Experiment 2: Email Spam or Ham Classification using Naïve Bayes and KNN

M. Tech (Integrated) Computer Science & Engineering
Semester V
Machine Learning Algorithms Laboratory (ICS1512)
Academic year: 2025–2026 (Odd)

Name: Akshith Viswanathan RegNo: 3122237001002

Aim and Objective

To classify emails as spam or ham using Naïve Bayes (Gaussian, Multinomial, Bernoulli) and K-Nearest Neighbors (KNN) classifiers on the Spambase dataset. Evaluate performance using test metrics and K-Fold cross-validation.

Libraries Used

- pandas, numpy
- scikit-learn (train_test_split, StratifiedKFold, StandardScaler, OneHotEncoder, SimpleImputer)
- scikit-learn models: GaussianNB, MultinomialNB, BernoulliNB, KNeighborsClassifier
- scikit-learn metrics: accuracy_score, precision_score, recall_score, f1_score, matthews_corrcoef, confusion_matrix, roc_curve, auc
- matplotlib.pyplot, seaborn

Code for All Variants and Models

Python Implementation

```python
# =====================================
#         Email Spam Classification Notebook
# Models: Naive Bayes, KNN, SVM
# Compatible with Google Colab
# =====================================

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

from sklearn.model_selection import train_test_split, cross_val_score,
    KFold
```

```python
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score,
    recall_score, f1_score, confusion_matrix, roc_curve, auc

# Models
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC

# 1. Load Dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/
    spambase/spambase.data"
dataset = pd.read_csv(url, header=None)
columns = [f"feature_{i}" for i in range(57)] + ["label"]
dataset.columns = columns
print("Dataset shape:", dataset.shape)
print(dataset.head())

# 2. EDA
plt.figure(figsize=(5,4))
sns.countplot(x="label", data=dataset, palette="Set2")
plt.title("Class Distribution (0=Ham, 1=Spam)")
plt.show()
print("\nClass distribution:\n", dataset["label"].value_counts())

# 3. Preprocessing
X = dataset.drop("label", axis=1)
y = dataset["label"]
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train_raw, X_test_raw, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)
X_train_scaled, X_test_scaled, _, _ = train_test_split(
    X_scaled, y, test_size=0.2, stratify=y, random_state=42
)

# 4. Train Models
# --- Naive Bayes Variants ---
results = {}
nb_models = {
    "GaussianNB": (GaussianNB(), X_train_scaled, X_test_scaled),
    "MultinomialNB": (MultinomialNB(), X_train_raw, X_test_raw),
    "BernoulliNB": (BernoulliNB(), X_train_raw, X_test_raw)
}
for name, (model, Xtr, Xte) in nb_models.items():
    model.fit(Xtr, y_train)
    y_pred = model.predict(Xte)
    results[name] = {
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred),
        "Recall": recall_score(y_test, y_pred),
        "F1 Score": f1_score(y_test, y_pred)
    }
nb_df = pd.DataFrame(results).T
print("\  n     Naive Bayes Results:\n", nb_df)

# --- KNN with different k ---
knn_results = []
```

```python
for k in [1, 3, 5, 7]:
    knn = KNeighborsClassifier(n_neighbors=k, algorithm="kd_tree")
    knn.fit(X_train_scaled, y_train)
    y_pred = knn.predict(X_test_scaled)
    knn_results.append([
        k,
        accuracy_score(y_test, y_pred),
        precision_score(y_test, y_pred),
        recall_score(y_test, y_pred),
        f1_score(y_test, y_pred)
    ])
knn_df = pd.DataFrame(knn_results, columns=["k", "Accuracy", "Precision", "Recall", "F1 Score"])
print("\n    KNN Results (varying k):\n", knn_df)

# --- KNN KDTree vs BallTree ---
tree_results = {}
for algo in ["kd_tree", "ball_tree"]:
    start = time.time()
