**Report: Feature Selection Using Genetic Algorithm**

Student name: Jumana Al Rayes

Student ID: 202320016

**Introduction**

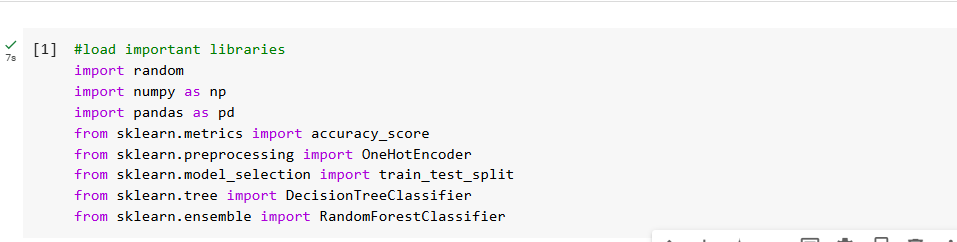
Feature selection is a crucial step in machine learning that helps improve model efficiency and performance. Reducing the number of irrelevant or redundant features, lowers the dimensionality of the dataset, speeding up computation and reducing the risk of overfitting. This enables the model to focus on the most important features, leading to better generalization and more accurate predictions. Overall, feature selection simplifies the model, enhances interpretability, and improves its ability to make predictions on unseen data.

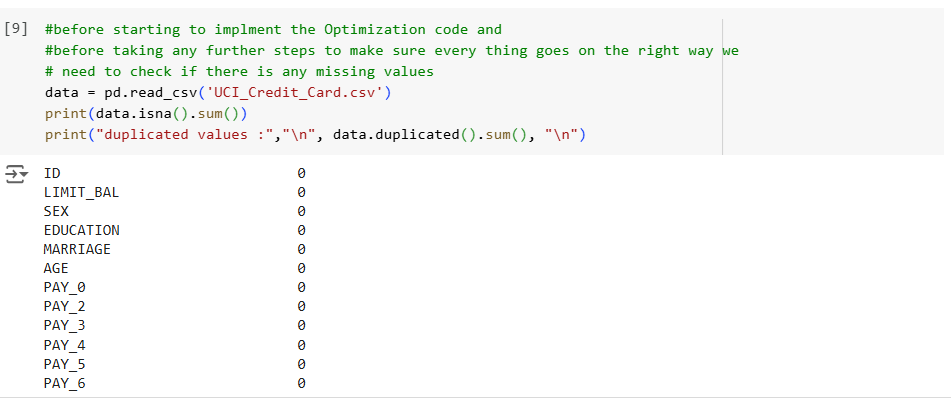
**Objective**: The goal of the project is to apply **genetic algorithms** for feature selection in a classification task using the **Default of Credit Card Clients** dataset.

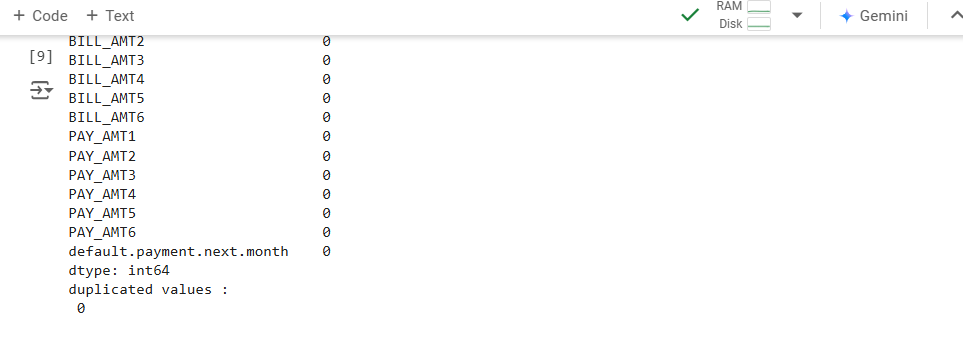
**About Dataset:**

This dataset contains information on default payments, demographic factors, credit data, history of payment, and bill statements of credit card clients in Taiwan from April to September 2005. The dataset is taken from: <https://www.kaggle.com/datasets/uciml/default-of-credit-card-clients-dataset?resource=download>

