Student information

Name: Hoang Tuan Tu

ID: 21000709

In [ ]:
```python
# Import library
import numpy as np
import pandas as pd
```

In [ ]:
```python
# Define function to calculate mean and standard deviation of the data
def mean_stddev(X):
    mean = np.mean(X, axis=0)
    stddev = np.std(X, axis=0)
    return mean, stddev

# Function to calculate Gaussian Probability
def gaussian_probability(x, mean, stddev):
    exponent = np.exp(-((x - mean) ** 2 / (2 * stddev ** 2)))
    return (1 / (np.sqrt(2 * np.pi) * stddev)) * exponent

# Naive Bayes Classification Function
def naive_bayes_classify(X_train_benign, X_train_malignant, X_test):
    # Calculate mean and standard deviation for each class
    mean_benign, stddev_benign = mean_stddev(X_train_benign)
    mean_malignant, stddev_malignant = mean_stddev(X_train_malignant)

    # Calculate prior probabilities for each class
    num_benign = len(X_train_benign)
    num_malignant = len(X_train_malignant)
    total_samples = num_benign + num_malignant
    prior_benign = num_benign / total_samples
    prior_malignant = num_malignant / total_samples

    # Predict for test data
    y_pred = []
    for sample in X_test:
        # Calculate prediction probabilities for benign and malignant samples
        prob_benign = np.prod(gaussian_probability(sample, mean_benign, stddev_benign)) * prior_benign
        prob_malignant = np.prod(gaussian_probability(sample, mean_malignant, stddev_malignant)) * prior_malignant

        # Compare probabilities and classify sample based on the highest probability
        if prob_benign > prob_malignant:
            y_pred.append(2)  # Benign
        else:
            y_pred.append(4)  # Malignant

    return np.array(y_pred)
```

In [ ]:
```python
# Read data
data_path = "data.csv"
with open(data_path, 'r') as f:
    data = f.readlines()
```

In [ ]:
```python
# Preprocessing
data = [list(map(int, x.strip().split(",")))[1:len(x)] for x in data]
data = np.array(data)
```

In [ ]:
```python
# Visualize data
label_df = ['Type', 'Thickness', 'Csize', 'Cshape','Adhesion',
 'Epithelial csize' ,'Nuclei', 'Chromatin' ,'Nucleoli','Mitoses']

temp = dict()

for i in range(len(label_df)):
    temp.setdefault(label_df[i], data[:,i])
df = pd.DataFrame(temp)

print(df.head(5))
```

```
   Type  Thickness  Csize  Cshape  Adhesion  Epithelial csize  Nuclei  \
0     2          5      1       1         1                 2       1
1     2          5      4       4         5                 7      10
2     2          3      1       1         1                 2       2
3     2          6      8       8         1                 3       4
4     2          4      1       1         3                 2       1

   Chromatin  Nucleoli  Mitoses
0          3         1        1
1          3         2        1
2          3         1        1
3          3         7        1
4          3         1        1
```

In [ ]:
```python
# Split data into classes
X_benign = data[data[:, 0] == 2][:, 1:]
X_malignant = data[data[:, 0] == 4][:, 1:]
```

In [ ]:
```python
# Split data into training and testing data
X_train_benign, X_test_benign = X_benign[:80], X_benign[80:]
X_train_malignant, X_test_malignant = X_malignant[:40], X_malignant[40:]
```

In [ ]:
```python
# Combine training and testing data again
X_train = np.concatenate((X_train_benign, X_train_malignant))
X_test = np.concatenate((X_test_benign, X_test_malignant))
```

In [ ]:
```python
# Predict labels for testing data
y_test_benign = np.full(len(X_test_benign), 2)
y_test_malignant = np.full(len(X_test_malignant), 4)
y_test = np.concatenate((y_test_benign, y_test_malignant))
```

In [ ]:
```python
# Predict results with X_test
y_pred = naive_bayes_classify(X_train_benign, X_train_malignant, X_test)
print(y_pred)
```

```
[2 2 2 2 4 2 2 2 2 2 2 2 2 4 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 4 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 4 2 2 2 2 2 2 2 2 2 2 2 4 2 2 2 4 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 4 2 2 2 2 2 2 2 2 2 2 2 4 2 4 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 4 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 4 2 2 2 2 2 2 2 2 2 2 2 2 2 2 4 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 4 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 4 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 4 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 4 2 2 2 2 2 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 2 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 2 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 2 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
```

In [ ]:
```python
# Calculate evaluation metrics
correct_predictions = np.sum(y_test == y_pred)
accuracy = correct_predictions / len(y_test)

true_positives = np.sum((y_test == 4) & (y_pred == 4))
false_positives = np.sum((y_test == 2) & (y_pred == 4))
false_negatives = np.sum((y_test == 4) & (y_pred == 2))

precision = true_positives / (true_positives + false_positives)
recall = true_positives / (true_positives + false_negatives)

# Print results
print(f"Accuracy: {accuracy * 100:.02f}")
```

```
print(f"Precision: {precision * 100:.02f}")
print(f"Recall: {recall * 100 :.02f}")
```

```
Accuracy: 96.72
Precision: 92.52
Recall: 98.51
```