Mathematics for Intelligent Systems-5

Project Phase-2

Title of the Project : Disease Diagnosis using Bayesian Filter

<u>Group Members:</u>

K Venkata Sai Sumitha – AM.EN.U4AIE21040

B Roopa Ravali – AM.EN.U4AIE21055

Introduction :

Disease diagnosis is a critical aspect of healthcare, aiming to identify and categorize various medical conditions based on observed symptoms. In the era of data-driven decision-making, machine learning techniques have found significant applications in the field of healthcare, offering efficient and accurate solutions for disease prediction.

One such machine learning algorithm widely used in disease diagnosis is the Naive Bayes Classifier. The Naive Bayes Classifier is a probabilistic model based on Bayes' theorem, which assumes that the presence of a particular feature in a class is independent of the presence of other features. Despite its simplistic assumption, the Naive Bayes Classifier has proven to be effective, especially in scenarios with a moderate amount of data.

In the context of disease diagnosis, the Naive Bayes Classifier can analyze patient symptoms and predict the likelihood of different medical conditions. The classifier is trained on historical data, where each instance consists of a set of symptoms and the corresponding diagnosed disease. Once trained, the model can generalize to new, unseen data and provide predictions on potential diseases based on observed symptoms.

Problem Formulation:

1.Notations

```
# Notations
n_samples: Total number of samples in the dataset
n_features: Total number of features in the dataset
X: Feature matrix (binary matrix representing presence or absence of
features)
y: Target variable (class labels)
c: Class label

class_probs: Probability of each class, P(c)

class_counts: Number of occurrences of each class in the training
set

feature_probs: Probability of each feature given the class,
P(feature|class)
```

```
class_feature_counts: Number of occurrences of each feature for a
given class

class_total_samples: Total number of samples for a given class
```

2.Fumulae

```
# Class Probability Calculation
     class_probs = class_counts / total_samples


# Feature Probability Calculation with Laplace Smoothing
feature_probs[i, :] = (class_feature_counts + 1) /
(class_total_samples + 2)

# Log Likelihood Calculation for Each Class
log_likelihoods = sum(X * log(feature_probs) + (1 - X) * log(1 -
feature_probs)) for each feature in X

# Log Probability Calculation for Each Class
log_probs = log_likelihoods + log(class_probs)

# Probability Calculation for Each Class
probs = exp(log_probs)
probs /= sum(probs)

# Prediction
predicted_class = argmax(probs)
```

3.Fumulae Usage

```
# Define a class for the custom Bernoulli Naive Bayes classifier
class BernoulliNB:
    # Initialize the classifier with a list of symptoms
    def __init__(self, symptoms)
        Store the list of symptoms for later use
        Initialize variables for class probabilities and feature
probabilities
```

```
    # Train the classifier with labeled training data
    def fit(self, training_data, training_labels)
        Calculate the number of samples and features in the training
data
        Identify unique classes and count occurrences
        Calculate class probabilities based on class occurrences
        Initialize feature probabilities with Laplace smoothing
        Update feature probabilities based on occurrences in each
class

    # Predict the class for new input data
    def predict(self, new_input)
        For each class, calculate the log likelihood of the input
data
        Choose the class with the highest log likelihood as the
prediction

# Create an instance of the Bernoulli Naive Bayes classifier with a
list of symptoms
custom_nb = BernoulliNB(symptoms)

# Train the classifier using labeled training data
custom_nb.fit(training_data, training_labels)

# Use the trained classifier to predict the class for new input data
predicted_class = custom_nb.predict(new_input)
```

Limitations of Existing Methods:

While Bernoulli Naive Bayes (BNB) can be effective for certain types of classification tasks, including disease diagnosis, it has its limitations. Here are some limitations of the existing methods, especially in the context of Bernoulli Naive Bayesian filtering for disease diagnosis:

Assumption of Independence:

BNB assumes that features are conditionally independent given the class label. This assumption might not hold in real-world scenarios where the presence or absence of symptoms may be correlated.

Limited Expressiveness:

BNB is a simple model, and its expressiveness may be limited in capturing complex relationships between symptoms and diseases. It assumes that features contribute independently to the probability of a particular class, which might not reflect the true nature of the data.

Sensitivity to Feature Independence Assumption:

BNB is sensitive to violations of the independence assumption. If certain symptoms are correlated or if their presence depends on the presence of others, the model's performance may degrade.

Lack of Continuous Feature Support:

BNB is designed for binary features (presence or absence). If the data includes continuous variables, discretization is required, which may lead to information loss and affect the model's performance.