    knn = KNeighborsClassifier(n_neighbors=5, algorithm=algo)
    knn.fit(X_train_scaled, y_train)
    y_pred = knn.predict(X_test_scaled)
    end = time.time()
    tree_results[algo] = {
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred),
        "Recall": recall_score(y_test, y_pred),
        "F1 Score": f1_score(y_test, y_pred),
        "Training Time (s)": round(end-start, 4)
    }
tree_df = pd.DataFrame(tree_results).T
print("\n    KNN KDTree vs BallTree:\n", tree_df)

# --- SVM with different kernels ---
svm_results = []
kernels = {
    "Linear": {"kernel": "linear", "C": 1},
    "Polynomial": {"kernel": "poly", "C": 1, "degree": 3, "gamma": "scale"},
    "RBF": {"kernel": "rbf", "C": 1, "gamma": "scale"},
    "Sigmoid": {"kernel": "sigmoid", "C": 1, "gamma": "scale"}
}
for kernel_name, params in kernels.items():
    start = time.time()
    svm = SVC(**params)
    svm.fit(X_train_scaled, y_train)
    y_pred = svm.predict(X_test_scaled)
    end = time.time()
    svm_results.append([
        kernel_name,
        str(params),
        accuracy_score(y_test, y_pred),
        f1_score(y_test, y_pred),
        round(end-start, 4)
    ])
svm_df = pd.DataFrame(
    svm_results,
    columns=["Kernel", "Hyperparameters", "Accuracy", "F1 Score", "Training Time"]
)
```

```python
print("\  n      SVM Kernel Comparison:\n", svm_df)

# 5. Cross Validation
cv = KFold(n_splits=5, shuffle=True, random_state=42)
cv_results = []
for model_name, (model, X_all) in {
    "Naive Bayes": (GaussianNB(), X_scaled),
    "KNN": (KNeighborsClassifier(n_neighbors=5), X_scaled),
    "SVM": (SVC(kernel="rbf"), X_scaled)
}.items():
    scores = cross_val_score(model, X_all, y, cv=cv, scoring="accuracy")
    for i, score in enumerate(scores):
        cv_results.append([i+1, model_name, score])
cv_df = pd.DataFrame(cv_results, columns=["Fold", "Model", "Accuracy"])
print("\  n      Cross-Validation Results:\n", cv_df.groupby("Model")["Accuracy"].mean())

# 6. Confusion Matrix & ROC
from sklearn.metrics import ConfusionMatrixDisplay

def plot_confusion_and_roc(model, Xtr, Xte, ytr, yte, title):
    model.fit(Xtr, ytr)
    y_pred = model.predict(Xte)
    cm = confusion_matrix(yte, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm)
    disp.plot(cmap="Blues")
    plt.title(f"Confusion Matrix     {title}")
    plt.show()
    if hasattr(model, "predict_proba"):
        y_prob = model.predict_proba(Xte)[:,1]
    else:
        y_prob = model.decision_function(Xte)
    fpr, tpr, _ = roc_curve(yte, y_prob)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f"AUC = {roc_auc:.2f}")
    plt.plot([0,1],[0,1],'r--')
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title(f"ROC Curve     {title}")
    plt.legend()
    plt.show()

plot_confusion_and_roc(GaussianNB(), X_train_scaled, X_test_scaled, y_train, y_test, "GaussianNB")
plot_confusion_and_roc(MultinomialNB(), X_train_raw, X_test_raw, y_train, y_test, "MultinomialNB")
plot_confusion_and_roc(BernoulliNB(), X_train_raw, X_test_raw, y_train, y_test, "BernoulliNB")
plot_confusion_and_roc(KNeighborsClassifier(n_neighbors=1, algorithm="ball_tree"), X_train_scaled, X_test_scaled, y_train, y_test, "KNN (k=1, BallTree)")
plot_confusion_and_roc(KNeighborsClassifier(n_neighbors=1, algorithm="kd_tree"), X_train_scaled, X_test_scaled, y_train, y_test, "KNN (k=1, KDTree)")
```

Listing 1: Email Spam Classification using Naive Bayes, KNN, and SVM

Confusion Matrix and ROC for Each Model

GaussianNB

- Confusion matrix: $\begin{bmatrix} 422 & 136 \\ 17 & 346 \end{bmatrix}$
- ROC AUC: 0.9449
- Test metrics: Accuracy: 0.833, Precision: 0.715, Recall: 0.959, F1: 0.819, MCC: 0.674

