y = df['class']

# Scaled version for GaussianNB & KNN
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_raw)

# Train-test split (same random state for consistency)
X_train_scaled, X_test_scaled, y_train, y_test = train_test_split(X_scaled, y,
    test_size=0.2, stratify=y, random_state=42)
X_train_raw, X_test_raw, _, _ = train_test_split( X_raw, y, test_size=0.2,
    stratify=y, random_state=42)

# ---------- 2. Naive Bayes Models ----------
nb_results = []

# GaussianNB (use scaled data)
gnb = GaussianNB()
gnb.fit(X_train_scaled, y_train)
y_pred = gnb.predict(X_test_scaled)

print("\nGaussianNB␣Classification␣Report:\n")
print(classification_report(y_test, y_pred, digits=4))
nb_results.append({
    "Model": "GaussianNB",
    "Accuracy": gnb.score(X_test_scaled, y_test),
    "Precision": classification_report(y_test, y_pred, output_dict=True)['1']['
        precision'],
    "Recall": classification_report(y_test, y_pred, output_dict=True)['1']['recall
        '],
    "F1␣Score": classification_report(y_test, y_pred, output_dict=True)['1']['f1-
        score']
})

# MultinomialNB (use raw data)
mnb = MultinomialNB()
mnb.fit(X_train_raw, y_train)
y_pred = mnb.predict(X_test_raw)
print("\nMultinomialNB␣Classification␣Report:\n")
print(classification_report(y_test, y_pred, digits=4))
nb_results.append({
    "Model": "MultinomialNB",
```

```
    "Accuracy": mnb.score(X_test_raw, y_test),
    "Precision": classification_report(y_test, y_pred, output_dict=True)['1']['
        precision'],
    "Recall": classification_report(y_test, y_pred, output_dict=True)['1']['recall
        '],
    "F1 Score": classification_report(y_test, y_pred, output_dict=True)['1']['f1-
        score']
})

# BernoulliNB (use raw data)
bnb = BernoulliNB()
bnb.fit(X_train_raw, y_train)
y_pred = bnb.predict(X_test_raw)
print("\nBernoulliNB Classification Report:\n")
print(classification_report(y_test, y_pred, digits=4))
nb_results.append({
    "Model": "BernoulliNB",
    "Accuracy": bnb.score(X_test_raw, y_test),
    "Precision": classification_report(y_test, y_pred, output_dict=True)['1']['
        precision'],
    "Recall": classification_report(y_test, y_pred, output_dict=True)['1']['recall
        '],
    "F1 Score": classification_report(y_test, y_pred, output_dict=True)['1']['f1-
        score']
})

#  3. KNN      Varying k
k_values = [1, 3, 5,7]
knn_results = []

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_scaled, y_train)
    y_pred = knn.predict(X_test_scaled)

    print(f"\nKNN (k={k}) Classification Report:\n")
    print(classification_report(y_test, y_pred, digits=4))

    knn_results.append({
        "k": k,
        "Accuracy": knn.score(X_test_scaled, y_test),
        "Precision": classification_report(y_test, y_pred, output_dict=True)['1'][
            'precision'],
        "Recall": classification_report(y_test, y_pred, output_dict=True)['1']['
            recall'],
        "F1 Score": classification_report(y_test, y_pred, output_dict=True)['1']['
            f1-score']
    })

#  4. KNN with KDTree vs BallTree
for algorithm in ['kd_tree', 'ball_tree']:
    knn = KNeighborsClassifier(n_neighbors=5, algorithm=algorithm)
    start = time.time()
    knn.fit(X_train_scaled, y_train)
    elapsed = time.time() - start
    y_pred = knn.predict(X_test_scaled)

    print(f"\nKNN with {algorithm.upper()} Report:\n")
    print(classification_report(y_test, y_pred, digits=4))
```

```
    print(f"Training␣time:␣{elapsed:.4f}␣seconds\n")

# ----------------------------------------
# 4. Model Building: Support Vector Machine (SVM)
# ----------------------------------------

from sklearn.svm import SVC

svm_results = []
kernels = ['linear', 'poly', 'rbf', 'sigmoid']

# Store fitted models in variables for later use
svm_model_linear = None
svm_model_poly = None
svm_model_rbf = None
svm_model_sigmoid = None


for kernel in kernels:
    print(f"\nTraining␣SVM␣with␣{kernel.upper()}␣kernel␣")
    svm_model = SVC(kernel=kernel, random_state=42, probability=True) #
        probability=True for ROC AUC

    start_time = time.time()
    svm_model.fit(X_train_scaled, y_train)
    end_time = time.time()
    training_time = end_time - start_time

    y_pred = svm_model.predict(X_test_scaled)

    report = classification_report(y_test, y_pred, digits=4, output_dict=True)

    svm_results.append({
        "Kernel": kernel.upper(),
        "Accuracy": report['accuracy'],
        "Precision": report['1']['precision'],
        "Recall": report['1']['recall'],
        "F1␣Score": report['1']['f1-score'],
        "Training␣Time␣(s)": training_time
    })

    print(f"Classification␣Report␣for␣{kernel.upper()}␣kernel:")
    print(classification_report(y_test, y_pred, digits=4))
    print(f"Training␣Time:␣{training_time:.4f}␣seconds")

    # Assign the fitted model to the corresponding variable
    if kernel == 'linear':
        svm_model_linear = svm_model
    elif kernel == 'poly':
        svm_model_poly = svm_model
    elif kernel == 'rbf':
        svm_model_rbf = svm_model
    elif kernel == 'sigmoid':
        svm_model_sigmoid = svm_model


# Display results in a table
print("\nSVM␣Kernel-wise␣Results␣")
svm_results_df = pd.DataFrame(svm_results)
```

```
display(svm_results_df)

# ----------------------------------------
# 5. Performance Analysis
# ----------------------------------------
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, roc_curve,
    auc
import matplotlib.pyplot as plt

def plot_conf_matrix_and_roc(model, X, y_true, title="Model"):
    y_pred = model.predict(X)
    y_proba = model.predict_proba(X)[:, 1]

