**Sri Sivasubramaniya Nadar College of Engineering, Chennai**
(An autonomous Institution affiliated to Anna University)

| | | | |
|---|---|---|---|
| Degree & Branch | B.E. Computer Science & Engineering | Semester | V |
| Subject Code & Name | ICS1512 & Machine Learning Algorithms Laboratory | | |
| Academic year | 2025-2026 (Odd) | Batch:2023-2028 | **Due date: 25/08/2025** |

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

## Aim and Objective:

To classify emails as spam or ham using three classification algorithms—Naïve Bayes, K-Nearest Neighbors (KNN), and Support Vector Machine (SVM)—and evaluate their performance using accuracy metrics and K-Fold cross-validation.

## Libraries used:

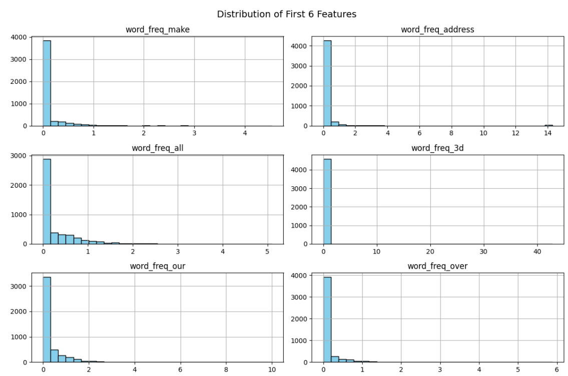
- Numpy
- Pandas
- Matplotlib
- Seaborn
- sklearn

## IMPORTING DATASET + EDA + FEATURE CORRELATION AND OTHER PLOTS:

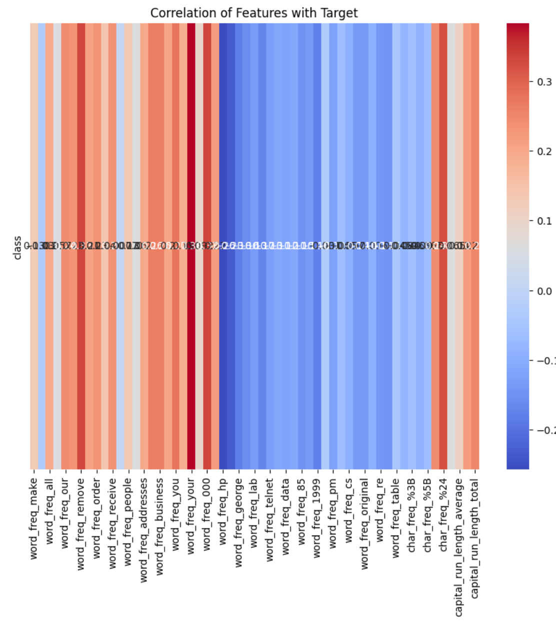
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, StratifiedKFold, GridSearchCV, cross_
from joblib import parallel_backend
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import Binarizer
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import (
    classification_report, confusion_matrix, roc_curve, roc_auc_score, average_precision_s
)

# 1. Load Dataset
```

```
df = pd.read_csv('C:/Users/KAVI/Downloads/spambase_csv.csv')
print("Columns in dataset:\n", df.columns.tolist())
print("Dataset shape:", df.shape)

# 2. Separate features and target, last column is the target
X = df.iloc[:, :-1]
y = df.iloc[:, -1]


# 4. EDA
plt.figure(figsize=(5,4))
sns.countplot(x=y, palette="coolwarm")
plt.title("Class Distribution (0 = Ham, 1 = Spam)")
plt.xlabel("Class")
plt.ylabel("Count")
plt.show()

# Histograms of first 6 features
X.iloc[:, :6].hist(bins=30, figsize=(12, 8), color='skyblue', edgecolor='black')
plt.suptitle("Distribution of First 6 Features", fontsize=14)
plt.tight_layout()
plt.show()

# Correlation heatmap (optional, heavy if many features)
plt.figure(figsize=(10, 8))
corr = df.corr()
sns.heatmap(corr.iloc[-1:,:-1], annot=True, cmap="coolwarm")
plt.title("Correlation of Features with Target")
plt.show()