**Preprocessing steps:**



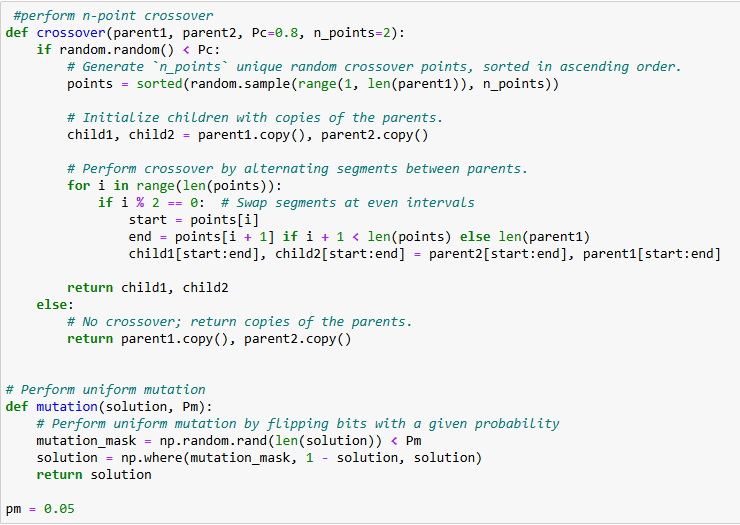




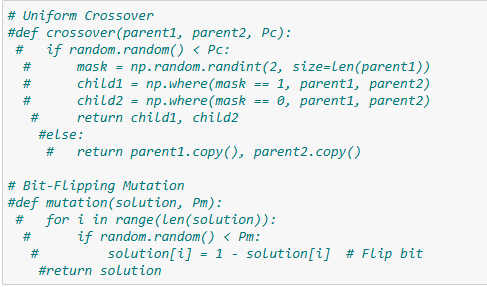
Explain: to see if there are any missing values or noisy data that can affect the process of applying optimization Evolutionary Algorithm to the dataset.



The function initializes a binary population for a genetic algorithm, creating a population Size × Num Features array where each element is randomly 0 or 1.

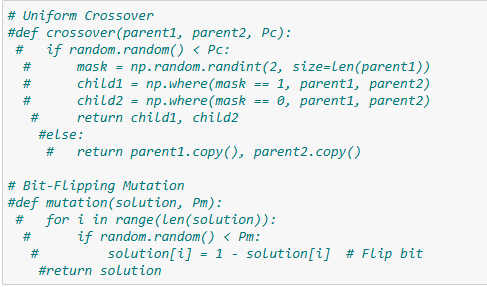


Performs **n-point crossover** between two parents. With probability Pc (default: 0.8), select **n-points crossover points**. Alternates segments between parents to create two offspring. If no crossover occurs, the parents are returned unchanged. **Pc**: Crossover probability controls how often crossover happens.



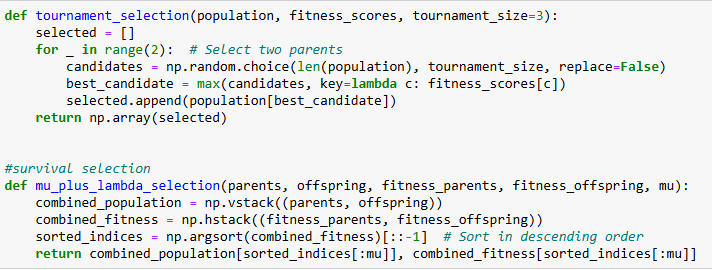
**Uniform Crossover**:

* Swaps genes between two parents based on a randomly generated mask.
* **Process**:
  + A binary mask of the same length as the parents is generated (random 0s and 1s).
  + Where the mask is 1, genes from parent1 are taken; otherwise, genes from parent2 are used.
* If the crossover probability (Pc) is not met, the parents are returned unchanged.