Pseudo Code:

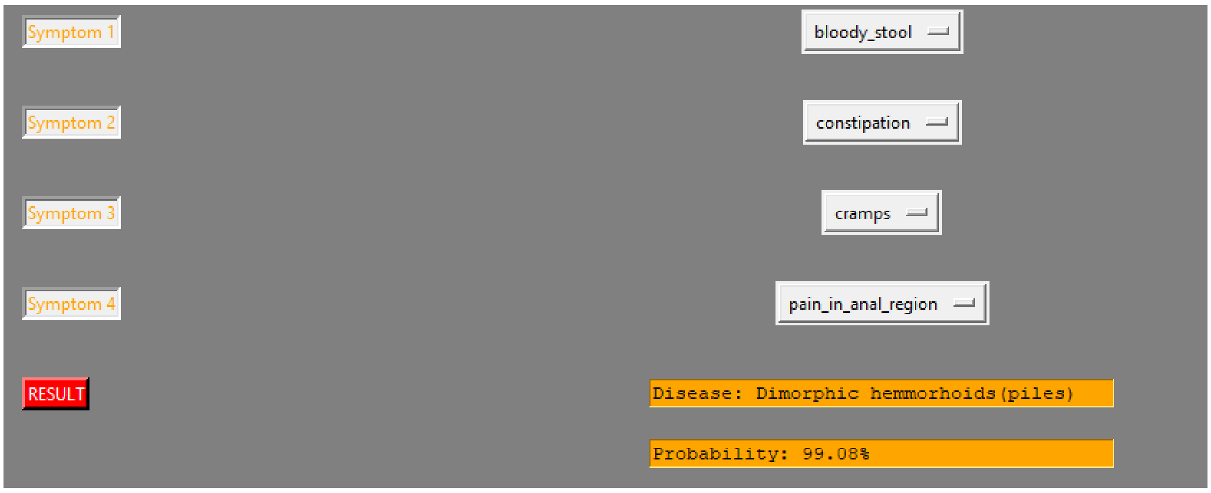
```
# Define a Naive Bayes classifier class
class BernoulliNB:
    # Constructor
    def __init__(self, symptoms):
        self.symptoms = symptoms
        self.class_probs = None
        self.feature_probs = None

    # Method to fit the model to the training data
    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.classes, class_counts = np.unique(y,
return_counts=True)
        n_classes = len(self.classes)

        self.class_probs = class_counts / n_samples
        self.feature_probs = np.zeros((n_classes, n_features))

        for i, c in enumerate(self.classes):
            class_mask = (y == c)
            class_samples = X[class_mask]

            # Estimate feature probabilities using Laplace smoothing
            self.feature_probs[i, :] = (np.sum(class_samples,
axis=0) + 1) / (np.sum(class_mask) + 2)

    # Method to predict class probabilities for input samples
    def predict_proba(self, X):
        class_probs_log = np.log(self.class_probs)
        feature_probs_log = np.log(self.feature_probs)

        # Convert the input list X to a NumPy array
        X_array = np.array(X)

        # Ensure X_array has the same number of features as
feature_probs_log
```

```python
        if X_array.shape[1] < feature_probs_log.shape[1]:
            X_array = np.pad(X_array, ((0, 0), (0,
feature_probs_log.shape[1] - X_array.shape[1])))

        # Use broadcasting to compute log likelihoods for each
sample and class
        log_likelihoods = X_array @ feature_probs_log.T + (1 -
X_array) @ (1 - feature_probs_log.T)

        # Add class log probabilities
        log_probs = log_likelihoods + class_probs_log

        # Convert log probabilities back to probabilities
        probs = np.exp(log_probs)
        probs /= np.sum(probs, axis=1, keepdims=True)

        return probs

    # Method to predict the class for input samples
    def predict(self, X):
        probs = self.predict_proba(X)
        return self.classes[np.argmax(probs, axis=1)]
```

Intermediate Results:

Symptoms : “bloody_stool”,”constipation”,”cramps”,and ”pain_in_anal_region”

The Naive Bayes algorithm predicts " Dimorphic hemmorhoids(piles)" with a probability of 99.08%



bloody_stool
constipation
cramps
pain_in_anal_region

Disease: Dimorphic hemmorhoids(piles)

Probability: 99.08%

The Naive Bayes algorithm predicts "Dimorphic hemmorhoids(piles)" with a high probability of 99.08% based on the symptoms "bloody_stool," "constipation," "cramps," and "pain_in_anal_region." The strong confidence level suggests a robust association between the provided symptoms and the predicted disease, showcasing the effectiveness of the model in identifying specific medical conditions from symptom inputs. This reinforces the utility of the Disease Diagnosis App in providing accurate and confident predictions for users.

The model considers the likelihood of these symptoms given each disease, producing a probability distribution. The highest probability indicates the predicted disease. This process provides users with a potential diagnosis along with a confidence level.