    # Confusion Matrix
    cm = confusion_matrix(y_true, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm)
    disp.plot()
    plt.title(f"Confusion␣Matrix:␣{title}")
    plt.show()

    # ROC Curve
    fpr, tpr, _ = roc_curve(y_true, y_proba)
    roc_auc = auc(fpr, tpr)
    plt.figure()
    plt.plot(fpr, tpr, label=f"{title}␣(AUC␣=␣{roc_auc:.4f})")
    plt.plot([0, 1], [0, 1], linestyle="--", color="gray")
    plt.xlabel("False␣Positive␣Rate")
    plt.ylabel("True␣Positive␣Rate")
    plt.title("ROC␣Curve")
    plt.legend(loc="lower␣right")
    plt.grid()
    plt.show()
#  Na ve Bayes
plot_conf_matrix_and_roc(gnb, X_test_scaled, y_test, "GaussianNB")
plot_conf_matrix_and_roc(mnb, X_test_raw, y_test, "MultinomialNB")
plot_conf_matrix_and_roc(bnb, X_test_raw, y_test, "BernoulliNB")

# KNN: Different k values
for k in [1, 3, 5, 7]:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_scaled, y_train)
    plot_conf_matrix_and_roc(knn, X_test_scaled, y_test, f"KNN␣(k={k})")

# KNN: KDTree and BallTree
for algo in ['kd_tree', 'ball_tree']:
    knn_tree = KNeighborsClassifier(n_neighbors=5, algorithm=algo)
    knn_tree.fit(X_train_scaled, y_train)
    plot_conf_matrix_and_roc(knn_tree, X_test_scaled, y_test, f"KNN␣({algo.upper()
        })")

# Plot ROC and AUC for each trained SVM model
plot_conf_matrix_and_roc(svm_model_linear, X_test_scaled, y_test, "SVM␣(Linear)")
plot_conf_matrix_and_roc(svm_model_poly, X_test_scaled, y_test, "SVM␣(Poly)")
plot_conf_matrix_and_roc(svm_model_rbf, X_test_scaled, y_test, "SVM␣(RBF)")
plot_conf_matrix_and_roc(svm_model_sigmoid, X_test_scaled, y_test, "SVM␣(Sigmoid)"
    )

# ----------------------------------------
# Plotting ROC and AUC for SVM Models
```

```
# ----------------------------------------
# Plot ROC and AUC for each trained SVM model
plot_conf_matrix_and_roc(svm_model_linear, X_test_scaled, y_test, "SVM (Linear)")
plot_conf_matrix_and_roc(svm_model_poly, X_test_scaled, y_test, "SVM (Poly)")
plot_conf_matrix_and_roc(svm_model_rbf, X_test_scaled, y_test, "SVM (RBF)")
plot_conf_matrix_and_roc(svm_model_sigmoid, X_test_scaled, y_test, "SVM (Sigmoid)"
   )
# ----------------------------------------
# 6. K-Fold Cross-Validation
# ----------------------------------------
from sklearn.model_selection import cross_val_score, KFold
models = {
    "Na ve Bayes (Gaussian)": gnb,
    "Na ve Bayes (Multinomial)": mnb,
    "Na ve Bayes (Bernoulli)": bnb,
    "KNN (k=7)": knn,
    "SVM (Linear)": svm_model_linear
}
data_for_cv = {
    "Na ve Bayes (Gaussian)": (X_scaled, y),
    "Na ve Bayes (Multinomial)": (X_raw, y),
    "Na ve Bayes (Bernoulli)": (X_raw, y),
    "KNN (k=7)": (X_scaled, y),
    "SVM (Linear)": (X_scaled, y)
}
# Perform K-Fold Cross-Validation (K=5)
n_splits = 5
kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)
cv_results = {}
for name, model in models.items():
    X_cv, y_cv = data_for_cv[name]
    print(f"\nPerforming {n_splits}-Fold Cross-Validation for {name}...")
    scores = cross_val_score(model, X_cv, y_cv, cv=kf, scoring='accuracy')
    cv_results[name] = scores
    print(f"Scores: {scores}")
    print(f"Average Accuracy: {scores.mean():.4f}")
# Display results in a table
print("\n--- K-Fold Cross-Validation Results (K=5) ---")
cv_results_df = pd.DataFrame(cv_results)
cv_results_df.index = [f"Fold {i+1}" for i in range(n_splits)]
cv_results_df.loc["Average"] = cv_results_df.mean()
display(cv_results_df)
```

5 Output Screenshots

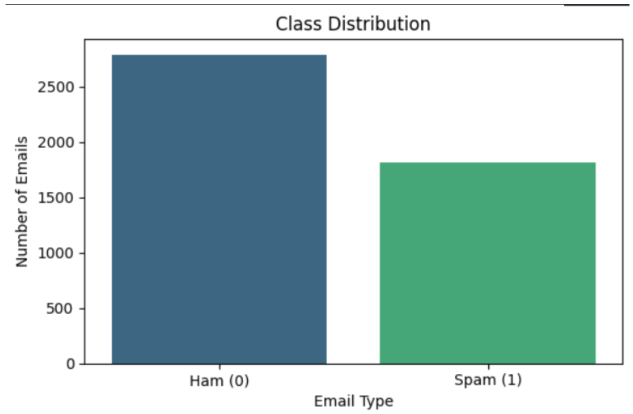
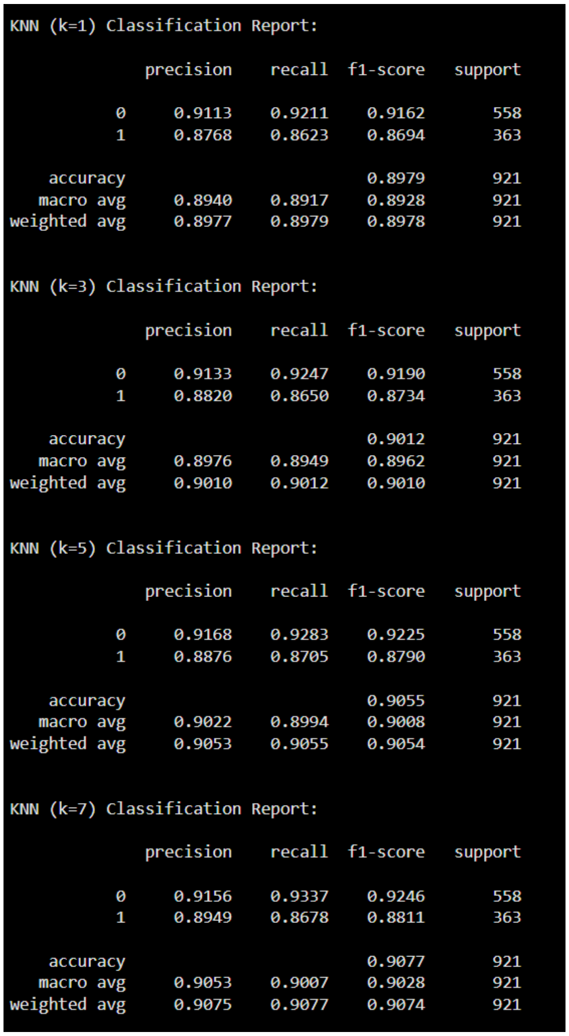


Figure 1: Class Distribution of Dataset

