Aplicando un Algoritmo Genético a la tarea de Machine Learning

Adán González Rodríguez
Sara Porto Álvarez

Dataset Utilizado

Hemos decidido usar el Dataset: Mall Customer Segmentation

El dataset está diseñado para enseñar conceptos relacionados con la segmentación de clientes, también conocida como análisis de canasta de mercado (market basket analysis)

Lo usaremos para hacer un clustering sobre los atributos del

Código: Código en GitHub

- **Imports** de las librerías necesarias
- Clase **GeneticAlgorithm**
  - Método **__init__**
  - Método **run(task_ml)**
  - Método **tournament_selection(population, fitnesses, k)**
  - Método **plot_clusters(best_solution, ml_task, dim_x=0, dim_y=1)**
- Clase **MachineLearningTask**
  - Método **__init__**
  - Método **create_individual()**
  - Método **calculate_sse(individual)**
  - Método **fitness_function(individual)**
  - Método **crossover(parent1, parent2, crossover_rate)**
  - Método **mutation(individual, generation, max_gens)**
- Función auxiliar **plot_fitness_evolution(best_fitness_per_generation)**
- Bloque **main**

Cómo representamos al individuo

- Como una tupla de k*dim valores flotantes
- Cada valor en la tupla es una coordenada de un centroide en el espacio de características
- Creación de un individuo:

```python
def create_individual(self):
    """
    Retorna una solución (individuo) aleatoria.
    En este caso:
      - Para clustering: lista de k*dim floats aleatorios.
    """
    return tuple(
        random.uniform(self.lower_bound, self.upper_bound)
        for _ in range(self.params_per_ind)
    )
```

Fitness

- Evalúa qué tan buena es la solución en base a la calidad del individuo
- Al tratarse de clustering, nosotros usamos el **SSE negativo**
  - El SSE es la suma de errores cuadrados
  - Será mejor cuanto más alto sea su valor

```python
def calculate_sse(self, individual):
    '''
    Función para calcular el error cuadrático medio de un individuo.
    '''
    centers = np.array(individual).reshape(self.k, self.dim)
    dists = np.linalg.norm(self.data[:, None, :] - centers[None, :, :], axis=2)
    min_dists = np.min(dists, axis=1)
    return np.sum(min_dists**2)
```

```python
def fitness_function(self, individual):
    """
    Evalúa la calidad del individuo y retorna
    un valor numérico (cuanto más alto, mejor).
    En este caso:
      - Clustering: -SSE (SSE negativo)
    """
    return -self.calculate_sse(individual)
```

Hiperparámetros del GA (Genetic algorithm)

Pop_size: Tamaño de Población (Cantidad de individuos en cada generación)

Generations: Número de Generaciones (Numero máximo de Iteraciones de nuestro programa)

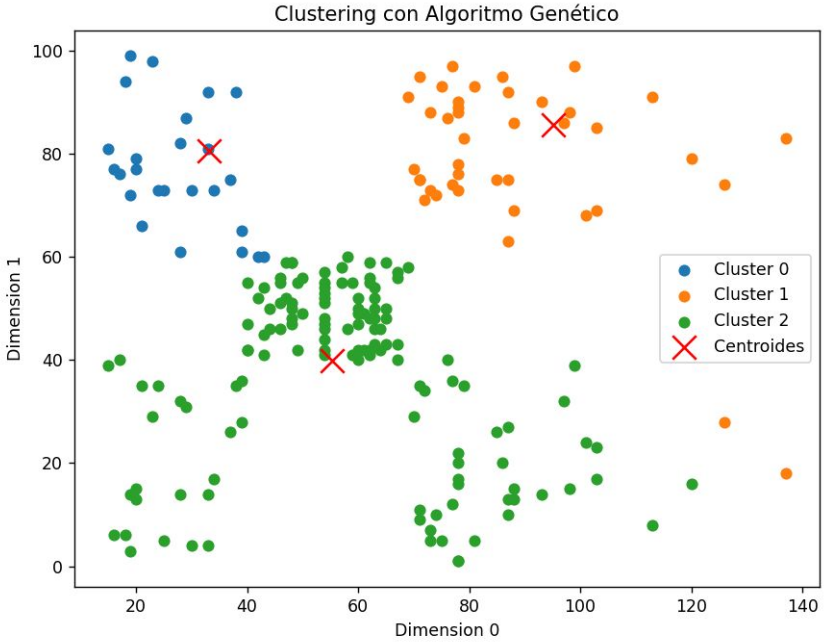
Crossover Rate: Tasa de Cruzamiento (Probabilidad de que dos individuos crucen sus genes)

