```
Name: Aarya Admane 22630

import numpy as np

class ART1:
    def __init__(self, num_features, num_clusters, vigilance=0.5):
        self.num_features = num_features
        self.num_clusters = num_clusters
        self.vigilance = vigilance
        self.weights = np.random.rand(num_clusters, num_features)

    def complement_coding(self, input_pattern):
        return np.concatenate((input_pattern, 1 - input_pattern))

    def match_function(self, input_pattern, cluster_weights):
        return np.sum(np.minimum(input_pattern, cluster_weights)) /
np.sum(input_pattern)

    def train(self, input_patterns):  # Ensure this method exists
        for input_pattern in input_patterns:
            input_pattern = self.complement_coding(input_pattern)
            for i in range(self.num_clusters):
                match = self.match_function(input_pattern,
self.weights[i])
                if match >= self.vigilance:
                    self.weights[i] = np.minimum(self.weights[i],
input_pattern)
                    break

    def predict(self, input_pattern):
        input_pattern = self.complement_coding(input_pattern)
        best_match = -1
        best_index = -1
        for i in range(self.num_clusters):
            match = self.match_function(input_pattern,
self.weights[i])
            if match > best_match and match >= self.vigilance:
                best_match = match
                best_index = i
        return best_index


# Running the model
if __name__ == "__main__":
    input_patterns = np.array([
        [1, 0, 1, 0],
        [1, 1, 0, 0],
        [0, 1, 1, 0],
        [1, 0, 1, 1]
    ])
    num_features = input_patterns.shape[1]
    num_clusters = 2
    vigilance = 0.6

    art1 = ART1(num_features * 2, num_clusters, vigilance)
    art1.train(input_patterns)  # Check if train() exists
    test_pattern = np.array([1, 0, 1, 0])
    cluster = art1.predict(test_pattern)
    print(f"The test pattern {test_pattern} belongs to cluster {cluster}")
```

```
The test pattern [1 0 1 0] belongs to cluster 0
```