```
GaussianNB Classification Report:

              precision    recall  f1-score   support

           0     0.9654    0.7509    0.8448       558
           1     0.7146    0.9587    0.8188       363

    accuracy                         0.8328       921
   macro avg     0.8400    0.8548    0.8318       921
weighted avg     0.8666    0.8328    0.8345       921


MultinomialNB Classification Report:

              precision    recall  f1-score   support

           0     0.8121    0.8208    0.8164       558
           1     0.7199    0.7080    0.7139       363

    accuracy                         0.7763       921
   macro avg     0.7660    0.7644    0.7651       921
weighted avg     0.7757    0.7763    0.7760       921


BernoulliNB Classification Report:

              precision    recall  f1-score   support

           0     0.8788    0.9229    0.9003       558
           1     0.8716    0.8044    0.8367       363

    accuracy                         0.8762       921
   macro avg     0.8752    0.8637    0.8685       921
weighted avg     0.8760    0.8762    0.8753       921
```

Figure 2: Naive Bayes Output



KNN (k=1) Classification Report:

	precision	recall	f1-score	support
0	0.9113	0.9211	0.9162	558
1	0.8768	0.8623	0.8694	363
accuracy			0.8979	921
macro avg	0.8940	0.8917	0.8928	921
weighted avg	0.8977	0.8979	0.8978	921

KNN (k=3) Classification Report:

	precision	recall	f1-score	support
0	0.9133	0.9247	0.9190	558
1	0.8820	0.8650	0.8734	363
accuracy			0.9012	921
macro avg	0.8976	0.8949	0.8962	921
weighted avg	0.9010	0.9012	0.9010	921

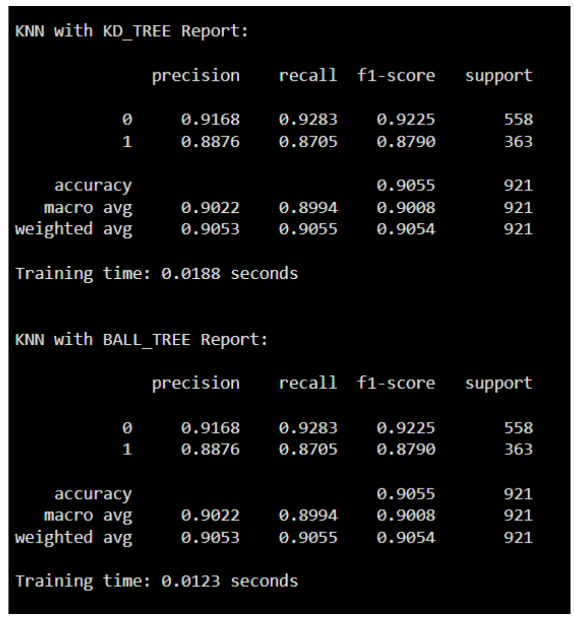
KNN (k=5) Classification Report:

	precision	recall	f1-score	support
0	0.9168	0.9283	0.9225	558
1	0.8876	0.8705	0.8790	363
accuracy			0.9055	921
macro avg	0.9022	0.8994	0.9008	921
weighted avg	0.9053	0.9055	0.9054	921

KNN (k=7) Classification Report:

	precision	recall	f1-score	support
0	0.9156	0.9337	0.9246	558
1	0.8949	0.8678	0.8811	363
accuracy			0.9077	921
macro avg	0.9053	0.9007	0.9028	921
weighted avg	0.9075	0.9077	0.9074	921

(a) KNN Output



KNN with KD_TREE Report:

	precision	recall	f1-score	support
0	0.9168	0.9283	0.9225	558
1	0.8876	0.8705	0.8790	363
accuracy			0.9055	921
macro avg	0.9022	0.8994	0.9008	921
weighted avg	0.9053	0.9055	0.9054	921

Training time: 0.0188 seconds

KNN with BALL_TREE Report:

	precision	recall	f1-score	support
0	0.9168	0.9283	0.9225	558
1	0.8876	0.8705	0.8790	363
accuracy			0.9055	921
macro avg	0.9022	0.8994	0.9008	921
weighted avg	0.9053	0.9055	0.9054	921

Training time: 0.0123 seconds

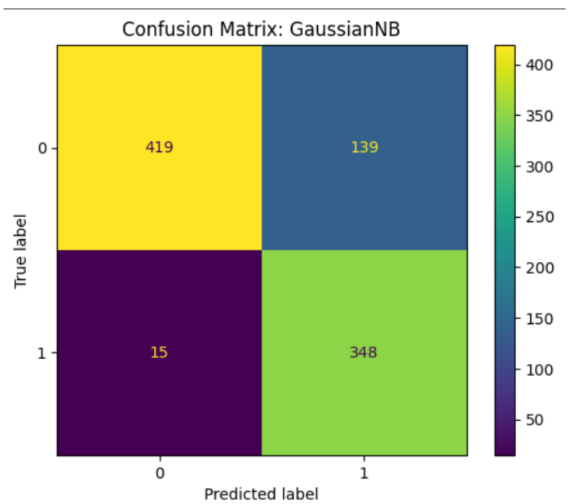
(b) KDTREE Vs BALLTREE Output