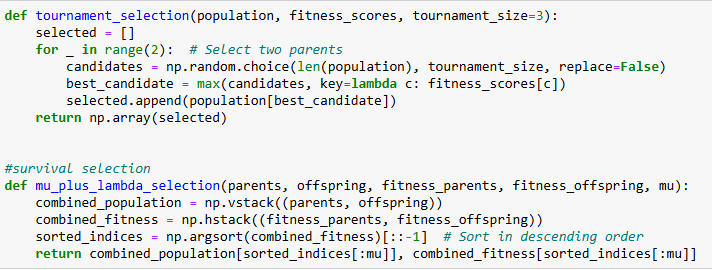
**Bit-Flipping Mutation**:

* Each bit in a solution is flipped (0 to 1, or 1 to 0) with a mutation probability Pm.
* Ensures genetic diversity in the population by introducing new solutions.



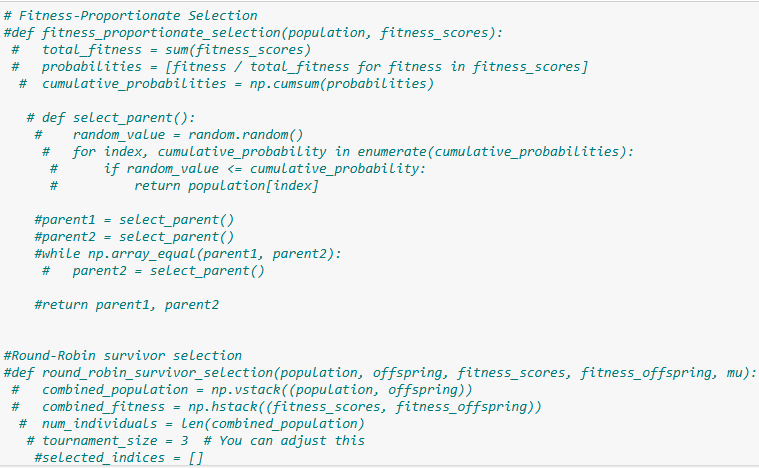
**Tournament Selection**:

* **Purpose**: Select two parents for crossover based on their fitness.
* **Process**:
  + Randomly selects tournament-size individuals from the population.
  + Chooses the individual with the highest fitness from the selected candidates.
  + Repeats the process of selecting two parents.
* Ensures the fittest individuals have a higher chance of being selected while maintaining diversity.



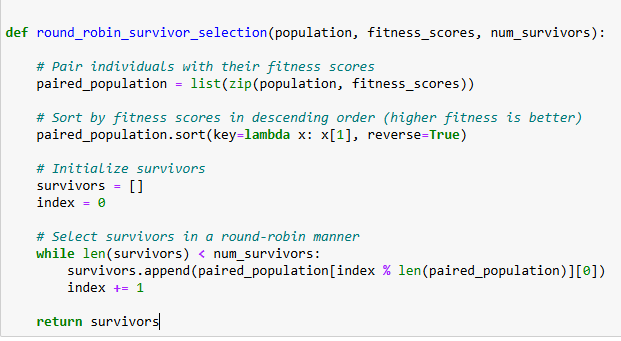
**(μ + λ) - Selection**:

* **Purpose**: Select the top **μ** individuals from the combined parent and offspring populations based on fitness.
* **Process**:
  + Combines parents and offspring into a single population.
  + Sorts individuals by their fitness in descending order.
  + Select the top **μ** individuals to form the next generation.
* Promotes survival of the fittest while allowing offspring to compete with parents.



