UNIVERSIDAD DE GRANADA

Práctica 3.b:
Técnicas de Búsqueda basadas en Trayectorias para el Problema del Aprendizaje de Pesos en Características (APC)

Brao Aissaoni, Isaac
Grupo de prácticas 3 Jueves(17:30-19:30)

Índice

1. Descripción del problema: Problema del Aprendizaje de Pesos en Características (APC)

El problema consiste en la realización de la clasificación de datos en clases que ya son conocidas. La pertenencia de cada dato a una clase ya se conoce a priori.

El objetivo es que el modelo de clasificación sea capaz de clasificar los datos correctamente con un error mínimo, además de aprender la importancia de cada característica que describe a los datos. Para ello debe asignar distinta importancia a cada característica a la que le es asignada un peso distinto, el cual se tendrá en cuenta al momento de clasificar.

2. Representación del problema y algoritmos empleados

2.1 Codificación de los datos del problema

2.1.1 Representación del problema

El problema se representa con 5 matrices $n_i$xm donde n_i es el número de datos de cada partición del problema y m es el número de característica.

$$X_i = \begin{pmatrix} x_{i11} & \cdots & x_{i1m} \\ \cdots & \cdots & \\ x_{in_i1} & \cdots & x_{in_im} \end{pmatrix}$$

Donde i corresponde al número de la partición.

2.1.2 Representación de la solución

Los pesos están representados en un vector de tamaño m donde m es el número de características

$$w_i = \begin{pmatrix} w_{i1} & \cdots & w_{im} \end{pmatrix}$$

Donde i corresponde al número de la partición.

2.2 Función objetivo

La función fitness tendrá en cuenta la tasa de acierto sobre el conjunto de entrenamiento y la tasa de reducción de los pesos. Buscamos maximizar el fitness.

$$fitness = \alpha \cdot \%aciertos + (1 - \alpha) \cdot \%reducción \quad donde\, \alpha = 0.75\, en\, los\, problemas$$

```python
def fitness(X_train, y_train, X_test, y_test, pesos, alfa=0.75):
    predicciones = clasificador1NN(X_train, y_train, X_test, pesos)
    aciertos = accuracy(y_test, predicciones)
    tasa_red = tasa_reduccion(pesos)
    return alfa*aciertos + (1-alfa)*tasa_red
```

```python
def tasa_reduccion(pesos):
    cont = 0
    for p in pesos:
        if p >= 0.1:
            cont += 1

    return 100*(cont / len(pesos))
```

```python
def accuracy(y_true, y_pred):
    correctos = 0

    for i in range(len(y_true)):
        if y_true[i] == y_pred[i]:
            correctos += 1

    return correctos / len(y_true)
```

```python
def clasificador1NN(X_train, y_test, X_test, pesos=None, k=1):
    # Inicializar pesos a 1 si no se especifican simulando un KNN normal
    if pesos is None:
        pesos = []
        for i in range(n_columnas(X_train)):
            pesos.append(1)

    predictions = np.array(n_columnas(X_test), dtype=str)

    # Calcular matriz de distancias
    for i in range(n_filas(X_test)):
        distancias = np.zeros(n_filas(X_train))
        for j in range(n_filas(X_train)):
            distancias[j] = distanciaEuclidiana(X_test[i], X_train[j], pesos)

        indices_ord = ordenarPorDistancia(distancias, eje='fila')

        # Si elegimos el numero 1, el vecino más cercano es el segundo porque el primero es el mismo
        indice_cercano = indices_ord[1]

        # Asignar la clase del vecino más cercano
        clase = y_test[indice_cercano]

        # Asignar la clase al conjunto de predicciones
        predictions[i] = clase

    return predictions
```

2.3 Generación de soluciones aleatorias

```python
def generar_poblacion_inicial(n_poblacion, n_caracteristicas):
    # Inicializar la poblacion con una matriz de n_poblacion x n_caracteristicas de pesos aleatorios de forma uniforme
    poblacion = np.empty((n_poblacion, n_caracteristicas))

    for i in range(n_poblacion):
        for j in range(n_caracteristicas):
            poblacion[i][j] = np.random.uniform(0, 1)
```

2.4 Algoritmo BL

```python
def BL(X, y, individuo, fitness, max_eval):
    # Inicializar el numero de evaluaciones
    n_evaluaciones = 0
    fitness_individuo = fitness

    while n_evaluaciones < max_eval:
        # Obtener un orden de mutacion aleatorio
        orden_mutacion = np.random.permutation(len(individuo))

        for mut in orden_mutacion:
            # Obtener un vecino
            vecino = obtenerVecino(individuo, mut)

            # Calcular fitness del vecino
            fitness_vecino = fitness(X, y, vecino)
            n_evaluaciones += 1

            # Si el vecino es mejor, actualizamos pesos
            if fitness_vecino > fitness_individuo:
                individuo = vecino
                fitness_individuo = fitness_vecino

            # Si se alcanza el maximo de evaluaciones, salimos
            if n_evaluaciones >= max_eval:
                break

    return individuo, fitness_individuo, n_evaluaciones
```

