Pattern Recognition HW1

110550085 房天越

1. Introduction

In this homework, we implemented two classifiers, Gaussian Naïve Bayes (GNB) and k-Nearest Neighbors (k-NN). Then, we selected two binary-class dataset and two multiclass datasets to evaluate how the classifiers perform under different data characteristics.

2. Methods I Have Implemented

A. Dataset Selection

I. Binary-class Datasets

- Breast Cancer Dataset

  This dataset is from UCI Machine Learning Repository, which is available to use via scikit-learn.

  30 features are available to be used.

  569 samples are split into 398 training and 171 testing.

- Synthetic Binary Dataset

  This dataset is generated using make_classification.

  There are 20 features, 15 of them are informative, 5 of them are redundant.

  300 samples are split into 210 training and 90 testing.

II. Multiclass Datasets

Both of the datasets are from UCI Machine Learning Repository.

- Iris Dataset

  There are 4 features and 3 classes.

  150 samples are split into 105 training and 45 testing.

- Wine Dataset

  There are 13 features and 3 classes.

  178 samples are split into 124 training and 54 testing.

B. Methods

I. Gaussian Naïve Bayes (GNB)

In the training phase, we compute the prior probability for each class

and estimates the mean and variance of each feature assuming Gaussian distribution.

In the prediction phase, we calculate the log-posterior probability for each class based on test samples, then select the class with the highest probability. The log probabilities are exponentiated for ROC evaluation.

II. K-Nearest Neighbors (k-NN)

First, we store all training samples (No learning required).

Then, in the prediction phase, for each test sample, we calculate its Euclidean distance to all training samples, select the k nearest neighbors, and classify based on majority vote.

For binary classification the proportion of votes for the positive class is used as a probability estimate.

3. Experiments I Have Done, and the Results

A. Confusion Matrix

For each dataset and classifier, we display the confusion matrix to illustrate the classification performance, including false positives and false negatives.

B. ROC Curve and AUC

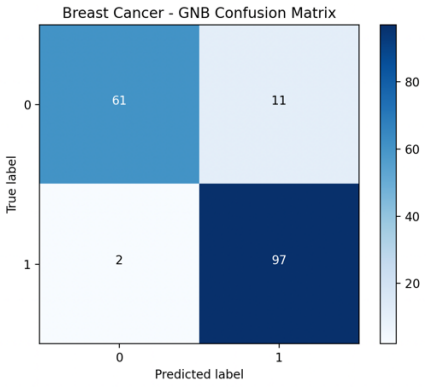
For binary datasets, we plot the ROC curve using the probability scores and compute the area under the curve (AUC) to quantify classification performance across thresholds.

C. Results

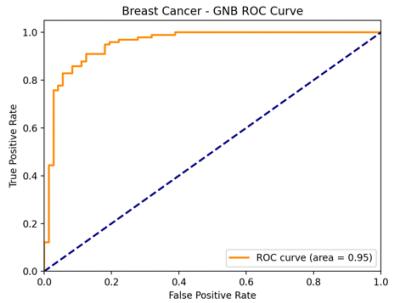
I. Breast Cancer

- GNB

Confusion Matrix:



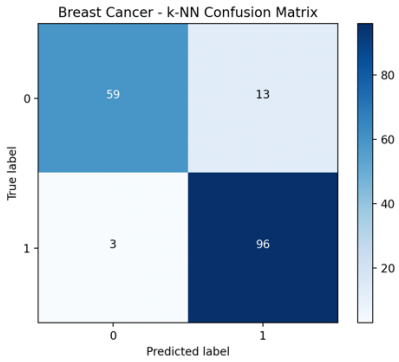
ROC Curve:



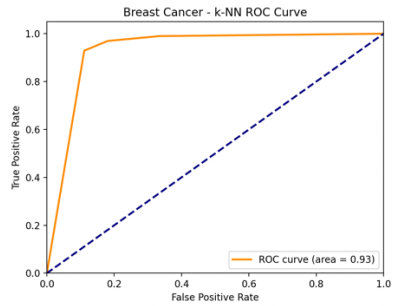
AUC: 0.953

- k-NN

Confusion Matrix:



ROC Curve:

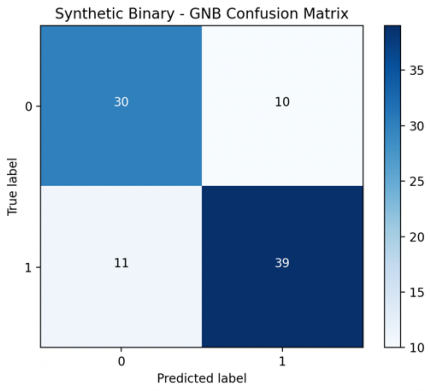


AUC: 0.931

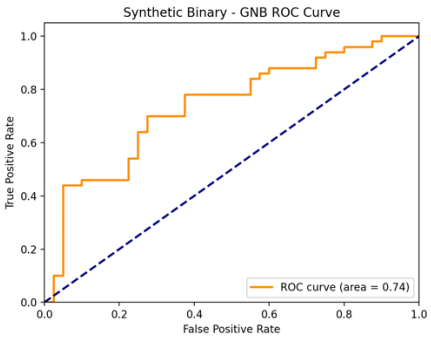
II. Synthetic Binary

- GNB

Confusion Matrix:



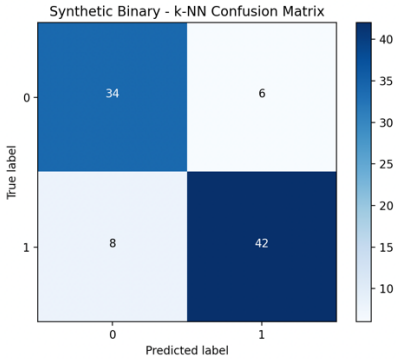
ROC Curve:



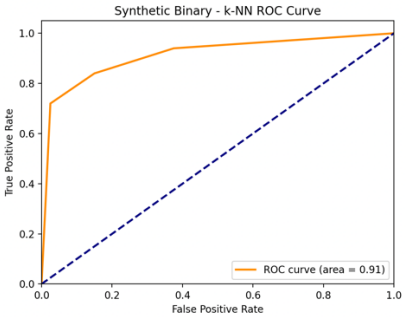
AUC: 0.737

- k-NN

Confusion Matrix:



ROC Curve:

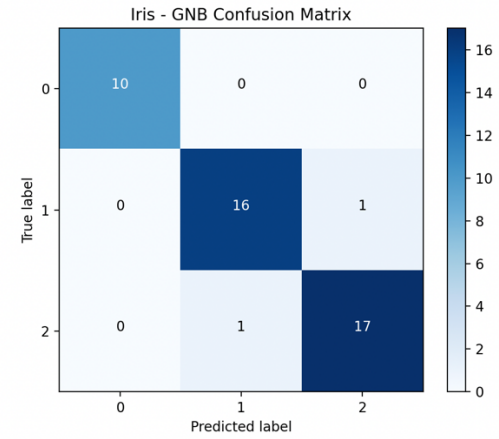


AUC: 0.913

III. Iris

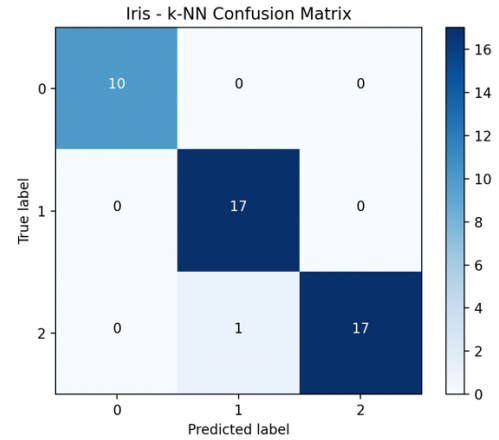
- GNB

  Confusion Matrix:



- k-NN

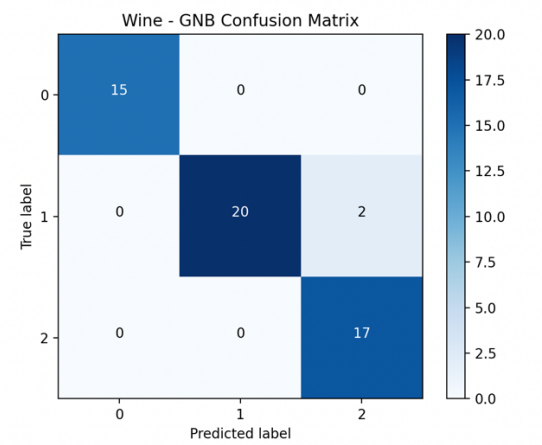
  Confusion Matrix:



IV. Wine

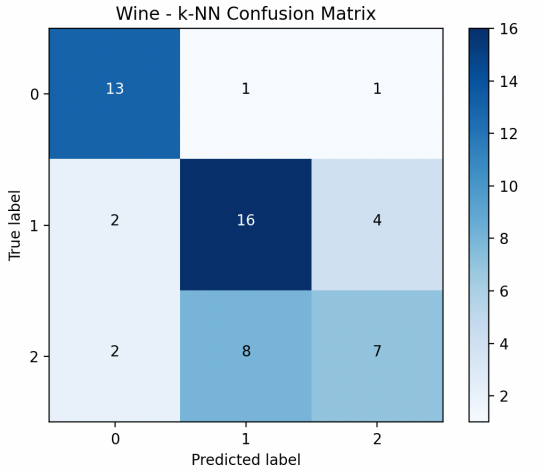
- GNB

  Confusion Matrix:



- k-NN

  Confusion Matrix:



4. Analysis

Let's analyze the results for the four datasets.

A. Breast Cancer

I. Expectation

GNB often excels when features roughly follow Gaussian distributions and classes are well separated.

II. Observation

GNB's AUC of 0.953 slightly outperforms k-NN's AUC of 0.931, and GNB yields fewer false negatives.

III. Conclusion

Matches expectation that Gaussian assumption holds reasonably well, and minimizing false negatives is critical in medical diagnosis.

B. Synthetic Binary

I. Expectation

A synthetic dataset with correlated or complex boundaries can violate Naïve Bayes' independence assumption, favoring non-parametric methods.

II. Observation

GNB underperforms with AUC of 0.737, while k-NN achieves a strong AUC of 0.913.

III. Conclusion

Exactly as suspected, k-NN's flexibility captures the underlying structure better than GNB's rigid Gaussian model.

C. Iris

I. Expectation

A classic, well-behaved dataset; both classifiers perform near perfectly.

II. Observation

GNB has 2 errors, k-NN only 1.

III. Conclusion

Consistent with theory, simple and low-dimensional data where both methods work well.

D. Wine

I. Expectation

Wine features generally conform to Gaussian distributions after normalization, so GNB should excel.

II. Observation

GNB misclassifies only 2 samples; k-NN struggles with 18 errors.

III. Conclusion

Matches theory that GNB's parametric model fits the data well, while k-NN's fixed-k strategy struggles to capture inter-class nuances.

Conclusion:

The effectiveness of a model is fundamentally tied to the validity of its underlying assumptions. When these assumptions are met, parametric methods such as Gaussian Naïve Bayes can offer both simplicity and strong performance. Conversely, when the assumptions are violated, non-parametric approaches like

k-Nearest Neighbors can provide greater flexibility and accuracy, albeit at the expense of interpretability and computational efficiency.

5. Appendix

The code is here:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.datasets import make_classification
from sklearn.metrics import roc_curve, auc

# Split data into training and testing sets
def train_test_split(X, y, test_size=0.3, random_state=None):
    if random_state is not None:
        np.random.seed(random_state)
    indices = np.random.permutation(len(X))
    split_idx = int(len(X) * (1 - test_size))
    train_idx, test_idx = indices[:split_idx], indices[split_idx:]
    return X[train_idx], X[test_idx], y[train_idx], y[test_idx]

# Gaussian Naïve Bayes
class GaussianNaiveBayes:
    def fit(self, X, y):
        self.classes = np.unique(y)
        self.parameters = {}
        for c in self.classes:
            X_c = X[y == c]
            self.parameters[c] = {
                'mean': np.mean(X_c, axis=0),
                'var': np.var(X_c, axis=0),
                'prior': X_c.shape[0] / X.shape[0]
            }

    def predict(self, X):
        preds = []
        # Store the log probs
        log_probs = []
        for x in X:
```

```
            posteriors = []
            for c in self.classes:
                prior = np.log(self.parameters[c]['prior'])
                # Calculate likelihood using Gaussian distribution
                var = self.parameters[c]['var']
                mean = self.parameters[c]['mean']
                log_likelihood = -0.5 * np.sum(np.log(2 * np.pi *
var))
                log_likelihood -= 0.5 * np.sum(((x - mean) ** 2) /
var)
                posterior = prior + log_likelihood
                posteriors.append(posterior)
            log_probs.append(posteriors)
            preds.append(self.classes[np.argmax(posteriors)])
        return np.array(preds), np.array(log_probs)

# k-NN classifier
class KNN:
    def __init__(self, k=3):
        self.k = k

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y

    def predict(self, X):
        preds = []
        scores = []
        for x in X:
            distances = np.sqrt(np.sum((self.X_train - x) ** 2,
axis=1))
            k_indices = np.argsort(distances)[:self.k]
            k_nearest_labels = self.y_train[k_indices]
            # Determine the most common class label
            counts = np.bincount(k_nearest_labels)
            pred = np.argmax(counts)
            preds.append(pred)
```

