Experiment No. 4

Problem Statement - To apply the artificial immune pattern recognition to perform a task of structure damage Classification.

Objective: To apply the artificial immune pattern recognition to perform a task of structure damage Classification.

Theory:
AIS emerged in the mid-1980s with articles authored by Farmer, Packard and Perelson (1986) and Bersini and Varela (1990) on immune networks. Artificial Immune Systems (AIS). AIS take inspiration from the immune system in order to build novel computational tools to solve problems in a vast range of domain areas. The basic immune theories used to explain how the immune system perform pattern recognition are described and their corresponding computational models are presented. This is followed with a survey from the literature of AIS applied to pattern recognition. The chapter is concluded with a trade-off between AIS and artificial neural networks as pattern recognition paradigms. Artificial immune systems can be defined as abstract or metaphorical computational systems developed using ideas, theories, and components, extracted from the immune system. Most AIS aim at solving complex computational or engineering problems, such as pattern recognition, elimination, and optimization. In artificial intelligence, artificial immune systems (AIS) are a class of computationally intelligent, rule-based machine learning systems inspired by the principles and processes of the vertebrate immune system. The algorithms are typically modeled after the immune system's characteristics of learning and memory for use in problemsolving. The field of artificial immune systems (AIS) is concerned with abstracting the structure and function of the immune system to computational systems, and investigating the application of these systems towards solving computational problems from mathematics, engineering, and information technology. Is a sub-field of biologically inspired computing, and natural computation, with interests in machine learning and belonging to the broader field of artificial intelligence.

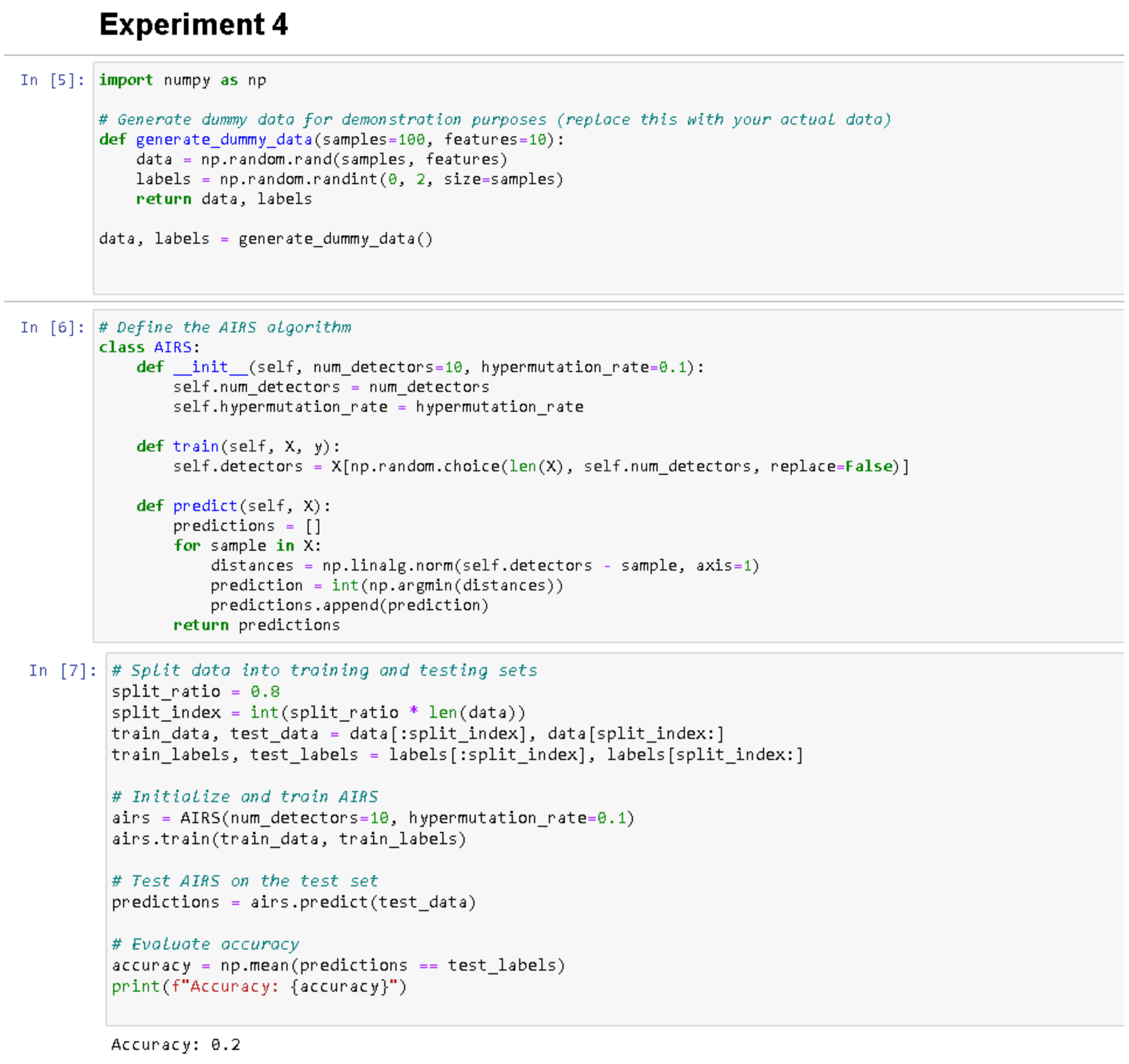
Techniques
The common techniques are inspired by specific immunological theories that explain the function and behavior of the mammalian adaptive immune system.