2.4.1 Generación de vecinos

```python
def obtenerVecino(pesos, i):
    #Obtener una mutacion aleatoria de la distribucion normal de media 0 y varianza 0.3
    mutacion = np.random.normal(0, np.sqrt(0.3))
    vecino = pesos.copy()

    # Aplicar la mutacion
    vecino[i] += mutacion

    # Normalizar el peso cambiado
    vecino[i] = max(vecino[i], 0)
    vecino[i] = min(vecino[i], 1)

    return vecino

```

3. Algoritmo de enfriamiento simulado (ES)

```python
def fit_ES(X, y, individuo, max_eval):
    # Inicializar el numero de evaluaciones
    n_evaluaciones = 0

    # Inicializar el fitness del individuo
    fitness_individuo = fitness(X, y, individuo)

    # Peso actual
    actual = individuo
    fitness_actual = fitness_individuo

    # Guardamos el mejor individuo
    mejor_individuo = individuo
    fitness_mejor_individuo = fitness_individuo

    temperatura = temperatura_inicial(sigma=0.3, mu=0.1, eval=fitness_actual)
    temperatura_final = 10e-3

    # Inicializar maximo de vecinos, exitos y enfriamientos
    max_vecinos = 10*n_columnas(individuo)
    max_exitos = max_vecinos // 10
    max_enfriamientos = max_eval // max_vecinos

    beta = (temperatura - temperatura_final) / (max_enfriamientos * temperatura * temperatura_final)

    # Varible para saber si ha habido exito
    exito = True

    while n_evaluaciones < max_eval and exito:
        n_vecinos = 0
        n_exitos = 0

        while n_evaluaciones < max_eval and n_vecinos < max_vecinos and n_exitos < max_exitos:
            # Obetener el gen a mutar
            gen = np.random.randint(0, n_columnas(individuo))

            # Obtener un vecino
            vecino = obtenerVecino(actual, gen)

            # Calcular fitness del vecino
            fitness_vecino = fitness(X, y, vecino)
            n_evaluaciones += 1
            n_vecinos += 1

            delta = fitness_actual - fitness_vecino

            # Si el vecino es mejor,o se decide por probabilidad,empeorar se actualiza el actual
            if delta < 0 or np.random.uniform(0, 1) <= np.exp(delta/temperatura):
                actual = vecino
                fitness_actual = fitness_vecino
                n_exitos += 1

                # Si el vecino es mejor que el mejor individuo, lo actualizamos
                if fitness_vecino > fitness_mejor_individuo:
                    mejor_individuo = vecino
                    fitness_mejor_individuo = fitness_vecino

        # Enfriar la temperatura
        temperatura = enfriar(temperatura, beta)
        # Comprobar si ha habido exito
        exito = n_exitos > 0

    return mejor_individuo, fitness_mejor_individuo, n_evaluaciones
```

3.1 Cálculo de la temperatura inicial

$$T_0 = \frac{\mu \cdot C(S_0)}{-\ln(\varphi)}$$

```python
def temperatura_inicial(sigma, mu, eval):
    # La ecuacion es mu*eval/-ln(sigma)
    return mu*eval/-np.log(sigma)
```

3.2 Esquema de enfriamiento

Cauchy modificado

$$T_{k+1} = \frac{T_k}{1+\beta \cdot T_k} \quad ; \quad \beta = \frac{T_0 - T_f}{M \cdot T_0 \cdot T_f}$$

```python
def enfriar(temperatura, beta):
    # Esquema de enfriamiento Cauchy modificado
    return temperatura / (1 + beta*temperatura)
```

4. Algoritmo multiarranque básico (BMB)

```python
def fit_BMB(X, y, max_iter, eval_bl, semilla=7):
    # Semilla para inicializar pesos aleatorios
    np.random.seed(semilla)

    # Inicializar pesos con una distribución uniforme con el tamaño de columnas de X
    soluciones_iniciales = generar_poblacion_inicial(max_iter, n_columnas(X))
    fitness_soluciones = np.empty(max_iter)

    # Mejorar las soluciones iniciales con busqueda local
    for i in range(max_iter):
        soluciones_iniciales[i], fitness_soluciones[i], _ = BL(X, y, soluciones_iniciales[i], eval_bl)

    # Obtener la mejor solucion
    mejor_solucion = soluciones_iniciales[np.argmax(fitness_soluciones)]

    return mejor_solucion
```