```
            # Calculate the score as the proportion of positive
labels
            scores.append(np.mean(k_nearest_labels == 1))
        return np.array(preds), np.array(scores)

# evaluation module to calculate confusion matrix and plot it
def confusion_matrix(y_true, y_pred):
    classes = np.unique(np.concatenate([y_true, y_pred]))
    matrix = np.zeros((len(classes), len(classes)), dtype=int)
    for i, true in enumerate(classes):
        for j, pred in enumerate(classes):
            matrix[i, j] = np.sum((y_true == true) & (y_pred ==
pred))
    return matrix

def plot_confusion_matrix(cm, classes, title='Confusion Matrix'):
    plt.figure()
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes)
    plt.yticks(tick_marks, classes)
    thresh = cm.max() / 2.0
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            plt.text(j, i, format(cm[i, j], 'd'),
                    horizontalalignment="center",
                    color="white" if cm[i, j] > thresh else "black")
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()

# For binary classification, plot ROC curve and calculate AUC
def plot_roc_auc(y_true, scores, title='ROC Curve'):
    fpr, tpr, _ = roc_curve(y_true, scores)
    roc_auc = auc(fpr, tpr)
    plt.figure()
```

```
    plt.plot(fpr, tpr, color='darkorange', lw=2,
             label=f'ROC curve (area = {roc_auc:0.2f})')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(title)
    plt.legend(loc="lower right")
    plt.show()
    return roc_auc

# main program to choose datasets and run classifiers
def main():
    # Choose datasets
    # Binary dataset 1 : Breast Cancer
    bc = datasets.load_breast_cancer()
    X_bc, y_bc = bc.data, bc.target

    # Binary dataset 2 : Synthetic
    X_syn, y_syn = make_classification(n_samples=300,
n_features=20,
                                       n_informative=15, n_redundant=5,
                                       n_classes=2, random_state=42)

    # Multiclass dataset 1 : Iris
    iris = datasets.load_iris()
    X_iris, y_iris = iris.data, iris.target

    # Multiclass dataset 2 : Wine
    wine = datasets.load_wine()
    X_wine, y_wine = wine.data, wine.target

    datasets_list = [
        ('Breast Cancer', X_bc, y_bc, 'binary'),
        ('Synthetic Binary', X_syn, y_syn, 'binary'),
        ('Iris', X_iris, y_iris, 'multiclass'),
        ('Wine', X_wine, y_wine, 'multiclass')
```

```
    ]

    for name, X, y, dtype in datasets_list:
        print(f"\nDataset: {name}")
        X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
        print("Train shape:", X_train.shape, "Test shape:",
X_test.shape)

        # GNB classifier
        print("Training Gaussian Naïve Bayes...")
        gnb = GaussianNaiveBayes()
        gnb.fit(X_train, y_train)
        preds_gnb, log_probs_gnb = gnb.predict(X_test)
        cm_gnb = confusion_matrix(y_test, preds_gnb)
        print("Confusion Matrix (GNB):\n", cm_gnb)
        plot_confusion_matrix(cm_gnb, classes=np.unique(y),
title=f'{name} - GNB Confusion Matrix')

        if dtype == 'binary':
            pos_idx = list(gnb.classes).index(1)
            scores_gnb = np.exp(log_probs_gnb[:, pos_idx])
            auc_value = plot_roc_auc(y_test, scores_gnb,
title=f'{name} - GNB ROC Curve')
            print("AUC (GNB):", auc_value)

        # k-NN classifier
        print("Training k-NN classifier...")
        knn = KNN(k=3)
        knn.fit(X_train, y_train)
        preds_knn, scores_knn = knn.predict(X_test)
        cm_knn = confusion_matrix(y_test, preds_knn)
        print("Confusion Matrix (k-NN):\n", cm_knn)
        plot_confusion_matrix(cm_knn, classes=np.unique(y),
title=f'{name} - k-NN Confusion Matrix')

        if dtype == 'binary':
```

```
            auc_value_knn = plot_roc_auc(y_test, scores_knn,
title=f'{name} - k-NN ROC Curve')
            print("AUC (k-NN):", auc_value_knn)
        print("-" * 50)

if __name__ == '__main__':
    main()
```

To execute it, make it a python file, hw1.py for instance. Then execute:

```
python3 hw1.py
```