Here are responses to your queries:

**Q1. Probability that an employee is a smoker given that they use the health insurance plan**

This is a conditional probability question. We are given:

* P(H)=0.7P(H) = 0.7 (Probability an employee uses the health insurance plan)
* P(S∣H)=0.4P(S | H) = 0.4 (Probability an employee is a smoker given they use the health insurance plan)

We need to find P(S∣H)P(S | H). From the problem, this is already directly given:

P(S∣H)=0.4P(S | H) = 0.4

Thus, the probability that an employee is a smoker given that they use the health insurance plan is **0.4 or 40%**.

**Q2. Difference between Bernoulli Naive Bayes and Multinomial Naive Bayes**

| **Aspect** | **Bernoulli Naive Bayes** | **Multinomial Naive Bayes** |
| --- | --- | --- |
| **Data Assumption** | Features are binary (e.g., 0 or 1). | Features are frequency counts or non-negative integers. |
| **Application** | Text classification tasks where binary occurrence matters (e.g., whether a word is present in a document). | Text classification tasks where word frequency matters (e.g., bag of words representation). |
| **Probability Model** | Based on Bernoulli distribution. | Based on Multinomial distribution. |
| **Performance** | Suitable for sparse, binary datasets. | Suitable for datasets where term frequency is critical. |

**Q3. How does Bernoulli Naive Bayes handle missing values?**

Bernoulli Naive Bayes inherently assumes binary feature vectors and does not natively handle missing values. Missing values can disrupt the binary nature of the features. To address this:

* **Imputation**: Replace missing values with a binary default (e.g., 0 or the most frequent value).
* **Feature Engineering**: Add a feature to indicate whether a value was missing.
* **Preprocessing**: Ensure missing data is handled before fitting the model.

**Q4. Can Gaussian Naive Bayes be used for multi-class classification?**

Yes, Gaussian Naive Bayes can handle **multi-class classification**. The algorithm calculates the posterior probability for each class using the Gaussian distribution and assigns the class with the highest posterior probability. Scikit-learn's implementation supports multi-class classification out of the box.

**Q5. Assignment: Spambase Dataset with Naive Bayes**

**Steps to Implement the Assignment**

1. **Data Preparation**
   * Download the [Spambase Dataset](https://archive.ics.uci.edu/ml/datasets/Spambase).
   * Load and preprocess the data (e.g., handle missing values, normalize if necessary).
2. **Model Implementation**
   * Use Scikit-learn to implement **Bernoulli Naive Bayes**, **Multinomial Naive Bayes**, and **Gaussian Naive Bayes** classifiers.
   * Split the data using **10-fold cross-validation**.
3. **Performance Metrics**
   * Evaluate and report metrics for each model:
     + **Accuracy**
     + **Precision**
     + **Recall**
     + **F1 Score**
4. **Discussion**
   * Compare the performance metrics.
   * Analyze why one variant outperforms others (e.g., dataset characteristics).
5. **Conclusion**
   * Summarize findings.
   * Suggest improvements or future research directions (e.g., handling skewed distributions or incorporating feature selection).

**Template Python Code for Implementation**

import pandas as pd

from sklearn.model\_selection import cross\_val\_score, StratifiedKFold

from sklearn.naive\_bayes import BernoulliNB, MultinomialNB, GaussianNB

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

# Load dataset

data = pd.read\_csv("spambase.data", header=None)

X = data.iloc[:, :-1].values

y = data.iloc[:, -1].values

# Initialize models

models = {

"BernoulliNB": BernoulliNB(),

"MultinomialNB": MultinomialNB(),

"GaussianNB": GaussianNB()

}

# Cross-validation setup

skf = StratifiedKFold(n\_splits=10, shuffle=True, random\_state=42)

results = {}

for model\_name, model in models.items():

metrics = {"accuracy": [], "precision": [], "recall": [], "f1\_score": []}

for train\_idx, test\_idx in skf.split(X, y):

X\_train, X\_test = X[train\_idx], X[test\_idx]

y\_train, y\_test = y[train\_idx], y[test\_idx]

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

metrics["accuracy"].append(accuracy\_score(y\_test, y\_pred))

metrics["precision"].append(precision\_score(y\_test, y\_pred))

metrics["recall"].append(recall\_score(y\_test, y\_pred))

metrics["f1\_score"].append(f1\_score(y\_test, y\_pred))

results[model\_name] = {metric: sum(scores) / len(scores) for metric, scores in metrics.items()}

# Report results

for model\_name, metrics in results.items():

print(f"\n{model\_name} Performance:")

for metric, score in metrics.items():

print(f"{metric.capitalize()}: {score:.4f}")

Let me know if you'd like detailed assistance with any part!