5. Algoritmo Iterative Local Search (ILS)

```python
def fit_ILS(X, y, tipo_fit, max_iter, eval_bl, prob_mutacion, semilla=7):
    # Semilla para inicializar pesos aleatorios
    np.random.seed(semilla)

    # Elegir el la funcion de fit que se va a utilizar, si BL o ES
    match tipo_fit:
        case 'BL':
            fit = BL
        case 'ES':
            fit = fit_ES
        case _:
            raise ValueError("Tipo de fit no válido.")

    # Inicializar pesos con una distribución uniforme con el tamaño de columnas de X
    solucion_actual = np.random.uniform(0, 1, n_columnas(X))

    # Calcular el fitness de la solucion actual
    solucion_actual, fitness_actual, _ = fit(X, y, solucion_actual, eval_bl)
    iteraciones = 1

    # Almacenar la mejor solucion
    mejor_solucion = solucion_actual
    mejor_fitness = fitness_actual

    while iteraciones < max_iter:
        # Mutar la mejor solucion
        solucion_actual = mutacion_ILS(mejor_solucion, prob_mutacion)

        #Mejorar la solucion mutada
        solucion_actual, fitness_actual, _ = fit(X, y, solucion_actual, eval_bl)
        iteraciones += 1

        # Si la solucion actual es mejor que la mejor, la actualizamos
        if fitness_actual > mejor_fitness:
            mejor_solucion = solucion_actual
            mejor_fitness = fitness_actual

    return mejor_solucion
```

5.1 Operador mutación

```python
def mutacion_ILS(solucion, prob_mutacion):
    # Elelgir genes aleatorios a mutar
    genes = np.random.uniform(0, 1, n_columnas(solucion)) < prob_mutacion

    # HSi hay almenos 3 genes a mutar, necesitamos al menos 3 genes que mutar
    if n_columnas(solucion) > 3:
        while np.sum(genes) < 3:
            genes[np.random.randint(0, n_columnas(solucion))] = True

    # Mutar los genes
    nueva_solucion = solucion.copy()

    for i in range(n_columnas(solucion)):
        if genes[i]:
            # Variar el gen en un rango de [-0.25, 0.25]
            nueva_solucion[i] = nueva_solucion[i] + np.random.uniform(-0.25, 0.25)

            # Poner el gen en el rango [0, 1]
            nueva_solucion[i] = max(nueva_solucion[i], 0)
            nueva_solucion[i] = min(nueva_solucion[i], 1)

    return nueva_solucion
```

6. Procedimiento de desarrollo de la práctica

6.1 Software utilizado para el desarrollo del código fuente

La práctica se ha realizado en el lenguaje de programación python. Se ha usado **numpy** para las operaciones vectoriales y matriciales; **scipy** para el cálculo de distancias euclidianas, con peso y sin pesos para el KNN; **scikit learn** para la normalización y escalado de los datos; **pandas** para las tablas, estadísticas y guardar la salida en formato csv; el paquete **liac-arff** para la lectura de los datos y el paquete **tqdm** para las barras de progreso.

6.2 Manual de usuario

1. **Inicio del Programa:**
   Para ejecutar el script usar python3 main.py

   Al ejecutar el script, se te solicitará introducir la base de datos que deseas utilizar.
   Las opciones disponibles son:

   - **BreastCancer** (Por defecto): Base de datos sobre cáncer de mama.
   - **Ecoli**: Base de datos sobre secuencias de ADN de bacterias E. coli.
   - **Parkinson**: Base de datos sobre la detección de la enfermedad de Parkinson.

2. **Selección del Modelo:**
   Después de seleccionar la base de datos, se te pedirá que elijas el modelo de aprendizaje automático que deseas utilizar. Las opciones son:

   - **KNN** (Por defecto): K-Nearest Neighbors.
   - **Relief**: Algoritmo de selección de características Relief.
   - **BL**: Algoritmo Búsqueda Local.
   - **AGG**: Algoritmo Genético Generacional.
   - **AGE**: Algoritmo Genético Estacionario.
   - **AM**: Algoritmo Memético.
   - **BMB**: Algoritmo Multiarranque básico.
   - **ILS**: Algoritmo Iterative Local Search.
   - **ILS-ES**: Algoritmo Iterative Local Search con Enfriamiento Simulado.
   - **ES**: Algoritmo Enfriamiento Simulado.
   - **P3**: Ejecutar los algoritmos de la práctica 3.
   - **ALL**: Ejecutar todos los modelos disponibles.

3. **Configuración Adicional:**
   Dependiendo del modelo seleccionado, puede ser necesario introducir configuraciones adicionales:
   - **Valor de k**: Relevante para el modelo KNN y la opción ALL. Representa el número de vecinos más cercanos a considerar.