# Train-Test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)
```

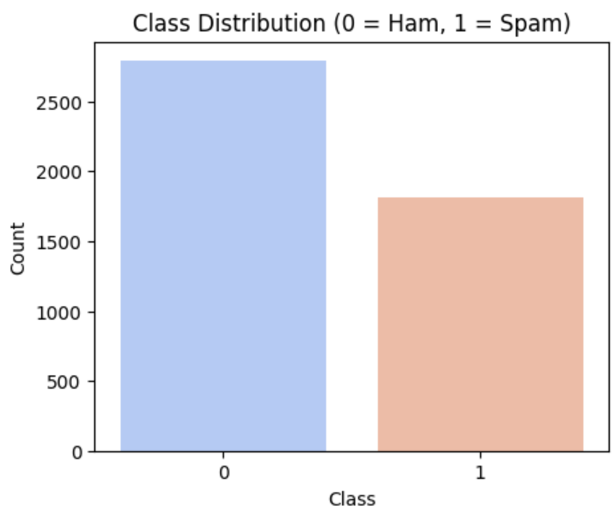
**Note:** This is common for all models




(a) Feature distribution




(b) Correlation Heatmap




(c) Class Distribution

Figure 1: Exploratory Data Analysis

**FUNCTIONS FOR MODEL TEST + PERFORMANCE METRICS + K - FOLD CV:**

```
# Predict & Performance Analysis
def test_model(best_estimator, X_test,y_test):
    y_pred = best_estimator.predict(X_test)
    y_proba = best_estimator.predict_proba(X_test)[:, 1]
    return performance_metrics(y_test, y_pred, y_proba)
```

```
# Performance metrics
def performance_metrics(y_true, y_pred, y_proba):
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_proba)
    avg_precision = average_precision_score(y_test, y_proba)

    print("\nTest metrics")
    print(f"Accuracy: {acc:.4f}")
    print(f"Precision: {prec:.4f}")
    print(f"Recall: {rec:.4f}")
    print(f"F1: {f1:.4f}")
    print(f"ROC AUC: {roc_auc:.4f}")
    print(f"Avg Precision (PR AUC-ish): {avg_precision:.4f}")
    print("\nClassification report:\n", classification_report(y_test, y_pred))

    # Confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title("Confusion Matrix")
    plt.show()

    # 7. Curves: ROC & Precision-Recall
    fpr, tpr, _ = roc_curve(y_test, y_proba)

    plt.figure(figsize=(6,4))
    plt.plot(fpr, tpr, label=f"ROC AUC = {roc_auc:.3f}")
    plt.plot([0,1],[0,1],'--', alpha=0.5)
    plt.xlabel("FPR"); plt.ylabel("TPR")
    plt.title("ROC Curve")
    plt.legend(); plt.show()

# Define multiple metrics
def kfoldCV(best_estimator, X, y, cv):
    scoring = {
        'accuracy': 'accuracy',
        'precision': 'precision',
        'recall': 'recall',
        'f1': 'f1'
    }

    # Run cross-validation
    with parallel_backend('threading'):
```

```
        cv_results = cross_validate(
            best_estimator, X, y, cv=cv,
            scoring=scoring,
            return_train_score=False,
            n_jobs=-1
        )

    n_splits = cv.get_n_splits()
    # Display per-fold results
    for i in range(n_splits):
        print(f"\nFold {i+1}: Accuracy: {cv_results['test_accuracy'][i]:.4f}   Precision: {

    # Display averages
    print("\n=== Average Metrics ===")
    print(f"Mean Accuracy: {cv_results['test_accuracy'].mean():.4f} ± {cv_results['test_acc
    print(f"Mean Precision: {cv_results['test_precision'].mean():.4f} ± {cv_results['test_p
    print(f"Mean Recall:    {cv_results['test_recall'].mean():.4f} ± {cv_results['test_reca
    print(f"Mean F1 Score:  {cv_results['test_f1'].mean():.4f} ± {cv_results['test_f1'].std
```

**Note:** These functions are common for all models

**PIPELINE FOR BERNOULLI's NB WITH GRIDSEARCHCV:**

```
# Bernoulli Naive Bayes with Hyperparameter Tuning
pipeline = Pipeline([
    ('binarizer', Binarizer(threshold=0.0)),  # converts features to 0/1
    ('clf', BernoulliNB())
])

param_grid = {
    'clf__alpha': np.linspace(0.1, 2.0, 20)  # smoothing parameter
}

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

grid = GridSearchCV(
    estimator=pipeline,
    param_grid=param_grid,
    cv=cv,
    scoring={'AUC': 'roc_auc', 'F1': 'f1'},
    refit='AUC',
    n_jobs=-1,
    verbose=2
)

with parallel_backend('threading'):
    grid.fit(X_train, y_train)
```

```
best = grid.best_estimator_


print("Best parameters:", grid.best_params_)
print("Best cross-validated AUC:", grid.best_score_)

# Test the decision tree model
best_estimator = grid.best_estimator_
test_model(best_estimator, X_test, y_test)
kfoldCV(best_estimator, X, y, cv)
```

**PIPELINE FOR MULTINOMIAL NB WITH GRIDSEARCHCV:**