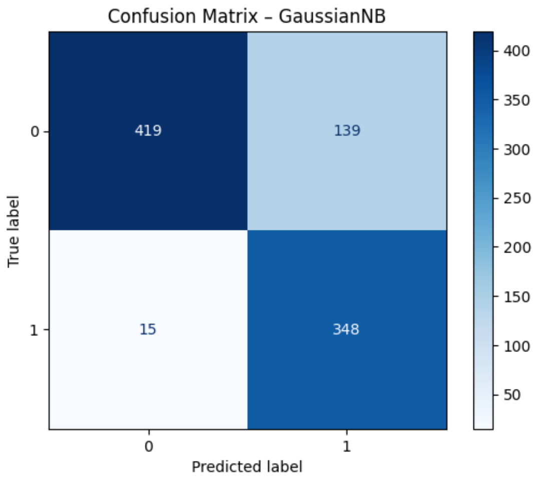

Figure 1: Confusion Matrix – GaussianNB

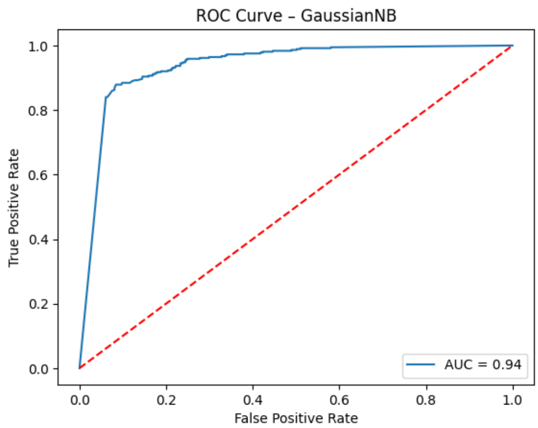

Figure 2: ROC Curve – GaussianNB

MultinomialNB

- Confusion matrix: $\begin{bmatrix} 458 & 100 \\ 106 & 257 \end{bmatrix}$
- ROC AUC: 0.8248
- Test metrics: Accuracy: 0.776, Precision: 0.720, Recall: 0.708, F1: 0.714, MCC: 0.560

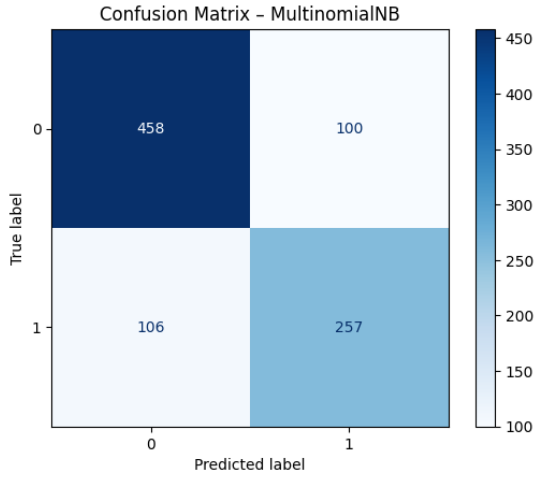

Figure 3: Confusion Matrix – MultinomialNB

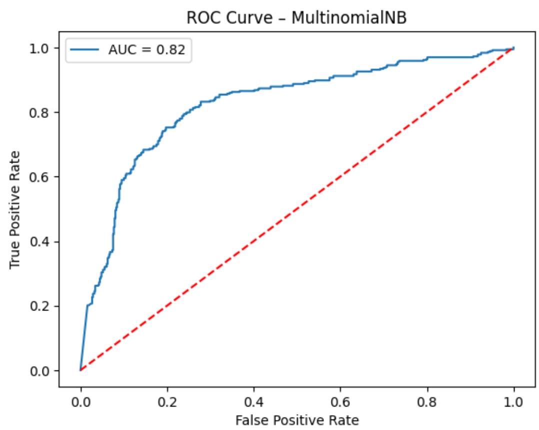

Figure 4: ROC Curve – MultinomialNB

BernoulliNB

- Confusion matrix: $\begin{bmatrix} 515 & 43 \\ 71 & 292 \end{bmatrix}$
- ROC AUC: 0.9496
- Test metrics: Accuracy: 0.876, Precision: 0.872, Recall: 0.804, F1: 0.837, MCC: 0.762