- **Valor de la semilla**: Requerido para los modelos BL, AGG, AGE, AM y la opción ALL. La semilla se utiliza para inicializar la generación de números aleatorios.
- **Versión Mejorada**: Para los modelos AGG, AGE, AM y la opción ALL, se te preguntará si deseas utilizar la versión mejorada de los algoritmos genéticos.
- **Operador de Cruce**: Para AGG y AGE, se te solicitará seleccionar un operador de cruce entre 'CA' (Cruce Aritmético) y 'BLX' (Blend Crossover).
- **Operador de Selección en BL para el algoritmo memético**: Para el modelo AM, deberás seleccionar un operador de selección entre 'All', 'Random' y 'Best'.

4. **Ejecución del Modelo:**
   Una vez que se han introducido todas las configuraciones, el modelo seleccionado se ejecutará con las opciones especificadas.

5. **Guardar Resultados en Archivo CSV:**
   Puedes seleccionar `S` para guardar los resultados en un archivo CSV o `N` para no guardarlos.

6. **Requisitos del Programa:**
   Este programa requiere los siguientes paquetes de Python instalados en tu sistema:

   - **numpy**
   - **scipy**
   - **pandas**
   - **liac-arff**
   - **scikit-learn**
   - **tqdm**

7. **Errores y Salida del Programa:**
   Si introduces una base de datos o modelo no válido, el programa mostrará un mensaje de error correspondiente y se cerrará.

6.3 Consideraciones a tener en cuenta

1. Si tiene instalado el paquete arff de google tendrá que desinstalarlo e instalar a continuación liac-arff porque generan incompatibilidad entre ellos.
2. Los valores utilizados para el análisis de los resultados son: elegir la opción P3 en la selección de algoritmos y la semilla 7.

7. Experimentos y resultados

Los experimentos han sido realizados por el parámetro con la semilla 7 para los algoritmos, ejecutando los algoritmos de la P3.

7.1 Ecoli

7.1.1 Resultados

BL						
Partición	Tasa de clasificación	Tasa de reducción	Fitness train	Fitness test	Accuracy	Tiempo de ejecución
1	80.08	57.14	74.34	73.21	78.57	0.09
2	80.08	57.14	74.34	65.71	68.57	0.05
3	80.60	57.14	74.73	71.64	76.47	0.07
4	75.00	57.14	70.54	67.23	70.59	0.05
5	81.16	42.86	71.58	74.46	85.00	0.07
Media	79.38	54.29	73.11	70.45	75.84	0.07
Std dev	2.49	6.39	1.91	3.81	6.56	0.02

BMB						
Partición	Tasa de clasificación	Tasa de reducción	Fitness train	Fitness test	Accuracy	Tiempo de ejecución
1	81.58	57.14	75.47	72.14	77.14	1.17
2	80.83	57.14	74.91	63.57	65.71	1.14
3	80.97	57.14	75.01	72.74	77.94	1.14
4	73.88	71.43	73.27	70.80	70.59	1.23
5	79.35	57.14	73.80	78.04	85.00	1.31
Media	79.32	60.00	74.49	71.46	75.28	1.20
Std dev	3.15	6.39	0.92	5.20	7.39	0.07

ILS						
Partición	Tasa de clasificación	Tasa de reducción	Fitness train	Fitness test	Accuracy	Tiempo de ejecución
1	81.58	57.14	75.47	73.21	78.57	1.00
2	75.56	71.43	74.53	58.57	54.29	0.80
3	73.88	71.43	73.27	67.49	66.18	0.72
4	73.88	71.43	73.27	70.80	70.59	0.77
5	79.35	57.14	73.80	78.04	85.00	0.89
Media	76.85	65.71	74.07	69.62	70.92	0.84
Std dev	3.46	7.82	0.94	7.28	11.79	0.11

ILS-ES						
Partición	Tasa de clasificación	Tasa de reducción	Fitness train	Fitness test	Accuracy	Tiempo de ejecución
1	81.95	57.14	75.75	72.14	77.14	3.45
2	80.83	57.14	74.91	63.57	65.71	3.45
3	80.97	57.14	75.01	72.74	77.94	3.47
4	73.88	71.43	73.27	70.80	70.59	3.35
5	79.35	57.14	73.80	78.04	85.00	3.53
Media	79.40	60.00	74.55	71.46	75.28	3.45
Std dev	3.22	6.39	1.00	5.20	7.39	0.07

ES						
Partición	Tasa de clasificación	Tasa de reducción	Fitness train	Fitness test	Accuracy	Tiempo de ejecución
1	79.70	57.14	74.06	71.07	75.71	3.53
2	80.83	57.14	74.91	63.57	65.71	3.48
3	73.88	71.43	73.27	67.49	66.18	3.32
4	78.36	57.14	73.05	74.95	80.88	3.51
5	76.09	57.14	71.35	74.29	80.00	3.67
Media	77.77	60.00	73.33	70.27	73.70	3.50
Std dev	2.80	6.39	1.32	4.78	7.34	0.13