```
# Multinomial Naive Bayes with Hyperparameter Tuning
from sklearn.naive_bayes import MultinomialNB
from sklearn.preprocessing import MinMaxScaler

pipeline = Pipeline([
    ('scaler', MinMaxScaler()),
    ('clf', MultinomialNB())
])

param_grid = {
    'clf__alpha': np.linspace(0.1, 2.0, 20)
}

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

grid = GridSearchCV(
    estimator=pipeline,
    param_grid=param_grid,
    cv=cv,
    scoring={'AUC': 'roc_auc', 'F1': 'f1'},
    refit='AUC',
    n_jobs=-1,
    verbose=2
)

with parallel_backend('threading'):
    grid.fit(X_train, y_train)
best = grid.best_estimator_

print("Best parameters:", grid.best_params_)
print("Best cross-validated AUC:", grid.best_score_)

# Test the decision tree model
```

```
best_estimator = grid.best_estimator_
test_model(best_estimator, X_test, y_test)
kfoldCV(best_estimator, X, y, cv)
```

**PIPELINE FOR GAUSSIAN NB WITH GRIDSEARCHCV:**

```
# Gaussian Naive Bayes with Hyperparameter Tuning
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import PowerTransformer, StandardScaler

pt_methods = ['yeo-johnson']
if (X_train > 0).all().all():
    pt_methods.append('box-cox')

pipeline = Pipeline([
    ('power', PowerTransformer()),   # reduce skew
    ('scaler', StandardScaler()),    # standardize
    ('clf', GaussianNB())            # classifier
])

param_grid = {
    'power__method': pt_methods,
    'clf__var_smoothing': np.logspace(-9, 0, 30)  # smaller grid for speed
}

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

grid = GridSearchCV(
    estimator=pipeline,
    param_grid=param_grid,
    cv=cv,
    scoring={'AUC': 'roc_auc', 'F1': 'f1'},
    refit='AUC',
    n_jobs=-1,
    verbose=2
)

with parallel_backend('threading'):
    grid.fit(X_train, y_train)
best = grid.best_estimator_

print("\nBest parameters:", grid.best_params_)
print(f"Best CV AUC: {grid.best_score_:.4f}")