- Clonal selection algorithm: A class of algorithms inspired by the clonal selection theory of acquired immunity that explains how and T lymphocytes improve their response to antigens over time called affinity maturation. These algorithms focus on the Darwinian attributes of the theory where selection is inspired by the affinity of antigen–antibody interactions, reproduction is inspired by cell division, and variation is inspired by somatic hypermutation. Clonal selection algorithms are most commonly applied to optimization and pattern recognition domains, some of which resemble parallel hill climbing and the genetic algorithm without the recombination operator.

- Negative selection algorithm: Inspired by the positive and negative selection processes that occur during the maturation of T cells in the thymus called T cell tolerance. Negative selection refers to the identification and deletion (apoptosis) of self-reacting cells, that is T cells that may select for and attack self tissues.
- This class of algorithms are typically used for classification and pattern recognition problem domains where the problem space is modeled in the complement of available knowledge. For example, in the case of an anomaly detection domain the algorithm prepares a set of exemplar pattern detectors trained on normal (nonanomalous) patterns that model and detect unseen or anomalous patterns.
- Immune network algorithms: Algorithms inspired by the idiotypic network theory proposed by Niels Kaj Jerne that describes the regulation of the immune system by anti-idiotypic antibodies (antibodies that select for other antibodies). This class of algorithms focus on the network graph structures involved where antibodies (or antibody producing cells) represent the nodes and the training algorithm involves growing or pruning edges between the nodes based on affinity (similarity in the problems representation space). Immune network algorithms have been used in clustering, data visualization, control, and optimization domains, and share properties with artificial neural networks
- Dendritic cell algorithms: The dendritic cell algorithm (DCA) is an example of an immune inspired algorithm developed using a multi-scale approach. This algorithm is based on an abstract model of dendritic cells (DCs). The DCA is abstracted and implemented through a process of examining and modeling various aspects of DC function, from the molecular networks present within the cell to the behaviour exhibited by a population of cells as a whole. Within the DCA information is granulated at different layers, achieved through multi-scale processing.

Conclusion: Thus, we have successfully implemented a program for Artificial immune pattern recognition.

Code and Output:



Experiment 4

In [5]:
```python
import numpy as np

# Generate dummy data for demonstration purposes (replace this with your actual data)
def generate_dummy_data(samples=100, features=10):
    data = np.random.rand(samples, features)
    labels = np.random.randint(0, 2, size=samples)
    return data, labels

data, labels = generate_dummy_data()
```

In [6]:
```python
# Define the AIRS algorithm
class AIRS:
    def __init__(self, num_detectors=10, hypermutation_rate=0.1):
        self.num_detectors = num_detectors
        self.hypermutation_rate = hypermutation_rate

    def train(self, X, y):
        self.detectors = X[np.random.choice(len(X), self.num_detectors, replace=False)]

    def predict(self, X):
        predictions = []
        for sample in X:
            distances = np.linalg.norm(self.detectors - sample, axis=1)
            prediction = int(np.argmin(distances))
            predictions.append(prediction)
        return predictions
```

In [7]:
```python
# Split data into training and testing sets
split_ratio = 0.8
split_index = int(split_ratio * len(data))
train_data, test_data = data[:split_index], data[split_index:]
train_labels, test_labels = labels[:split_index], labels[split_index:]

# Initialize and train AIRS
airs = AIRS(num_detectors=10, hypermutation_rate=0.1)
airs.train(train_data, train_labels)

# Test AIRS on the test set
predictions = airs.predict(test_data)

# Evaluate accuracy
accuracy = np.mean(predictions == test_labels)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.2
```