**Fitness-Proportionate Selection**:

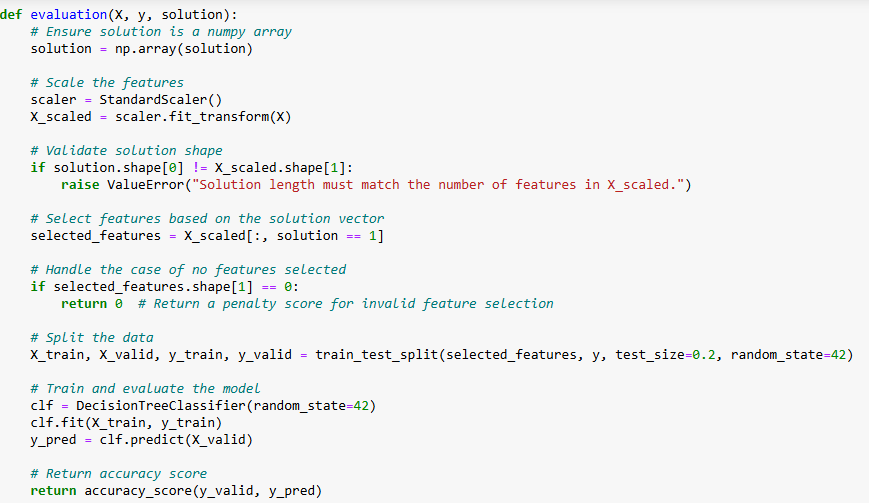
* **Purpose**: Select parents for crossover based on their fitness, giving fitter individuals a higher chance of selection.
* **Process**:
  + Calculates the selection probability for each individual by normalizing fitness scores.
  + Uses a cumulative probability distribution to select parents based on these probabilities randomly.
* **Ensures diversity** while favoring higher fitness solutions. Re-selects if both parents are identical.



**Round-Robin Survivor Selection**:

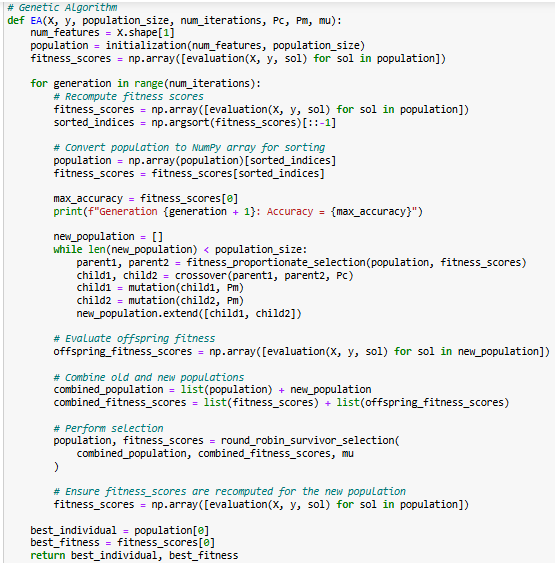
**Purpose:** Chooses survivors for the next generation.

* **Input:**
* population: A list of individuals in the population.
* fitness\_scores: A list of fitness scores corresponding to the individuals.
* num\_survivors: The desired number of survivors.
* **Process:**
* Pair individuals with their fitness scores.
* Sort the population by fitness in descending order.
* Use a circular index to rotate through the sorted population.
* **Output:**
* Returns a list of survivors selected in a round-robin manner
* This ensures fairness and maintains diversity by not just focusing on the fittest individuals.