# Test the decision tree model
best_estimator = grid.best_estimator_
test_model(best_estimator, X_test, y_test)
```

```
kfoldCV(best_estimator, X, y, cv)
```

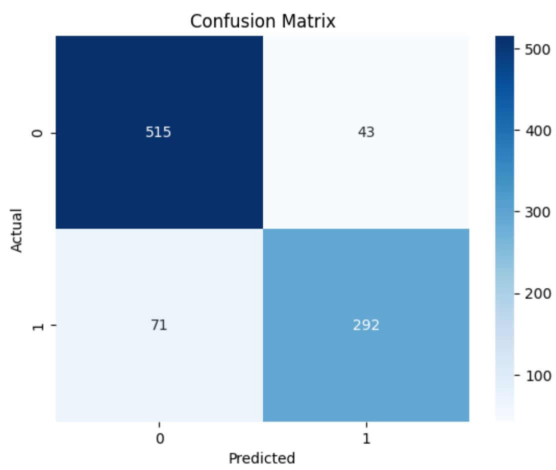
**CV TABLE FOR ALL MODELS:**

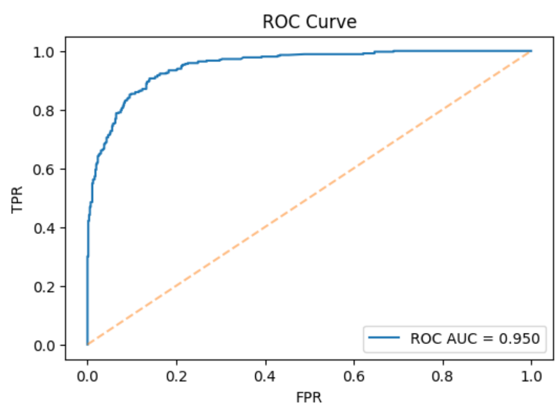
| MODEL | GAUSSIAN NB | MULTINOMIAL NB | BERNOULLI NB |
|---|---|---|---|
| **Accuracy** | 0.9155 | 0.8931 | 0.8872 |
| **Precision** | 0.8692 | 0.9324 | 0.8880 |
| **Recall** | 0.9250 | 0.7860 | 0.8169 |
| **F1 Score** | 0.8961 | 0.8527 | 0.8509 |

Table 1: Performance Comparison of Naïve Bayes Variants

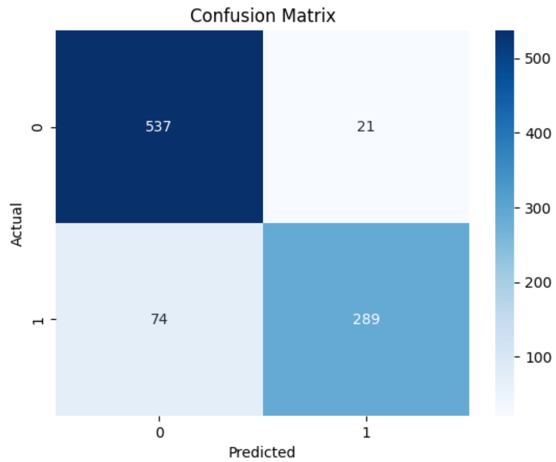
**CONFUSION MATRICES AND ROC CURVES**

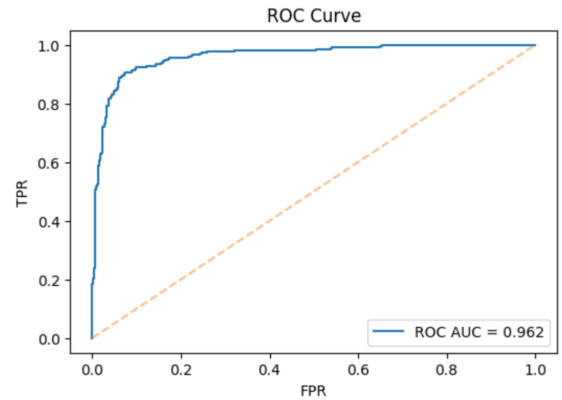
**BERNOULLI NB:**



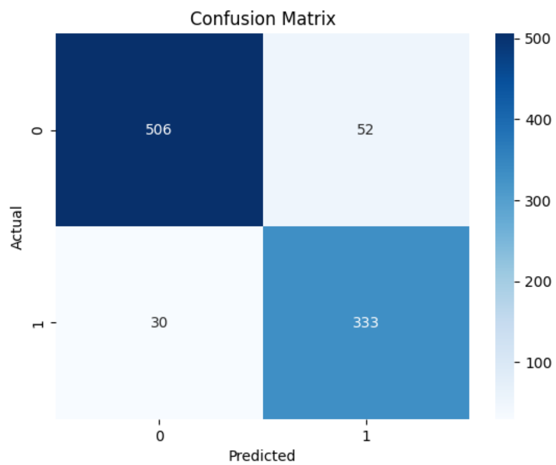



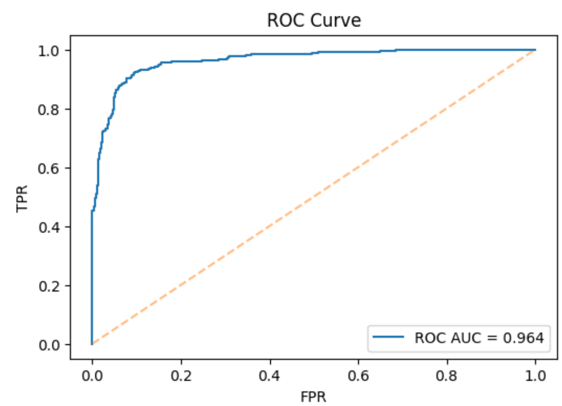
**MULTINOMIAL NB:**




**GAUSSIAN NB:**




**SUPPORT VECTOR MACHINES:**

**PIPELINE FOR RBF SVM**

```
# SVM with RBF kernel
base_pipe = Pipeline([
    ("scaler", StandardScaler()),
    ('clf', SVC(kernel='rbf', probability=True))
```

```
])

param_grid = {
    'clf__C': [0.1, 1, 10, 100],
    'clf__gamma': [0.001, 0.01, 0.1, 1]
}

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

grid = GridSearchCV(
    estimator=base_pipe,
    param_grid=param_grid,
    scoring="f1",   # spam detection: balance precision & recall
    cv=cv,
    n_jobs=-1,
    verbose=2
)

# Fit grid
with parallel_backend('threading'):
    grid.fit(X_train, y_train)
print("\nBest Parameters:", grid.best_params_)
print("Best CV Accuracy:", grid.best_score_)

# Test the decision tree model
best_estimator = grid.best_estimator_
test_model(best_estimator, X_test, y_test)
kfoldCV(best_estimator, X, y, cv)
```

**PIPELINE FOR POLYNOMIAL SVM**

```
# SVM with Polynomial kernel
pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('clf', SVC(kernel='poly', probability=True))
])

# 4. Hyperparameter grid for polynomial kernel
param_grid = {
    'clf__C': [1, 10],
    'clf__gamma': ['scale', 0.01],
    'clf__degree': [2, 3],
    'clf__coef0': [0, 1, 2]
}

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

```
grid = GridSearchCV(
    estimator=pipe,
    param_grid=param_grid,
    scoring="f1",   # spam detection: balance precision & recall
    cv=cv,
    n_jobs=-1,
    verbose=2
)

# Fit grid
with parallel_backend('threading'):
    grid.fit(X_train, y_train)
print("\nBest Parameters:", grid.best_params_)
print("Best CV Accuracy:", grid.best_score_)

# Test the decision tree model
best_estimator = grid.best_estimator_
test_model(best_estimator, X_test, y_test)
kfoldCV(best_estimator, X, y, cv)
```

