Implementing Naive Bayes from scratch with Python in OOP

In [22]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import datasets
import matplotlib.pyplot as plt
```

In [23]:
```
class NaiveBayes:
    # Don't need init function because don't have any parameters to configur
    def fit (self, X, y):
        n_samples, n_features = X.shape
        # Get the unique classes and count how many
        self._classes = np.unique(y)
        n_classes = len(self._classes)

        # Calculate the mean, variance, and prior probability for each class
        self._mean = np.zeros((n_classes, n_features), dtype=np.float64)
        self._var = np.zeros((n_classes, n_features), dtype=np.float64)
        self._priors = np.zeros(n_classes, dtype=np.float64)

        for idx, c in enumerate(self._classes):
            # Only get samples of class
            X_c = X[y == c]
            # Calculate for each class
            self._mean[idx,:] = X_c.mean(axis=0)
            self._var[idx,:] = X_c.var(axis=0)
            self._priors[idx] = X_c.shape[0] / float(n_samples)

    def predict(self, X):
        y_pred = [self._predict(x) for x in X]
        return np.array(y_pred)

    def _predict(self, x):
        posteriors = []

        # Calculate the posterior probability for each class
        for idx, c in enumerate(self._classes):
            prior = np.log(self._priors[idx])
            posterior = np.sum(np.log(self._gauss(idx,x))) # Gaussian is PDF
            posterior += prior
            posteriors.append(posterior)

        return self._classes[np.argmax(posteriors)] # find the maximum poste

    def _gauss(self, class_idx, x):
        mean = self._mean[class_idx]
        var = self._var[class_idx]
        numerator = np.exp(-((x - mean) ** 2) / (2 * var))
        denominator = np.sqrt(2 * np.pi * var)
        return numerator / denominator
```

In [24]:
```
if __name__ == "__main__":
    def accuracy(y_true,y_pred):
        accuracy = np.sum(y_true == y_pred) / len(y_true)
        return accuracy

    # Create dataset
    X, y = datasets.make_classification(
        n_samples=1000, n_features=10, n_classes=2, random_state=42
    )

    # Split data
    X_train, X_test, y_train, y_test = train_test_split(
        X,y, test_size=0.2, random_state=42
        )

    nb = NaiveBayes()
    nb.fit(X_train, y_train)
    predictions = nb.predict(X_test)

    print(f"Naive Bayes classification accuracy {accuracy(y_test, predictior
```

```
Naive Bayes classification accuracy 0.81
```