```
Training SVM with LINEAR kernel
Classification Report for LINEAR kernel:
              precision    recall  f1-score   support

           0     0.9349    0.9516    0.9432       558
           1     0.9235    0.8981    0.9106       363

    accuracy                         0.9305       921
   macro avg     0.9292    0.9248    0.9269       921
weighted avg     0.9304    0.9305    0.9303       921

Training Time: 3.6872 seconds

Training SVM with POLY kernel
Classification Report for POLY kernel:
              precision    recall  f1-score   support

           0     0.7376    0.9875    0.8444       558
           1     0.9598    0.4601    0.6220       363

    accuracy                         0.7796       921
   macro avg     0.8487    0.7238    0.7332       921
weighted avg     0.8252    0.7796    0.7568       921

Training Time: 2.8629 seconds
```

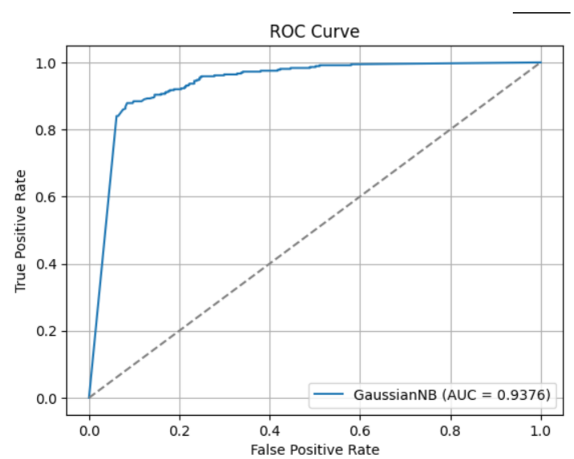
(a) SVM Output

```
Training SVM with RBF kernel
Classification Report for RBF kernel:
              precision    recall  f1-score   support

           0     0.9270    0.9552    0.9409       558
           1     0.9277    0.8843    0.9055       363

    accuracy                         0.9273       921
   macro avg     0.9274    0.9197    0.9232       921
weighted avg     0.9273    0.9273    0.9269       921

Training Time: 1.7237 seconds

Training SVM with SIGMOID kernel
Classification Report for SIGMOID kernel:
              precision    recall  f1-score   support

           0     0.9034    0.9050    0.9042       558
           1     0.8536    0.8512    0.8524       363

    accuracy                         0.8838       921
   macro avg     0.8785    0.8781    0.8783       921
weighted avg     0.8838    0.8838    0.8838       921