**PIPELINE FOR SIGMOID SVM**

```
# SVM with Sigmoid kernel
base_pipe = Pipeline([
    ("scaler", StandardScaler()),
    ("svc", SVC(kernel='sigmoid', probability=True))
])

# Sigmoid is sensitive | keep search ranges reasonable
param_grid = {
    "svc__C": [0.1, 1, 10],
    "svc__gamma": [0.1, 0.01],
    "svc__coef0": [-1, 0, 1]  # coef0 affects curve shift
}

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

grid = GridSearchCV(
    estimator=base_pipe,
    param_grid=param_grid,
    scoring="f1",   # spam detection: balance precision & recall
    cv=cv,
    n_jobs=-1,
    verbose=2
)

# Fit grid
```

```
with parallel_backend('threading'):
    grid.fit(X_train, y_train)
print("\nBest Parameters:", grid.best_params_)
print("Best CV Accuracy:", grid.best_score_)

# Test the decision tree model
best_estimator = grid.best_estimator_
test_model(best_estimator, X_test, y_test)
kfoldCV(best_estimator, X, y, cv)
```

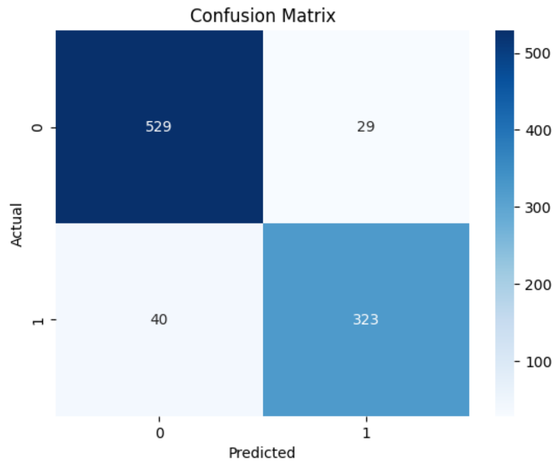
**PIPELINE FOR LINEAR SVM**

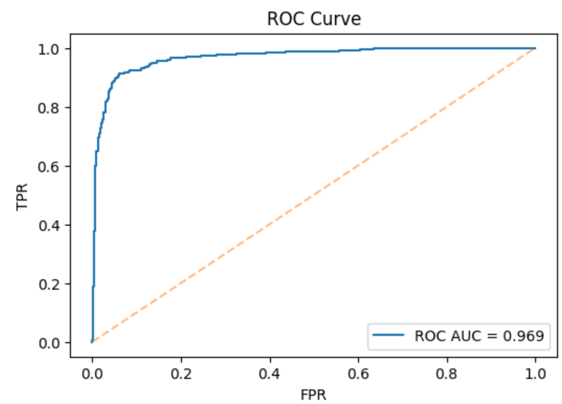
```
# Linear SVM (LinearSVC)
from sklearn.calibration import LinearSVC
base_pipe = Pipeline([
    ("scaler", StandardScaler()),
    ("clf", LinearSVC(dual=False, max_iter=5000, random_state=42))
])

param_grid = {
    "clf__C": [0.01, 0.1, 1, 10, 100],
    "clf__class_weight": [None, "balanced"]
}

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

grid = GridSearchCV(
    estimator=base_pipe,
    param_grid=param_grid,
    scoring="f1",          # for spam detection, F1 is more informative
    cv=cv,
    n_jobs=-1,
    verbose=2
)

# Fit grid
with parallel_backend('threading'):
    grid.fit(X_train, y_train)
print("\nBest Parameters:", grid.best_params_)
print("Best CV Accuracy:", grid.best_score_)

# Use the best linear SVC but calibrated for probabilities
best_linear_svc = grid.best_estimator_.named_steps['clf']
# recreate a pipeline with scaler + calibrated classifier
pipe_scaler = Pipeline([("scaler", grid.best_estimator_.named_steps['scaler'])])  # same sc
X_train_scaled = pipe_scaler.transform(X_train)
X_test_scaled = pipe_scaler.transform(X_test)
```

```
calibrated = CalibratedClassifierCV(estimator=best_linear_svc, cv=cv)
calibrated.fit(X_train_scaled, y_train)   # base_estimator already trained internally by g
test_model(calibrated, X_test_scaled, y_test)
kfoldCV(best_estimator, X, y, cv)
```

