Naive Bayes Classifier - Analysis Report

1. Introduction

This report provides a detailed analysis of the implementation of a Naive Bayes Classifier on a binary classification problem.

We utilized two approaches: a manual implementation in Python and a comparative analysis using the scikit-learn library.

The goal is to predict class labels, and performance is measured using accuracy, precision, recall, and F1-score.

2. Data Preprocessing

The data preprocessing phase is emphasized, where the function `encode_class` converts class labels
into numeric values, a necessary step for further processing. The dataset is split into training
and test sets using the `splitting` function, with 80% of the data allocated for training.
The `groupUnderClass` function organizes the data into dictionaries based on the class labels,
which helps in computing class-specific statistics such as mean and standard deviation for each feature.
These statistics are crucial for applying the Naive Bayes algorithm, as they allow the model to
estimate the likelihood of a feature value belonging to a particular class.

3. Model Implementation (Manual)

The manual implementation of the Naive Bayes classifier calculates probabilities using Gaussian probability density functions for continuous data. It involves functions to compute the mean and standard deviation of features, calculate the likelihood of feature values, and predict the class with the highest probability. Additionally, it includes accuracy calculations by comparing predicted and actual class labels in the test set.

Code Snippets (Manual Implementation)

```
def encode_class(mydata):
    classes = []
    for i in range(len(mydata)):
        if mydata[i][-1] not in classes:
            classes.append(mydata[i][-1])
    for i in range(len(classes)):
        for j in range(len(mydata)):
            if mydata[j][-1] == classes[i]:
                mydata[j][-1] = i
    return mydata


def calculateGaussianProbability(x, mean, stdev):
    expo = math.exp(-(math.pow(x - mean, 2) / (2 * math.pow(stdev, 2))))
    return (1 / (math.sqrt(2 * math.pi) * stdev)) * expo
```

4. Performance Matrix

The code calculates essential performance metrics for evaluating binary classifiers, including precision,

recall, and F1-score, derived from confusion matrix components (true positives, false positives, false negatives, and true negatives). Precision indicates the accuracy of positive predictions, recall measures the identification of actual positives, and F1-score balances both, offering a comprehensive analysis of the model's performance beyond just accuracy.

5. Comparison with Scikit-learn Implementation

The final section compares a manually implemented Naive Bayes classifier with the `GaussianNB` model from the scikit-learn library. It highlights the streamlined process of building the classifier, including data preprocessing and performance evaluation. After training and testing the model, metrics like accuracy, precision, recall, and F1-score are computed using scikit-learn functions. Additionally, a confusion matrix is generated and visualized with seaborn, emphasizing the benefits of using established libraries for machine learning.

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

model = GaussianNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
```

6. Conclusion

In Conclusion this offers a comprehensive overview of Naive Bayes classifiers through both manual implementation and the use of scikit-learn. The manual method provides insights into the algorithm's mechanics, while scikit-learn showcases the convenience of machine learning libraries for quick model development. Detailed performance metrics-accuracy, precision, recall, and F1-score-provide a nuanced understanding of the classifier's capabilities.

The comparison highlights the trade-offs between grasping algorithmic details and utilizing high-level libraries for practical machine learning tasks.