7.1.2 Resultados globales

	Ecoli					
Algoritmo	Tasa de clasificación	Tasa de reducción	Fitness train	Fitness test	Accuracy	Tiempo de ejecución
BL	79.38	54.29	73.11	70.45	75.84	0.07
AM-Best	80.29	57.14	74.50	72.50	77.62	1.80
BMB	79.32	60.00	74.49	71.46	75.28	1.20
ILS	76.85	65.71	74.07	69.62	70.92	0.84
ILS-ES	79.40	60.00	74.55	71.46	75.28	3.45
ES	77.77	60.00	73.33	70.27	73.70	3.50

7.2 Parkinson's

7.2.1 Resultados

	BL					
Partición	Tasa de clasificación	Tasa de reducción	Fitness train	Fitness test	Accuracy	Tiempo de ejecución
1	94.19	81.82	91.10	89.83	92.50	0.19
2	96.77	77.27	91.90	90.57	95.00	0.14
3	97.42	77.27	92.38	90.57	95.00	0.17
4	98.06	81.82	94.00	89.83	92.50	0.16
5	97.50	81.82	93.58	89.03	91.43	0.12
Media	96.79	80.00	92.59	89.96	93.29	0.16
Std dev	1.52	2.49	1.20	0.64	1.62	0.03

BMB						
Partición	Tasa de clasificación	Tasa de reducción	Fitness train	Fitness test	Accuracy	Tiempo de ejecución
1	98.06	77.27	92.87	94.32	100.00	1.99
2	98.06	77.27	92.87	90.57	95.00	1.99
3	98.71	77.27	93.35	88.69	92.50	1.90
4	98.06	81.82	94.00	89.83	92.50	1.98
5	98.13	81.82	94.05	93.31	97.14	2.03
Media	98.21	79.09	93.43	91.34	95.43	1.98
Std dev	0.28	2.49	0.58	2.38	3.21	0.05

ILS						
Partición	Tasa de clasificación	Tasa de reducción	Fitness train	Fitness test	Accuracy	Tiempo de ejecución
1	98.71	77.27	93.35	94.32	100.00	1.35
2	96.13	81.82	92.55	91.70	95.00	1.24
3	97.42	81.82	93.52	87.95	90.00	1.33
4	98.06	81.82	94.00	89.83	92.50	1.33
5	98.13	81.82	94.05	93.31	97.14	1.26
Media	97.69	80.91	93.49	91.42	94.93	1.30
Std dev	0.98	2.03	0.61	2.58	3.90	0.05

ILS-ES						
Partición	Tasa de clasificación	Tasa de reducción	Fitness train	Fitness test	Accuracy	Tiempo de ejecución
1	98.71	77.27	93.35	92.44	97.50	2.14
2	97.42	77.27	92.38	90.57	95.00	2.14
3	98.71	77.27	93.35	92.44	97.50	2.15
4	97.42	81.82	93.52	89.83	92.50	2.10
5	98.13	81.82	94.05	93.31	97.14	2.41
Media	98.08	79.09	93.33	91.72	95.93	2.19
Std dev	0.65	2.49	0.60	1.46	2.18	0.12

ES						
Partición	Tasa de clasificación	Tasa de reducción	Fitness train	Fitness test	Accuracy	Tiempo de ejecución
1	98.06	72.73	91.73	87.56	92.50	2.31
2	98.06	77.27	92.87	90.57	95.00	2.23
3	98.71	77.27	93.35	88.69	92.50	2.22
4	98.06	72.73	91.73	89.43	95.00	2.38
5	97.50	72.73	91.31	84.61	88.57	2.43
Media	98.08	74.55	92.20	88.17	92.71	2.31
Std dev	0.43	2.49	0.87	2.27	2.63	0.09

7.2.2 Resultados globales

	Parkinsons					
Algoritmo	Tasa de clasificación	Tasa de reducción	Fitness train	Fitness test	Accuracy	Tiempo de ejecución
BL	96.79	80.00	92.59	89.96	93.29	0.16
AM-Best	97.31	81.82	93.44	89.62	92.21	1.59
BMB	98.21	79.09	93.43	91.34	95.43	1.98
ILS	97.69	80.91	93.49	91.42	94.93	1.30
ILS-ES	98.08	79.09	93.33	91.72	95.93	2.19
ES	98.08	74.55	92.20	88.17	92.71	2.31