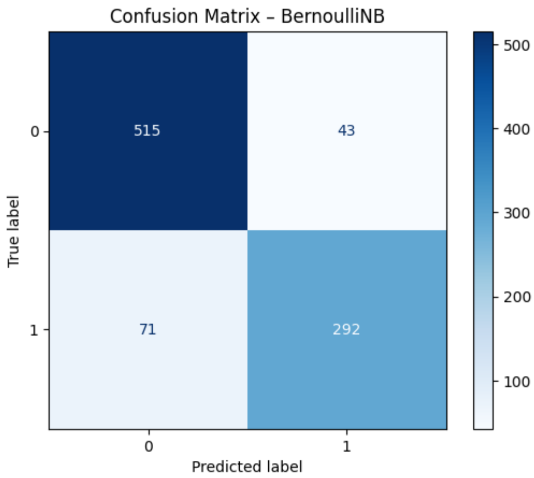

Figure 5: Confusion Matrix – BernoulliNB

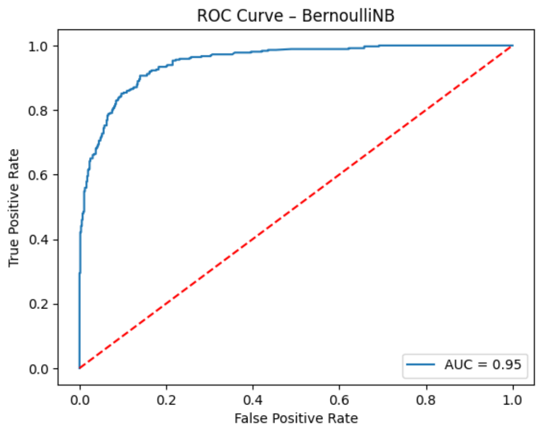

Figure 6: ROC Curve – BernoulliNB

KNN ($k = 1$, BallTree)

- Confusion matrix: $\begin{bmatrix} 514 & 44 \\ 50 & 313 \end{bmatrix}$
- ROC AUC: 0.8917
- Test metrics: Accuracy: 0.898, Precision: 0.877, Recall: 0.862, F1: 0.869, MCC: 0.816

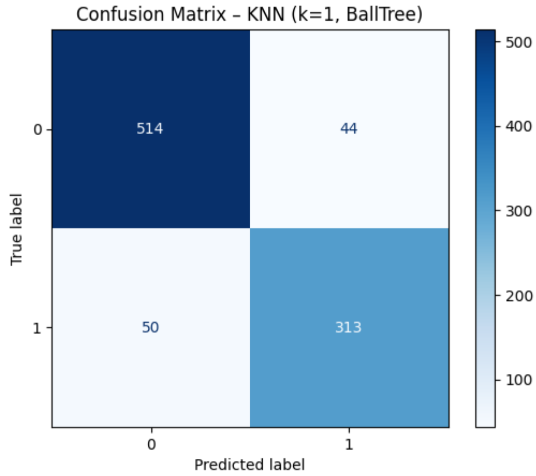

Figure 7: Confusion Matrix – KNN ($k = 1$, BallTree)

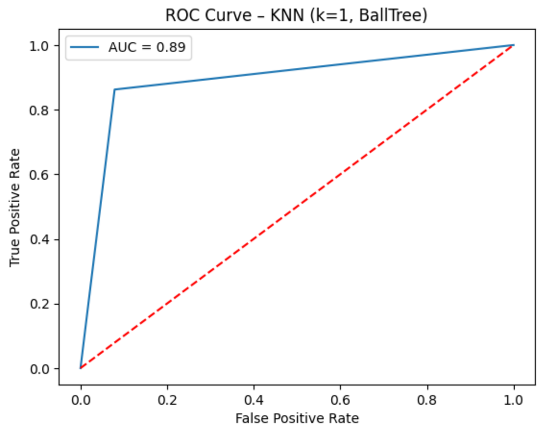

Figure 8: ROC Curve – KNN ($k = 1$, BallTree)