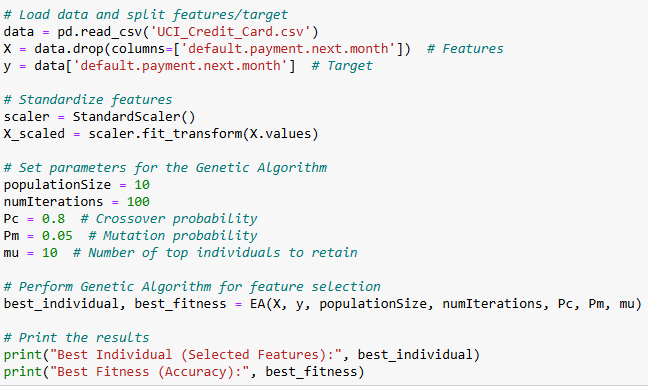


**Steps:**

1. **Input Validation**:
   * Ensures solution is a NumPy array.
   * Verifies the length of solution matches the number of features in X.
2. **Feature Scaling**:
   * Standardizes features in X using StandardScaler to improve model performance.
3. **Feature Selection**:
   * Uses the binary solution vector to select features (columns in X\_scaled where solution == 1).
4. **Handle Empty Selection**:
   * If no features are selected (solution has all zeros), it returns a fitness score of 0 as a penalty.
5. **Data Splitting**:
   * Splits the selected features and target y into training and validation sets (80%-20%).
6. **Model Training and Evaluation**:
   * Fits a DecisionTreeClassifier on the training set.
   * Evaluates the classifier on the validation set using accuracy\_score.
7. **Output**:
   * Returns the classification accuracy as the fitness score, which measures the quality of the selected features.



This Genetic Algorithm (GA) optimizes feature selection for classification tasks. It initializes a binary population, evaluates fitness (accuracy of a Decision Tree classifier), and iteratively evolves the population over generations using tournament selection, crossover, and mutation. A (μ + λ) survival strategy ensures the best solutions are retained. The algorithm balances exploration and exploitation to identify the feature subset that maximizes classification accuracy, returning the best subset and its fitness score.

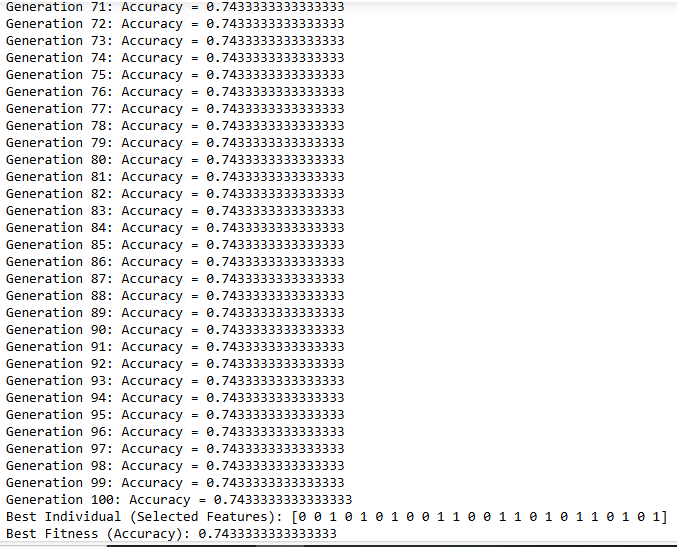


* Loading the data **Default of Credit Card Clients** dataset, and then split it into X, Y features and target, Standardizes the features using Standard Scaler Scaling for the data to ensure that features with different units or scales don't disproportionately affect machine learning models, improving their performance and convergence, for improved model performance.
* **GA Parameters**: Defines parameters such as **population size**, **number of iterations**, crossover probability **(Pc)**, mutation probability **(Pm)**, and the number of individuals **(μ)** to retain per generation.
* **Execution**: Calls the EA function to perform the Genetic Algorithm. The GA evolves a population to optimize feature selection, maximizing classification accuracy.
* **Output**: Prints the best individual (optimal feature subset) and its fitness score (classification accuracy) achieved after all generations.
* This approach identifies the most relevant features to improve the classification of credit card defaults, balancing accuracy with feature reduction.