7.3 Breast Cancer

7.3.1 Resultados

BL						
Partición	Tasa de clasificación	Tasa de reducción	Fitness train	Fitness test	Accuracy	Tiempo de ejecución
1	96.04	86.67	93.69	92.10	93.91	0.77
2	96.48	90.00	94.86	93.59	94.78	0.81
3	97.58	86.67	94.85	89.49	90.43	0.68
4	97.58	80.00	93.18	88.48	91.30	0.89
5	96.96	86.67	94.38	93.23	95.41	1.40
Media	96.92	86.00	94.19	91.38	93.17	0.91
Std dev	0.68	3.65	0.74	2.28	2.19	0.28

BMB						
Partición	Tasa de clasificación	Tasa de reducción	Fitness train	Fitness test	Accuracy	Tiempo de ejecución
1	97.14	90.00	95.35	94.89	96.52	11.88
2	95.37	90.00	94.03	94.89	96.52	11.98
3	97.80	90.00	95.85	92.28	93.04	12.21
4	96.26	90.00	94.69	90.98	91.30	12.91
5	96.30	90.00	94.73	94.75	96.33	12.42
Media	96.57	90.00	94.93	93.56	94.74	12.28
Std dev	0.93	0.00	0.69	1.82	2.43	0.41

ILS						
Partición	Tasa de clasificación	Tasa de reducción	Fitness train	Fitness test	Accuracy	Tiempo de ejecución
1	96.04	90.00	94.53	93.59	94.78	8.58
2	95.81	90.00	94.36	92.93	93.91	7.56
3	97.36	90.00	95.52	88.37	87.83	7.72
4	96.70	90.00	95.02	91.63	92.17	8.20
5	97.61	83.33	94.04	93.08	96.33	9.51

Media	96.70	88.67	94.69	91.92	93.01	8.31
Std dev	0.79	2.98	0.58	2.11	3.26	0.78

ILS-ES						
Partición	Tasa de clasificación	Tasa de reducción	Fitness train	Fitness test	Accuracy	Tiempo de ejecución
1	97.14	90.00	95.35	94.89	96.52	10.35
2	96.26	90.00	94.69	93.59	94.78	11.72
3	97.80	90.00	95.85	92.93	93.91	10.78
4	97.58	90.00	95.68	90.98	91.30	10.19
5	95.65	90.00	94.24	93.37	94.50	10.65
Media	96.88	90.00	95.16	93.15	94.20	10.74
Std dev	0.91	0.00	0.68	1.42	1.89	0.60

ES						
Partición	Tasa de clasificación	Tasa de reducción	Fitness train	Fitness test	Accuracy	Tiempo de ejecución
1	94.27	80.00	90.70	89.78	93.04	10.85
2	96.26	86.67	93.86	92.10	93.91	9.45
3	97.80	86.67	95.01	88.84	89.57	9.57
4	98.02	90.00	96.01	91.63	92.17	9.35
5	96.74	83.33	93.39	90.33	92.66	10.54
Media	96.62	85.33	93.80	90.54	92.27	9.95
Std dev	1.50	3.80	2.01	1.34	1.64	0.69

7.3.2 Resultados globales

	Breast Cancer					
Algoritmo	Tasa de clasificación	Tasa de reducción	Fitness train	Fitness test	Accuracy	Tiempo de ejecución
BL	96.92	86.00	94.19	91.38	93.17	0.91
AM-Best	97.01	90.00	95.26	94.09	95.45	7.35
BMB	96.57	90.00	94.93	93.56	94.74	12.28
ILS	96.70	88.67	94.69	91.92	93.01	8.31
ILS-ES	96.88	90.00	95.16	93.15	94.20	10.74
ES	96.62	85.33	93.80	90.54	92.27	9.95