KNN ($k = 1$, KDTree)

- Confusion matrix: $\begin{bmatrix} 514 & 44 \\ 50 & 313 \end{bmatrix}$
- ROC AUC: 0.8917
- Test metrics: Accuracy: 0.898, Precision: 0.877, Recall: 0.862, F1: 0.869, MCC: 0.816

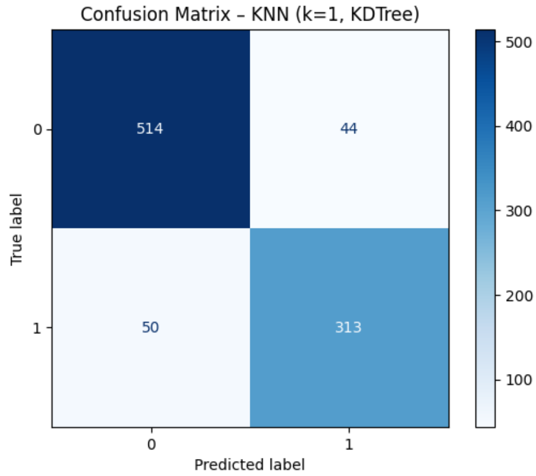

Figure 9: Confusion Matrix – KNN ($k = 1$, KDTree)

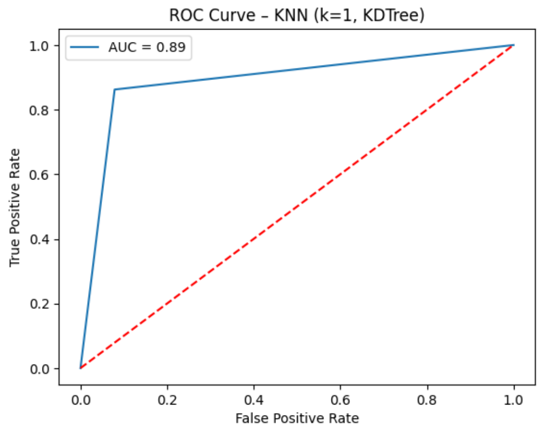

Figure 10: ROC Curve – KNN ($k = 1$, KDTree)

All Comparison Tables

Table 1: Performance Comparison of Naïve Bayes Variants

Metric	Gaussian NB	Multinomial NB	Bernoulli NB
Accuracy	0.833	0.776	0.876
Precision	0.715	0.720	0.872
Recall	0.959	0.708	0.804
F1 Score	0.819	0.714	0.837
MCC	0.674	0.560	0.762
ROC AUC	0.9449	0.8248	0.9496

Table 1: Test set performance for all Naïve Bayes variants.

Table 2: Best KNN ($k = 1$) Performance with BallTree and KDTree

Metric	BallTree	KDTree
Accuracy	0.898	0.898
Precision	0.877	0.877
Recall	0.862	0.862
F1 Score	0.869	0.869
MCC	0.816	0.816
ROC AUC	0.8917	0.8917

Table 2: Comparison of best KNN variant ($k = 1$) using BallTree and KDTree algorithms.

Table 3: Comparing Training Time

KD Tree	Ball Tree
0.1195 seconds	0.1177 seconds

Table 3: KD Tree vs Ball Tree Training time

Table 4: KNN: Varying k values

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

Table 4: KNN Performance for Different k Values

Observations and Conclusions

- **Best Overall Classifier:** KNN with $k = 7$ achieved the highest accuracy and F1 score.
- **Naïve Bayes:** BernoulliNB outperformed other Naïve Bayes variants, but KNN still surpassed it.
- **BallTree vs KDTree:** BallTree and KDTree produced identical results on accuracy and metrics, with BallTree training slightly faster.
- **Class Imbalance:** All models handled the class imbalance well, reflected by high recall and MCC.