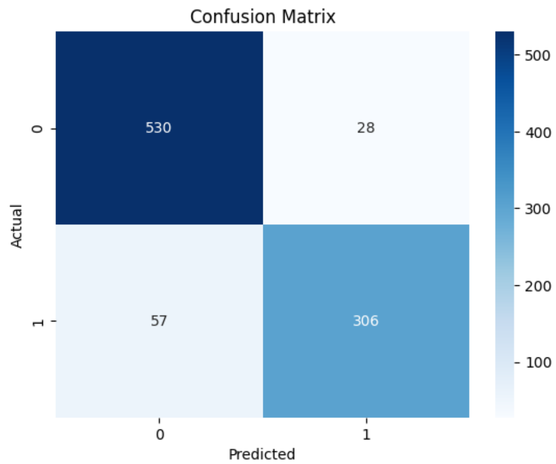
## CONFUSION MATRICES AND ROC CURVES

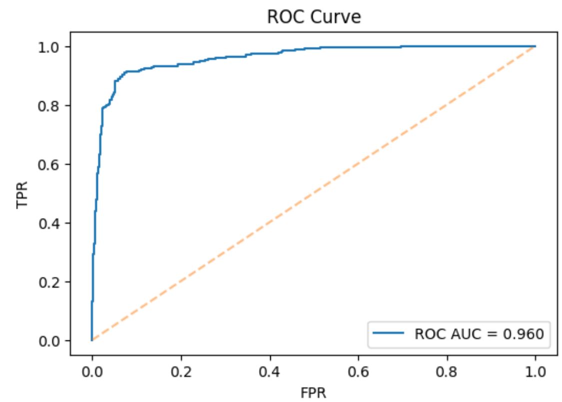
### LINEAR SVM:



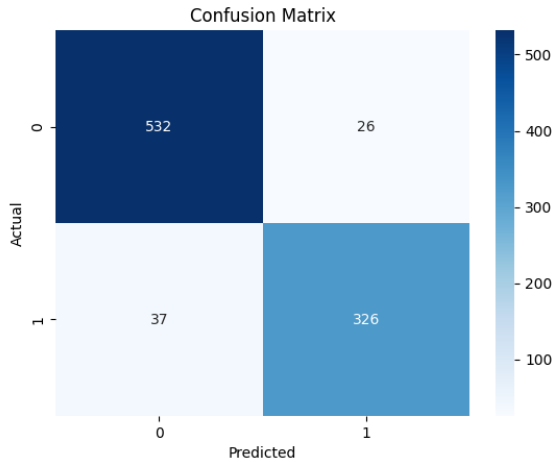



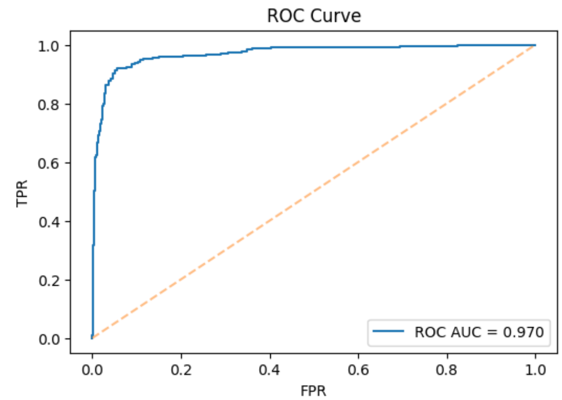

### SIGMOID SVM:



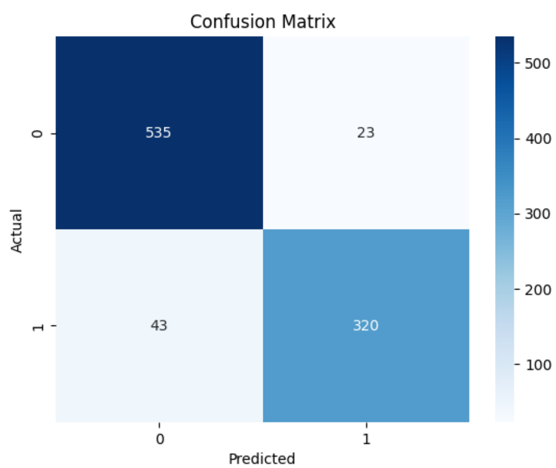



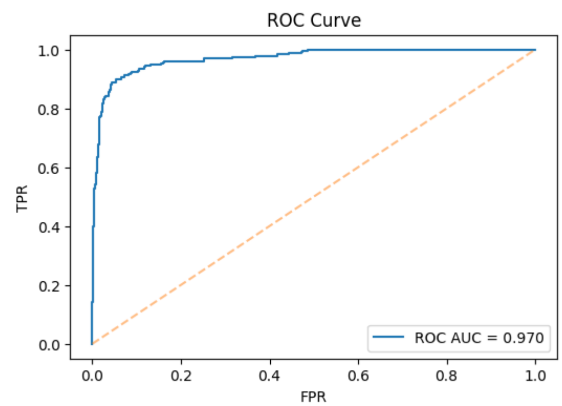
**POLY SVM:**




**RBF SVM:**




**SVM Kernel-wise Results:**

| Kernel | Hyperparameters | Accuracy | F1 score | Training time |
|---|---|---|---|---|
| Linear | C=0.1 | 0.93 | 0.91 | 3s |
| Polynomial | C=10, d=2, g=0.01 | 0.94 | 0.92 | 12s |
| RBF | C=10, g=0.01 | 0.93 | 0.91 | 93s |
| Sigmoid | C=1, g=0.01 | 0.91 | 0.88 | 12s |