Training Time: 1.8486 seconds
```

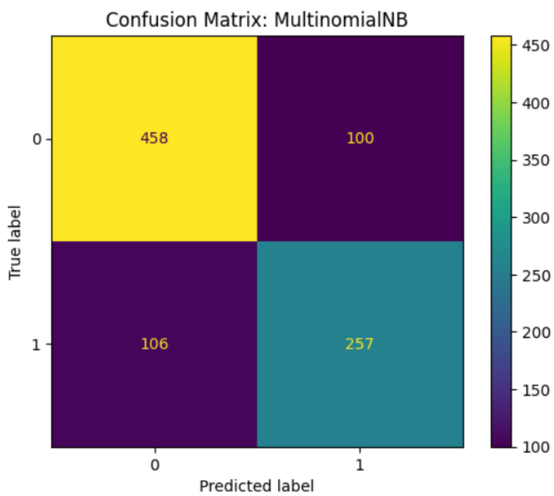
(b) SVM Output



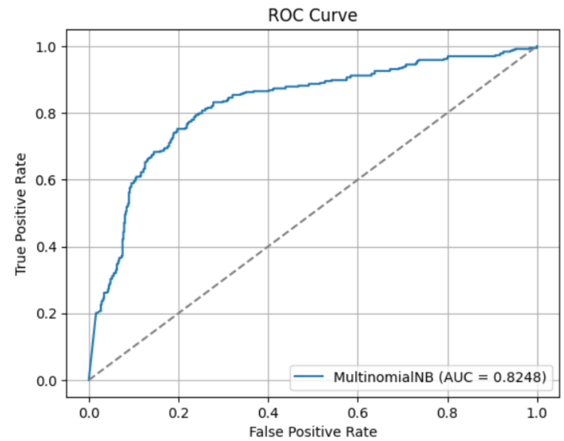
(a) GaussianNB confusion matrix



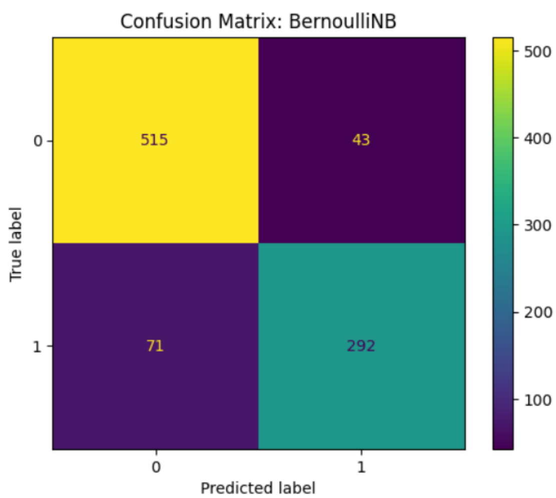
(b) GaussianNB ROC AUC



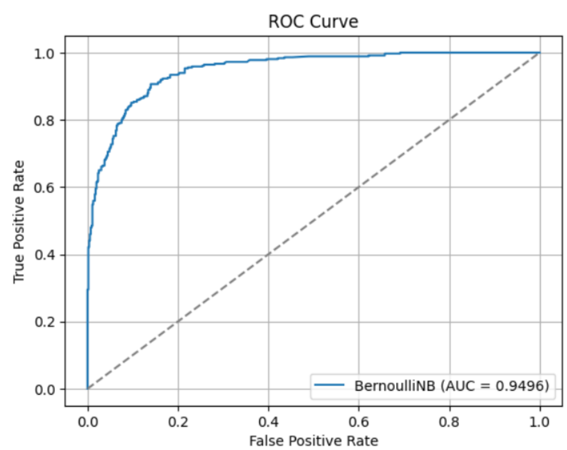
(a) MultinomialNB confusion matrix



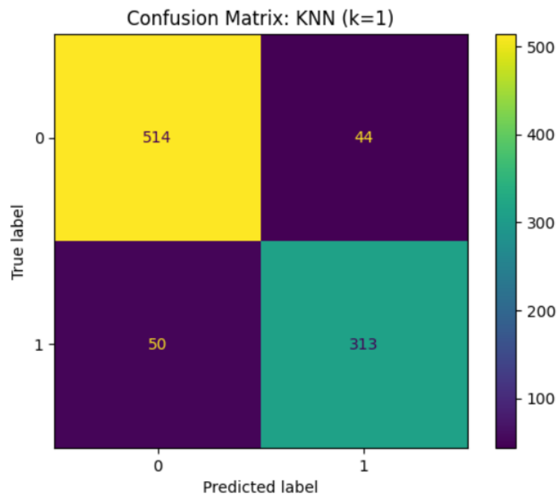
(b) MultinomialNB ROC AUC



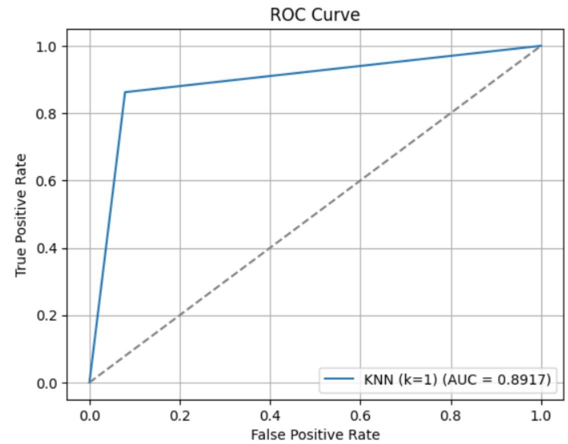
(a) BernoliNB confusion matrix



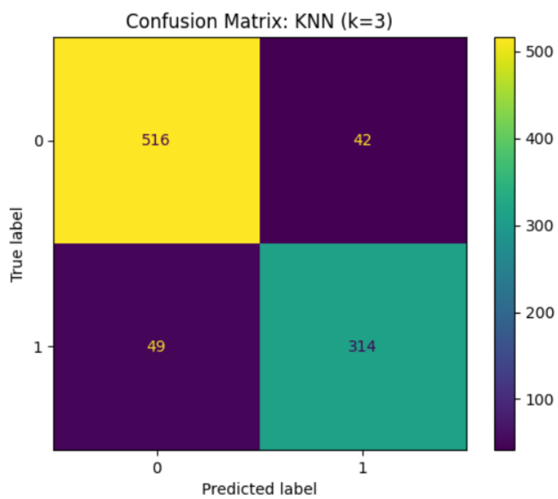
(b) BernoliNB ROC AUC



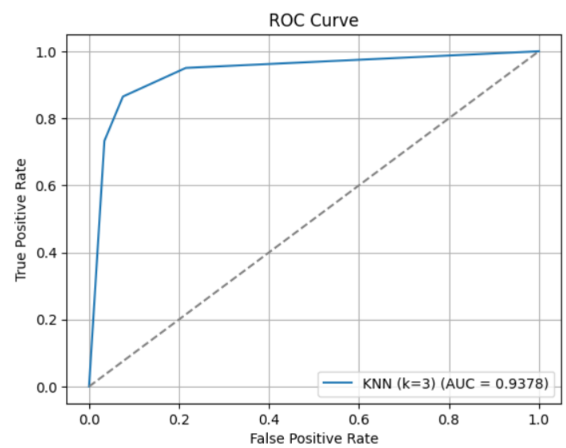
(a) KNN (k=1) confusion matrix



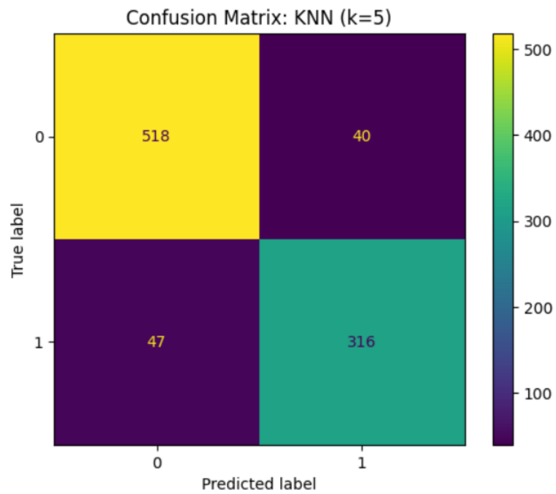
(b) KNN (k=1) ROC AUC



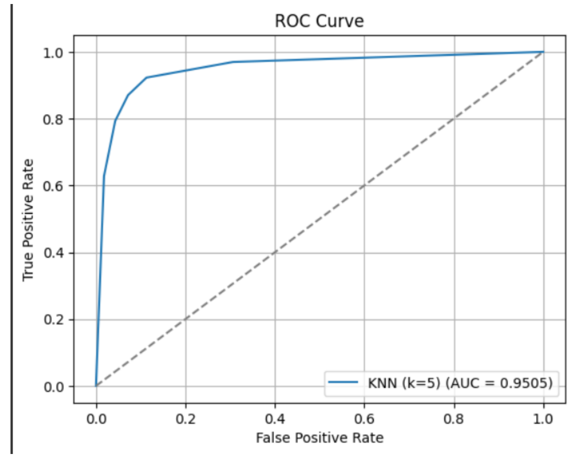
(a) KNN (k=3) confusion matrix



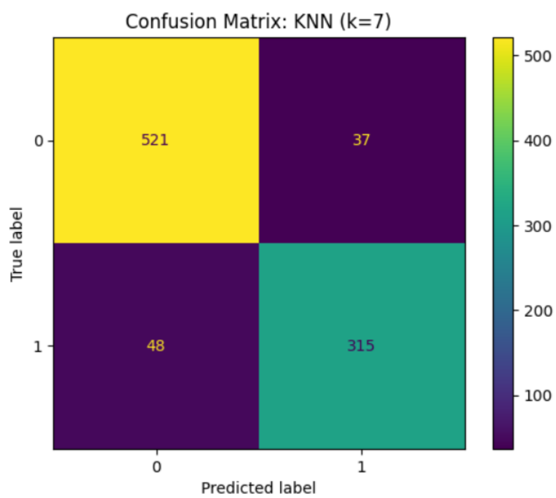
(b) KNN (k=3) ROC AUC



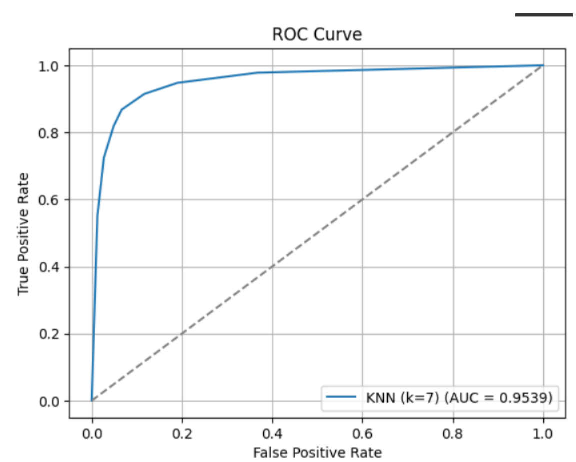
(a) KNN (k=5) confusion matrix



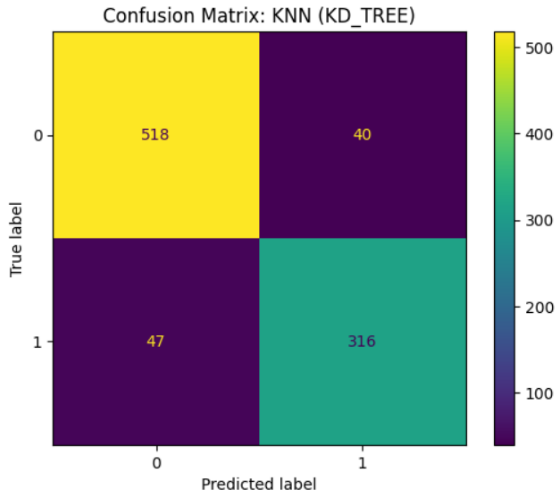
(b) KNN (k=5) ROC AUC



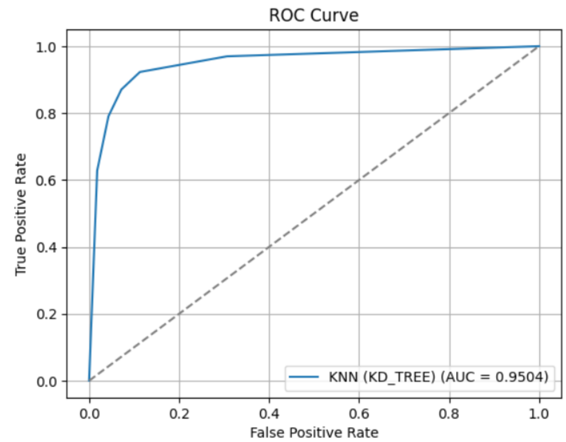
(a) KNN (k=7) confusion matrix



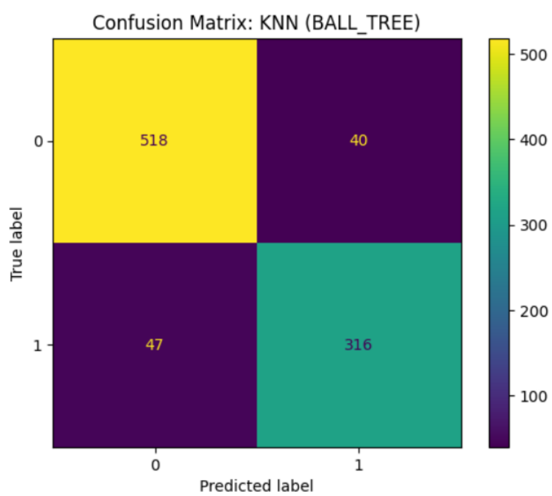
(b) KNN (k=7) ROC AUC



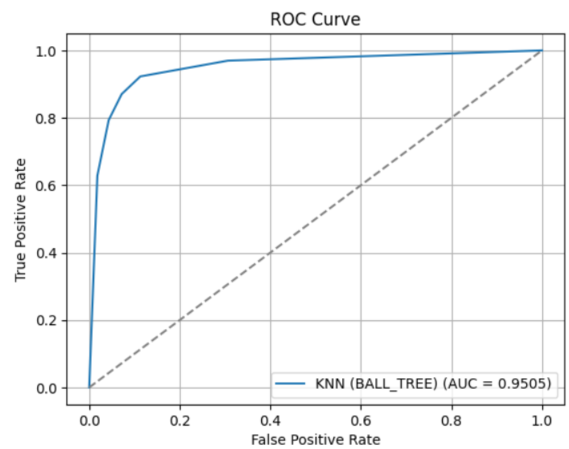
(a) KNN KDTREE confusion matrix



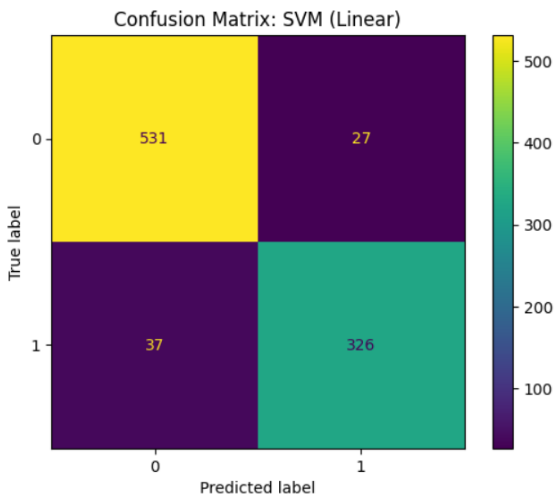
(b) KNN KDTREE ROC AUC



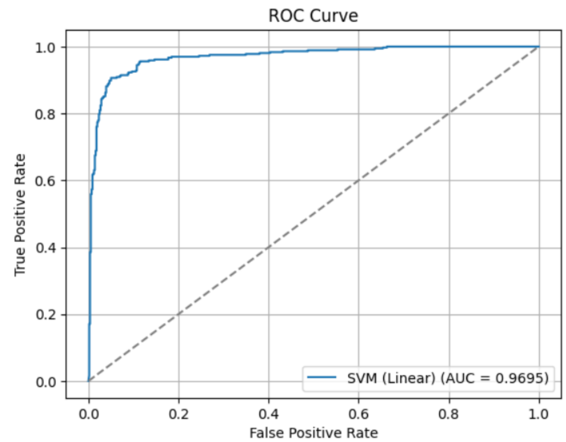
(a) KNN BALLTREE confusion matrix



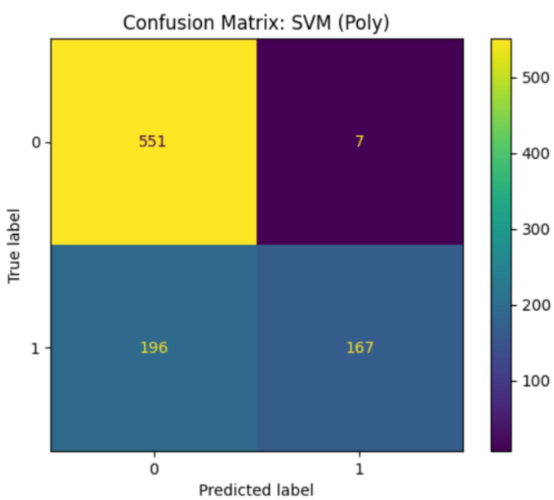
(b) KNN BALLTREE ROC AUC



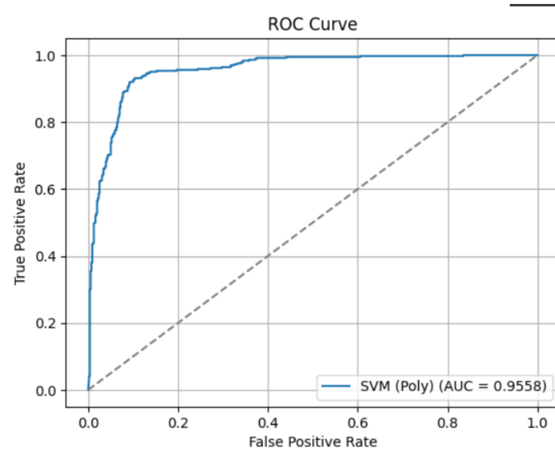
(a) SVM Linear confusion matrix



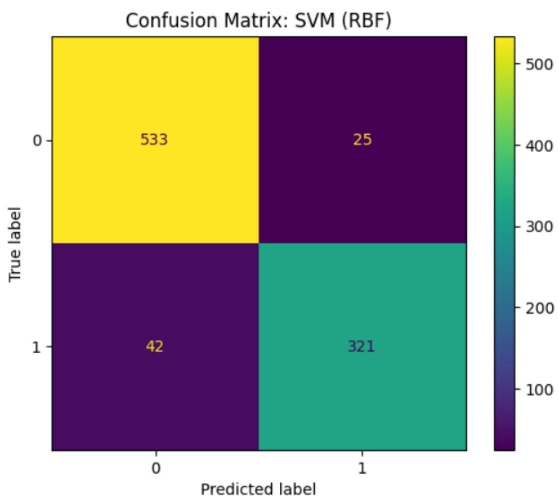
(b) SVM Linear ROC AUC



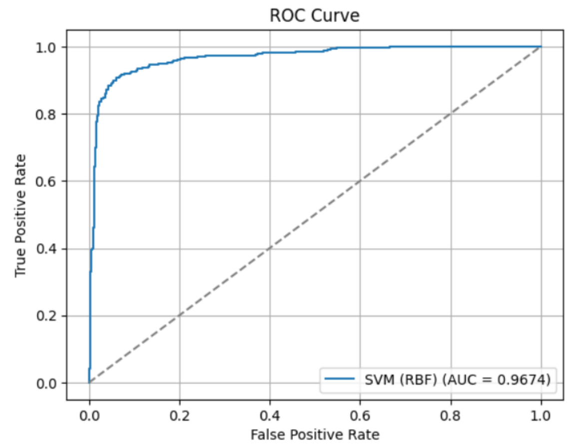
(a) SVM Poly confusion matrix



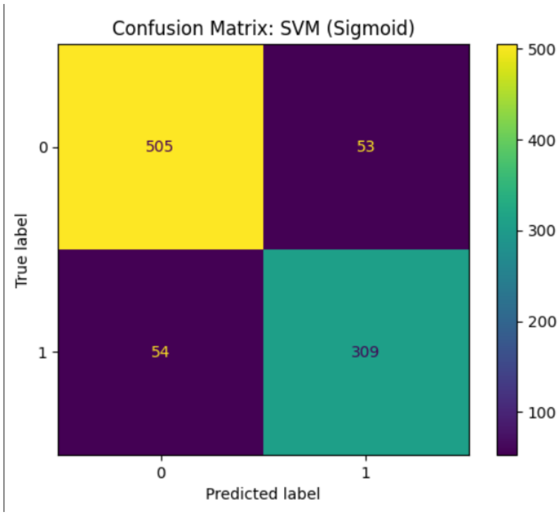
(b) SVM Poly ROC AUC



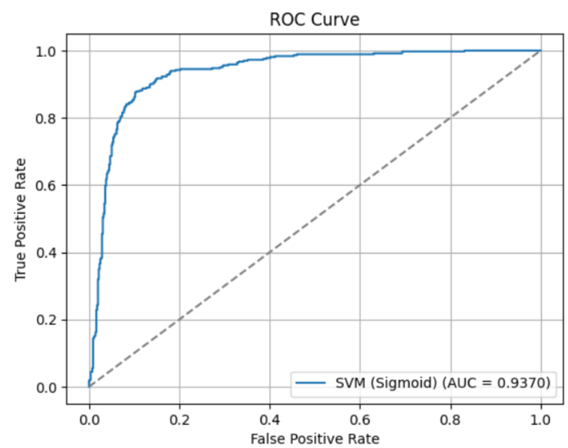
(a) SVM RBF confusion matrix



(b) SVM RBF ROC AUC



(a) SVM sigmoid confusion matrix