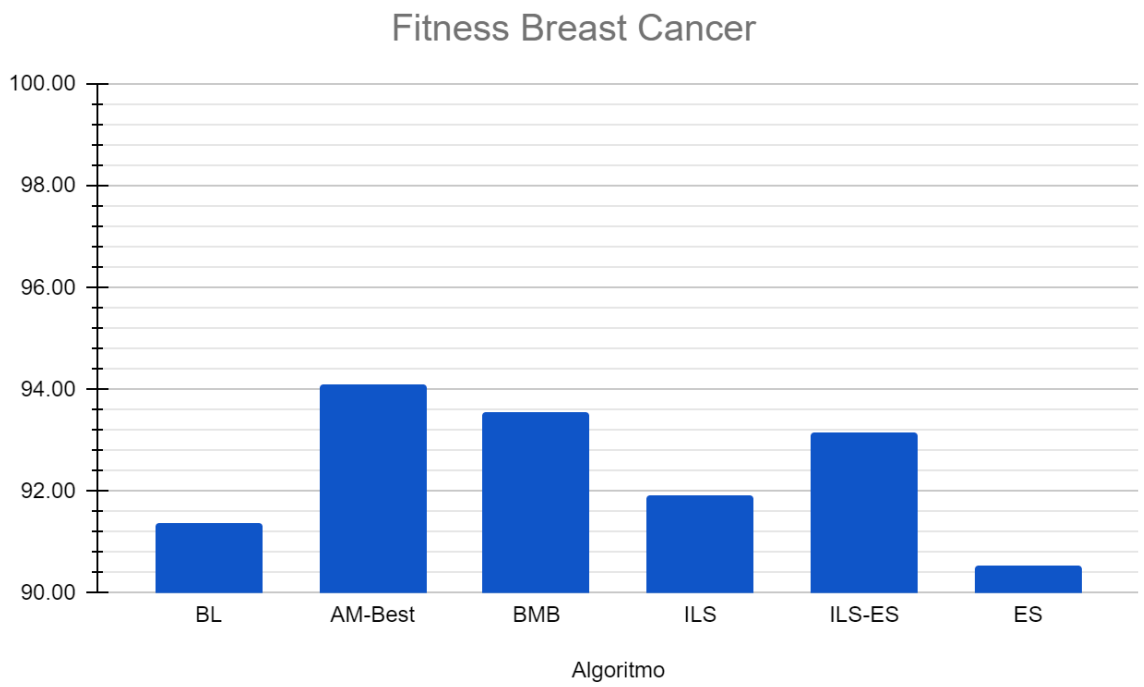
7.4 Todos los resultados globales

	Breast Cancer			Ecoli			Parkinsons		
Algoritmo	Tasa clas	Tasa red	Fitness	Tasa clas	Tasa red	Fitness	Tasa clas	Tasa red	Fitness
BL	96.92	86.00	91.38	79.38	54.29	70.45	96.79	80.00	89.96
AM-Best	97.01	90.00	94.09	80.29	57.14	72.50	97.31	81.82	89.62
BMB	96.57	90.00	93.56	79.32	60.00	71.46	98.21	79.09	91.34
ILS	96.70	88.67	91.92	76.85	65.71	69.62	97.69	80.91	91.42
ILS-ES	96.88	90.00	93.15	79.40	60.00	71.46	98.08	79.09	91.72
ES	96.62	85.33	90.54	77.77	60.00	70.27	98.08	74.55	88.17