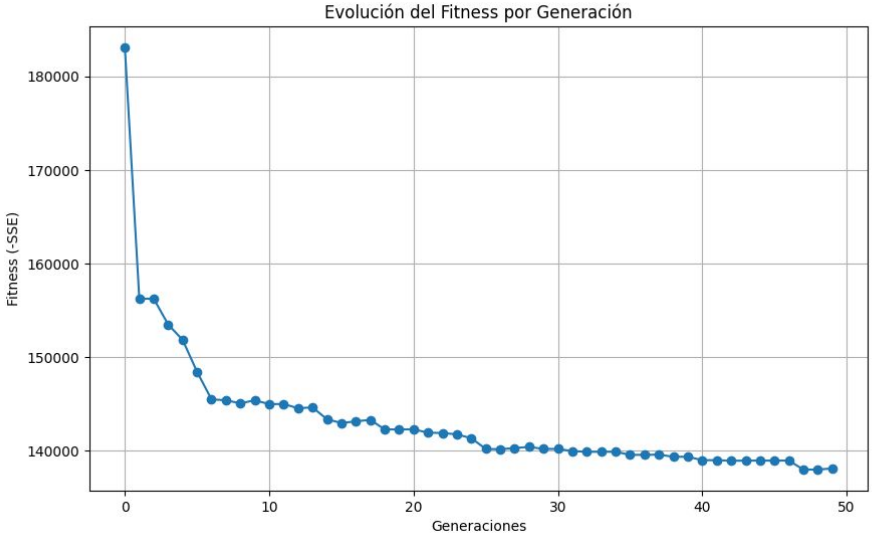
Mutation_Rate: Tasa de Mutación (Probabilidad de que cada gen de un individuo sufra una mutación/cambio aleatorio)

Patience: Paciencia (Numero de generaciones consecutivas permitidas sin mejora significativamente antes de detener el algoritmo)

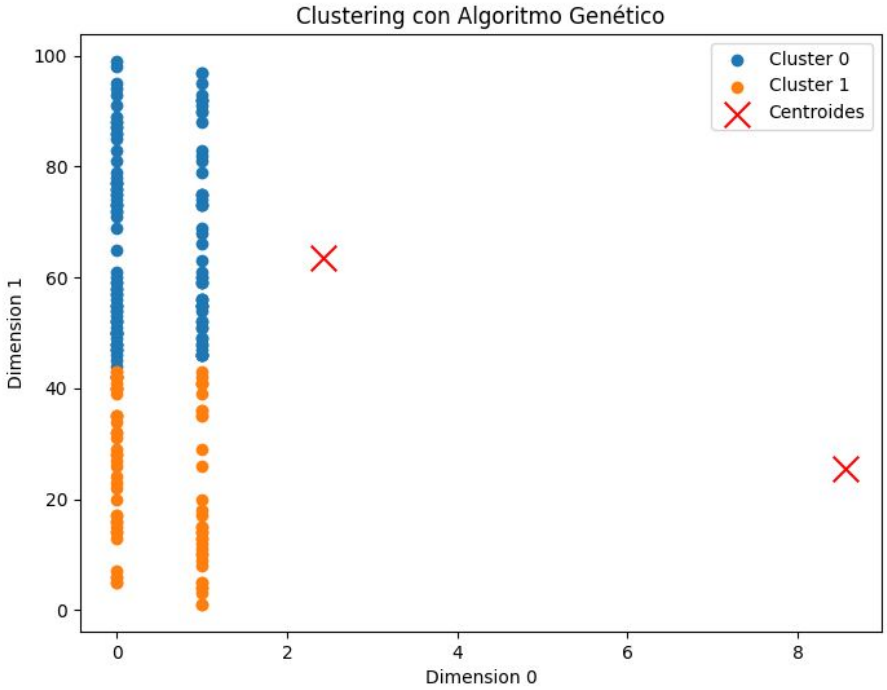
Min_Delta: (Minimo de Mejora esperada) Define el umbral mínimo de mejora en el fitness para considerar que ha habido progreso