(b) SVM sigmoid ROC AUC

SVM Kernel-wise Results

	Kernel	Accuracy	Precision	Recall	F1 Score	Training Time (s)
0	LINEAR	0.930510	0.923513	0.898072	0.910615	3.687203
1	POLY	0.779587	0.959770	0.460055	0.621974	2.862876
2	RBF	0.927253	0.927746	0.884298	0.905501	1.723683
3	SIGMOID	0.883822	0.853591	0.851240	0.852414	1.848590

(a) SVM Kernel-wise Results

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

	Naïve Bayes (Gaussian)	Naïve Bayes (Multinomial)	Naïve Bayes (Bernoulli)	KNN (k=7)	SVM (Linear)
Fold 1	0.821933	0.786102	0.880565	0.895765	0.925081
Fold 2	0.803261	0.776087	0.890217	0.913043	0.928261
Fold 3	0.794565	0.780435	0.883696	0.913043	0.916304
Fold 4	0.822826	0.814130	0.886957	0.900000	0.936957
Fold 5	0.833696	0.801087	0.890217	0.911957	0.930435
Average	0.815256	0.791568	0.886330	0.906762	0.927408

(b) 5-Fold Cross-Validation

6 Observation

1) **Which classifier had the best average accuracy?**

- Based on the 5-fold cross-validation results, the **SVM (Linear)** classifier achieved the best average accuracy of **0.9274**. Other classifiers and their average accuracies were: **KNN (k=7)** at 0.9068, **Naive Bayes (Bernoulli)** at 0.8863, **Naive Bayes (Gaussian)** at 0.8153, and **Naive Bayes (Multinomial)** at 0.7916.

2) **Which Naive Bayes variant worked best?**

- The **Bernoulli Naive Bayes** variant performed the best among the three Naive Bayes models. It had an average accuracy of **0.8863** from 5-fold cross-validation, a precision of 0.8536, a recall of 0.8512, and an F1 score of 0.8524 on the test set. This was significantly higher than Gaussian Naive Bayes (average accuracy 0.8153) and Multinomial Naive Bayes (average accuracy 0.7916). The Bernoulli model also had the highest AUC score of **0.9496** among the Naive Bayes variants.