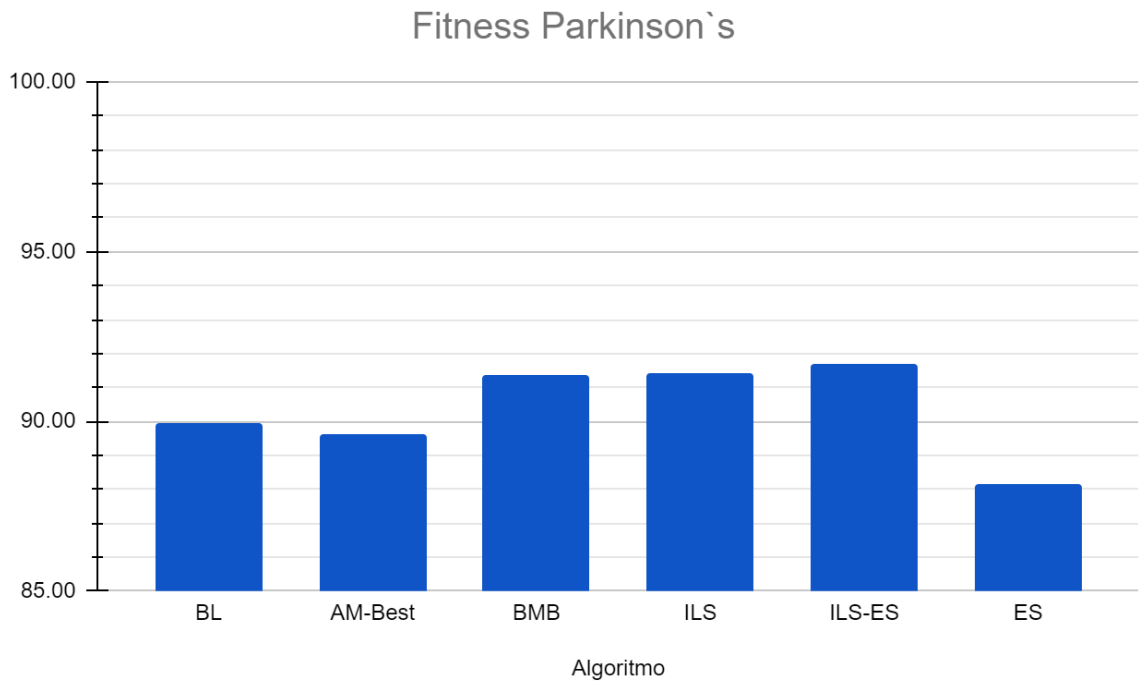
8. Análisis de resultados

8.1 Gráficas

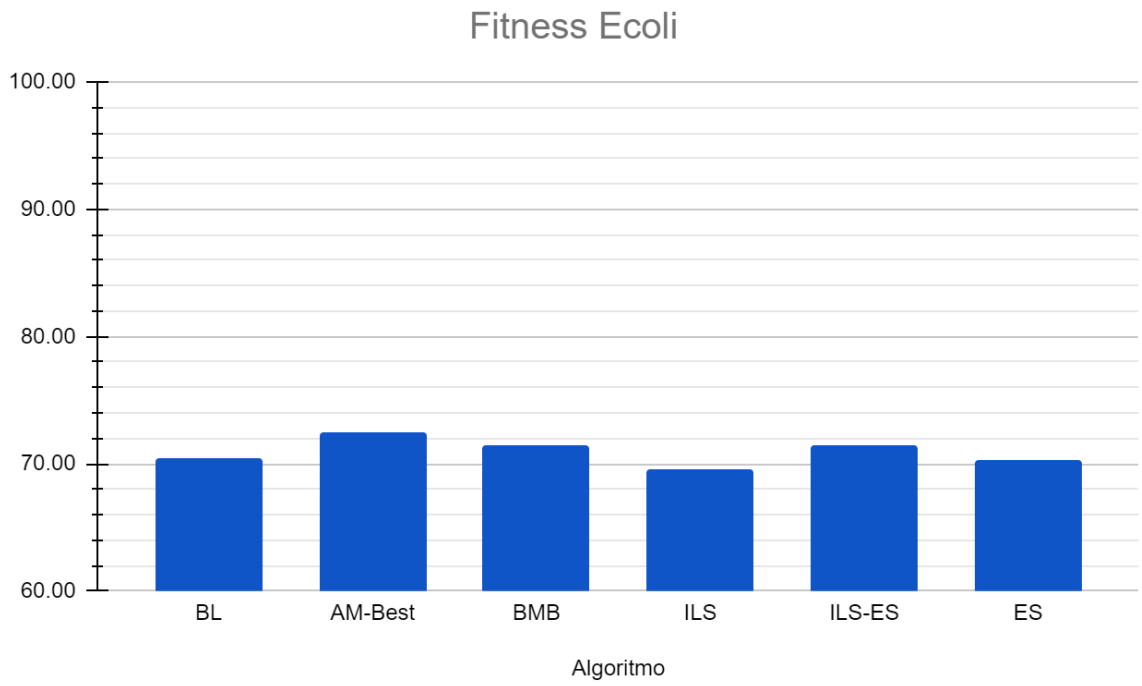
8.1.1 Fitness en Breast Cancer



8.1.2 Fitness en Parkinson's



8.1.3 Fitness en Ecoli



8.2 BL vs ES

ES me ha dado peores resultados que BL además de ser el peor algoritmo entre todos los algoritmos evaluados. Creo que es debido a la variable n_vecinos. Con n_vecinos = 10*n da resultados malos porque genera demasiados vecinos, un mejor valor es de 5*n que mejora los resultados en torno al 1-2% en fitness, igualando o mejorando el rendimiento de BL.

En conclusión, empeorar demasiado no es bueno, pero empeorar es positivo si se hace de manera más controlada, para conseguir la convergencia del algoritmo. Pero son los dos peores algoritmos entre todos los comparados.

8.3 BMB vs ILS vs ILS-ES vs AM-Best

BMB en todos los casos es mejor al ILS, pero iguala el rendimiento de ILS-ES. Creo que se debe a que ES añade la diversidad que le falta al ILS tradicional. Que BMB y ILS-ES rindan similar se puede deber a que ambos tienen un nivel de diversidad similar.

Tanto BMB como ILS-ES rinden muy bien, superando a algoritmos genéticos de la P2, por tanto esto indica que añadir multi arranque o reinicio de poblaciones es muy positivo para los algoritmos de búsqueda.

El único algoritmo al que no superan es al algoritmo memético mejorado que implemente en la P2, ya que tiene una ligera componente de multiarranque con el BL aplicado al algoritmo y a la alta diversidad que añadí al algoritmo aumentando la probabilidad de mutación. Pero en contrapartida BMB y ILS-ES son algoritmos más consistentes siempre dando buenos resultados.

9. Conclusión

Para tener un ***buen algoritmo de búsqueda*** hay que tener en cuenta mantener una ***buena diversidad***, ***conseguir la convergencia*** del algoritmo y ***añadir reinicios*** al algoritmo cuando el algoritmo converja, pudiendo explorar más el espacio de búsqueda para poder encontrar otras soluciones.