**Experimentation**

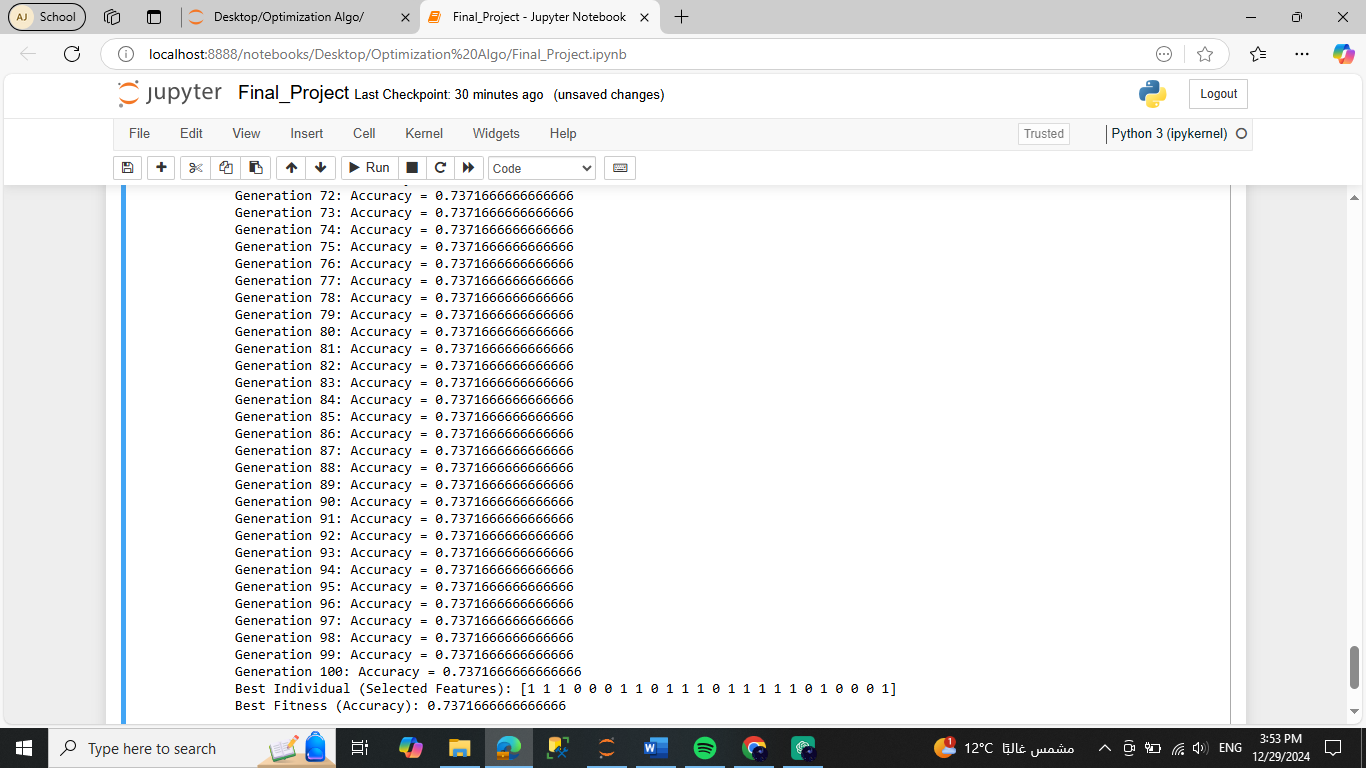
* **Parameters**
* **Population Size**: 10
* **Number of Iterations**: 100
* **Crossover Probability (Pc)**: 0.8
* **Mutation Probability (Pm)**: 0.05
* **μ (Top Individuals Retained)**: 10