3) **How did KNN accuracy vary with k and tree type?**

- The accuracy of the KNN classifier generally **improved as the value of k increased**. On the test set, the accuracy was 0.9055 for k=5 and 0.9054 for k=7. The Area Under the Curve (AUC) also increased with k, from 0.8917 at k=1, to 0.9378 at k=3, 0.9505 at k=5, and **0.9539 at k=7**.

- The choice of tree type (**KDTREE** vs. **BALLTREE**) for KNN with k=5 had **no impact on accuracy or performance metrics**; both models achieved an accuracy of 0.9055 and a near-identical AUC (0.9504 and 0.9505, respectively). The only difference was that the Ball Tree algorithm had a slightly faster training time (0.0123 seconds) compared to the KD Tree algorithm (0.0188 seconds).

4) **Which SVM kernel was most effective?**

- The **linear kernel** was the most effective for the SVM classifier. It achieved the highest accuracy of **0.9305** on the test set, followed by the RBF kernel at 0.9273. The linear kernel also had the best overall performance metrics, including an F1 score of 0.9106, while also having the highest AUC of **0.9695**.

- While the RBF kernel had a higher recall for ham emails (class 0) at 0.9552 compared to the linear kernel's 0.9516, the linear kernel's performance for spam emails (class 1) was stronger, with a precision of 0.9235 compared to RBF's 0.9277, and recall of 0.8981 compared to RBF's 0.8843.

5) **How did hyperparameters influence performance?**

- Hyperparameters significantly influenced the performance of the models.
  - **KNN**: The hyperparameter **k** (the number of neighbors) directly affected the model's accuracy, precision, recall, and AUC. A larger k value (up to 7) generally resulted in higher accuracy and AUC, suggesting that considering more neighbors led to a more robust classification. The algorithm hyperparameter had a negligible effect on performance but did influence training time.