3 Clusters, Gráfica Annual Income + Spending Scores

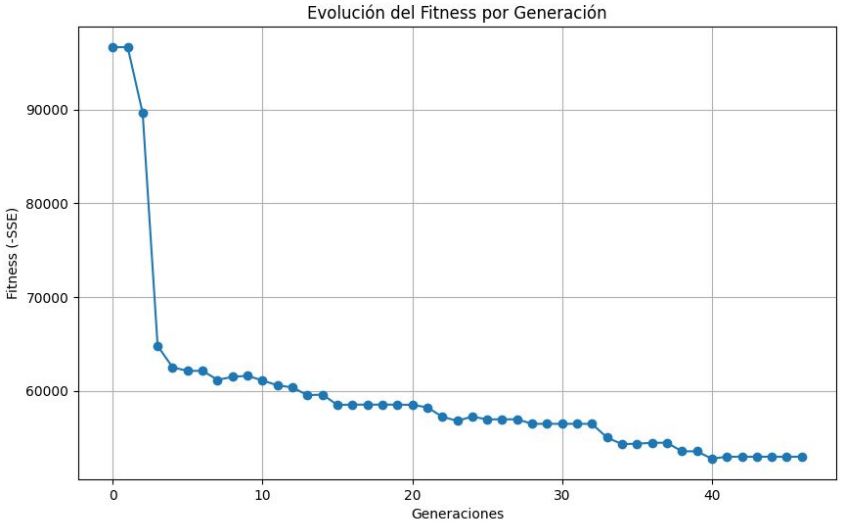




2 Clusters, Gráfica Género + Spending Score

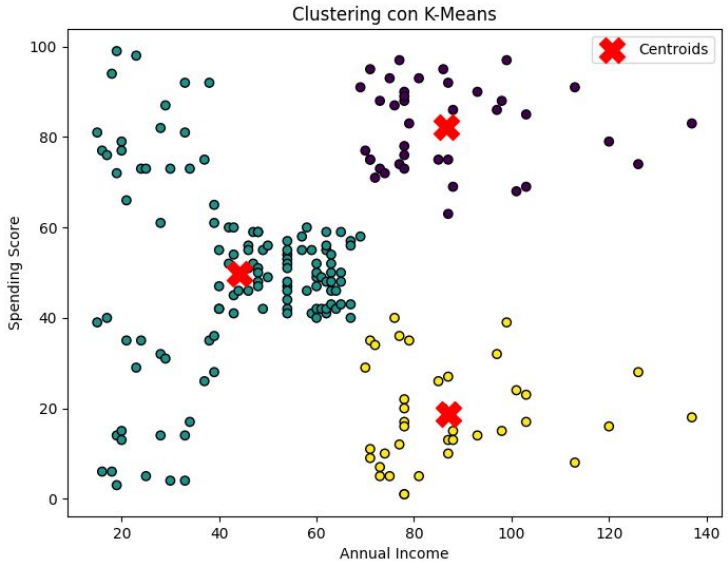


Resultados adicionales

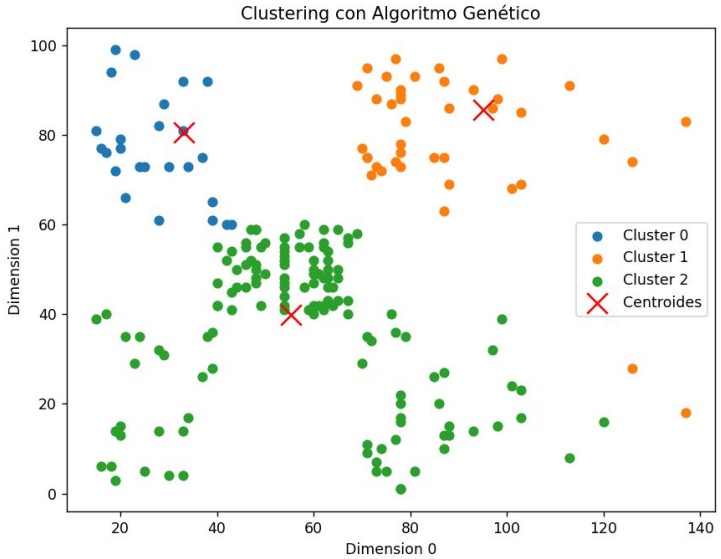


Prueba con K-means: Enlace al Código de K-Means

K-Means



Algoritmo Genético



Fin