**Results**



**Genetic operators(Uniform crossover, Bit-flipping mutation, fitness\_proportionate selection & Round Robin):**

* **Best Fitness:** 0.7433333333achieved consistently across multiple generations
* From Generation 3 to 12, there is an observable improvement in accuracy (up to 0.736). This indicates that genetic operators such as crossover and mutation are beginning to explore better feature combinations.
* Minor improvements, such as from 0.735 to 0.736 (Generation 12) and 0.740 to 0.743 (Generation 53 to 56), indicate the impact of genetic diversity introduced by crossover and mutation.
* The best solution achieves an accuracy of 0.743 (Fitness = 0.743) with a selected feature set:  
  [0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1].
* **Analysis of Genetic Operators:**
* **Uniform Crossover:** This crossover method randomly selects genes from either parent with equal probability. It ensures diverse combinations of features in the offspring, helping the search space exploration. The steady improvement over generations reflects effective recombination of good traits.
* **Bit-flipping Mutation:** This mutation operator introduces small, random changes by flipping bits in the feature vector (e.g., turning a 0 to a 1 or vice versa). It helps maintain genetic diversity, prevents premature convergence, and explores new regions in the search space.
* **Fitness-Proportionate Selection:** ensures that higher-fitness individuals have a greater chance of contributing to the next generation. The steady improvement in early generations demonstrates the success of this operator in propagating better solutions.
* **Round-Robin:** This operator ensures that diverse feature subsets compete, balancing exploration and exploitation. Its role is evident in maintaining population diversity during periods of stagnation.



**Genetic opreators(n-point Crossover , Bit-flipping Mutation, Tournment selection & (μ+λ)-Selection):**

* **Best Fitness:** **0.7371**, achieved consistently across multiple generations.
* **Accuracy** Improved during early generations, then plateaued at **0.7372**, starting from Generation 51, indicating the algorithm reached a convergence point.
* **Best Individual (Selected Features):**  
  [1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1]  
  This suggests that only certain features significantly contributed to the performance, while others were redundant or detrimental.
* Although fitness improved initially, the lack of progress after Generation 51 suggests:

1. Limited diversity in the population.
2. A local optimum may have been reached, preventing further search space exploration.

* **Analysis of Genetic Operators:**
* **n-point Crossover**: This crossover introduces variability by exchanging sections of the parent chromosomes at multiple points. Likely contributed to maintaining diversity in the population, preventing premature convergence during early generations.
* **Bit-flipping Mutation**: This mutation operator flips the binary value of randomly selected genes. Helps introduce new traits and recover diversity during stagnation but may have had limited impact here since the accuracy gain post-stagnation was minor.
* **Tournament Selection**: By selecting individuals based on relative fitness, tournament selection balances exploration and exploitation. The use of tournament selection likely ensured the survival of strong individuals, contributing to consistent improvements.
* **(μ+λ)-Selection:** Combines both parent and offspring populations to select the next generation. Likely ensured retention of high-performing solutions, reinforcing incremental improvements over generations.

**Performance Comparison :**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Genetic Operators: Uniform Crossover, Bit-flipping Mutation, Fitness-proportionate Selection & Round Robin** | **Genetic Operators: N-point Crossover, flipping Mutation, Tournament Selection & (μ+λ)-Selection** |
| **Best Accuracy** | 0.743 | 0.7371 |
| **Convergence Speed** | Faster: Accuracy increased steadily in earlier generations, plateauing around 0.7433 in later generations | Moderate: Accuracy improves gradually with some steady increases |
| **Stability** | Stable after Generation 50, with minor fluctuations around the 0.7433 value | |  | | --- | |  |  |  | | --- | | Stable after Generation 50, with little fluctuation | |
| **Diversity** | High: Maintained across generations with gradual improvements | High: Gradual improvement, suggesting exploration of feature space |
| **Exploration vs. Exploitation** | Balanced: Exploration allows gradual improvement, avoiding early exploitation | Balanced: Tends towards exploitation after early generations, leading to steady accuracy |
| **Premature Convergence** | Low: Avoids early convergence, with continued exploration towards 0.7433 | Low: Gradual convergence without early stagnation |