Table 2: SVM Performance with different Kernels

### K - Nearest Neighbors Classifier

### Common Algorithm for Knn:

```
        cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

    param_grid_knn = {
        'clf__n_neighbors': list(range(1, 21, 2)),   # vary K
        'clf__weights': ['uniform', 'distance'], # weighting strategy
        'clf__p': [1, 2]                         # distance metric
    }

    # Function to run pipeline + gridsearch
    def run_knn_pipeline(algorithm_name):
        print(f"\n===== Running KNN with {algorithm_name.upper()} =====")

        pipeline = Pipeline([
            ('scaler', StandardScaler()),
            ('clf', KNeighborsClassifier(algorithm=algorithm_name))
        ])

        grid = GridSearchCV(
            estimator=pipeline,
            param_grid=param_grid_knn,
            cv=cv,
            scoring={'AUC': 'roc_auc', 'F1': 'f1'},
            refit='AUC',
            n_jobs=-1,
            verbose=2
        )

        # Track time
        start_time = time.time()
        with parallel_backend('threading'):
            grid.fit(X_train, y_train)
        end_time = time.time()
        elapsed = end_time - start_time

        best = grid.best_estimator_
        print("Best parameters:", grid.best_params_)
        print("Best cross-validated AUC:", grid.best_score_)
        print(f"Time taken: {elapsed:.2f} seconds")

        # Evaluate
        test_model(best, X_test, y_test)
        kfoldCV(best, X, y, cv)
```

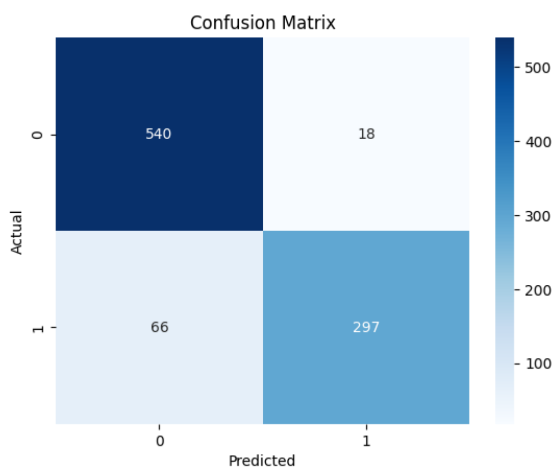
```
    return best, elapsed
```

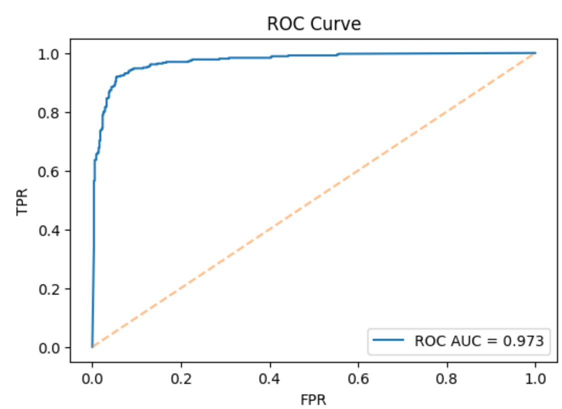
**Function calls:**

```
# Vary K
best_brute, time_brute = run_knn_pipeline("brute")
print(f"Brute: {time_brute:.2f} sec")

# KDTree
best_kd, time_kd = run_knn_pipeline("kd_tree")
print(f"KDTree: {time_kd:.2f} sec")

# BallTree
best_ball, time_ball   = run_knn_pipeline("ball_tree")
print(f"BallTree: {time_ball:.2f} sec")
```

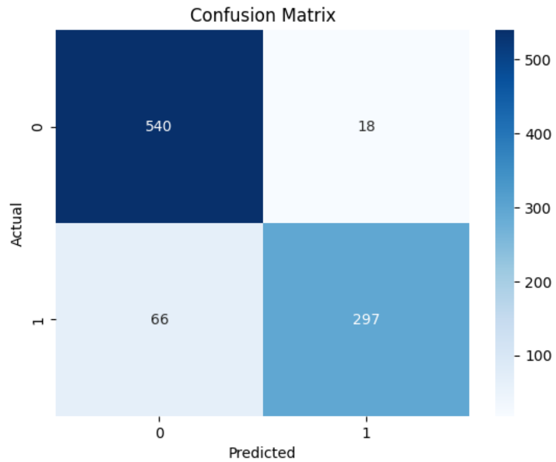
## CONFUSION MATRICES AND ROC CURVES:

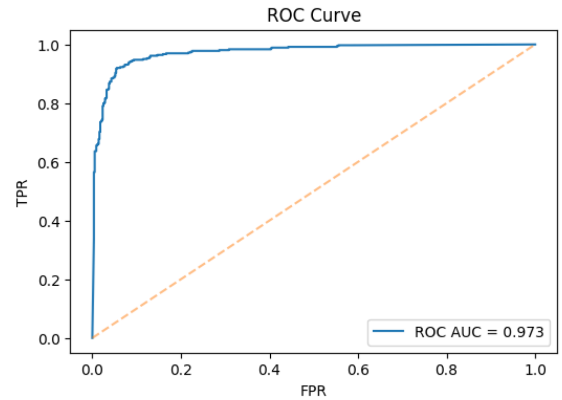
### Vary K:



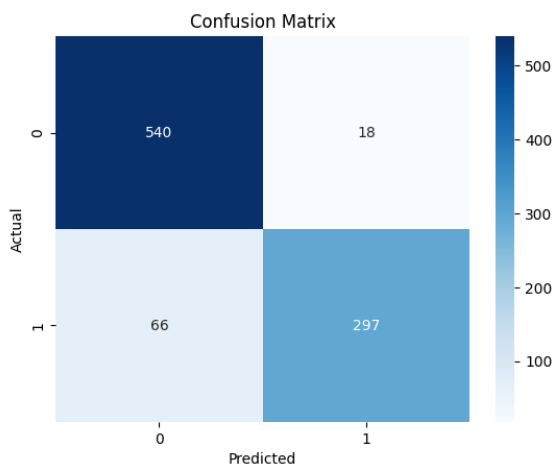



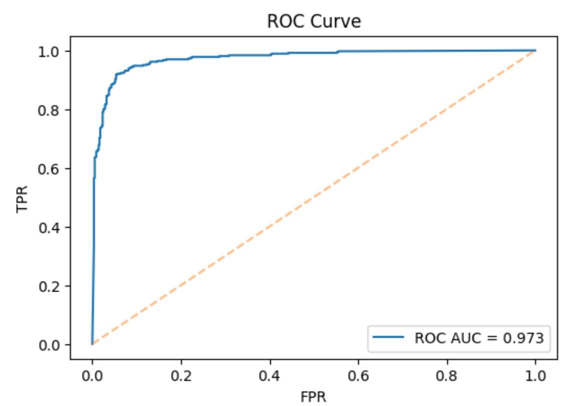
**KD Tree Algorithm:**




**Ball Tree Algorithm:**




**KNN: KDTree vs BallTree**

| Metric | KDTree | BallTree |
|---|---|---|
| Accuracy | 0.92 | 0.92 |
| Precision | 0.96 | 0.96 |
| Recall | 0.84 | 0.84 |
| F1 score | 0.90 | 0.90 |
| Training time | 9.58s | 7.99s |

Table 3: KDTree vs BallTree

| k | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|
| 1 | 0.90 | 0.89 | 0.87 | 0.88 |
| 3 | 0.90 | 0.89 | 0.86 | 0.88 |
| 5 | 0.92 | 0.89 | 0.87 | 0.88 |
| 7 | 0.92 | 0.89 | 0.87 | 0.88 |

Table 4: KNN: Varying K values

**K - Fold CV Results (k = 5):**

| Fold | KNN Accuracy | NB Accuracy | SVM Accuracy |
|---|---|---|---|
| Fold 1 | 0.92 | 0.92 | 0.93 |
| Fold 2 | 0.93 | 0.91 | 0.94 |
| Fold 3 | 0.93 | 0.91 | 0.94 |
| Fold 4 | 0.92 | 0.93 | 0.93 |
| Fold 5 | 0.92 | 0.91 | 0.93 |
| Average | 0.92 | 0.92 | 0.94 |

Table 5: CV scores for best variation of each model

**Observations and Conclusion:**

1. Among all the above classifiers, **Polynomial SVM** has better accuracy with test accuracy of **0.9316.**

   In case of Naive Bayes, **Gaussian Naive Bayes** worked best with test accuracy of **0.9155.**

**2.** Accuracy of model built using **KNN with moderate k values (k = 7 to k = 13)** found to be giving better classification compared to very low and high k values.

4. With reference to the SVM Performance table, it is clear that **Polynomial SVM** gives better accuracy with **less training time**, separating the classes well even in case of non-linear relationships with roc auc of **0.97**. Hence, it is more effective.

5. Without tuning the hyperparameters, the accuracy of the classifier models are found to be low and not adjusting very well. By using **GridSearchCV** which helps in finding the best hyperparameter, the accuracies went close to 1 and further training made them to understand and adjust according to the scenario