  - **SVM**: The choice of **kernel** was the most crucial hyperparameter for SVM. The linear and RBF kernels performed best, with the **linear kernel** having the highest accuracy and AUC. In contrast, the poly (polynomial) kernel performed poorly, with a low accuracy of 0.7796 and a low F1 score of 0.6220 for spam classification. The sigmoid kernel also performed worse than the linear and RBF kernels, with a lower accuracy of 0.8838. This demonstrates that some kernels are better suited for this specific dataset and classification problem.

7 Learning Outcomes

From this assignment,

- We gained practical experience in applying three fundamental machine learning models—Naïve Bayes, K-Nearest Neighbors (KNN), and Support Vector Machine (SVM)—to a real-world classification problem.

- We learned the importance of preparing data for machine learning, including handling missing values, standardizing or normalizing features, and splitting the dataset for training and testing.

- We explored how model performance is influenced by different hyperparameters, such as the k value in KNN and the kernel type in SVM.

- We learned to evaluate model effectiveness using a variety of metrics like accuracy, precision, recall, and F1-score. We also learned to visualize performance using confusion matrices and ROC curves to gain a deeper understanding of model behavior.
- We understood the significance of using K-fold cross-validation to get a more reliable estimate of a model's performance and avoid overfitting to a single train-test split.
- We were able to compare the strengths and weaknesses of different algorithms and their variants to determine which one is most suitable for a given dataset.