UNIVERSIDAD DE GRANADA

Aprendizaje de Pesos en Características

30 de marzo 2022

Metaheurísticas

Gallego Menor, Francisco Javier
javigallego@correo.ugr.es
74745747W

Índice

1 Introducción

El problema de clasificación consiste en, dado un conjunto $A = \{(a, b) \ : a \in R^n, b \ es \ una \ clase\}$ de datos ya clasificados, obtener un sistema que permita clasificar un objeto nuevo de forma automática.

Un ejemplo de clasificador, y el que utilizaremos en esta práctica,es el $k - NN$, k vecinos más cercanos. Este toma la clase que más se repita entre los $u_i \in A$ tales que su distancia al nuevo elemento u sea mínima. En nuestro caso, en una versión sencilla del problema, consideraremos el clasificador $1-NN$.

Consideraremos como distancias la distancia trivial si las características son discretas (esto es, la distancia será 1 si las características son diferentes, y 0 si son iguales. La denotamos como d_n), y la distancia euclídea para características que sean continuas. Además, cada característica tendrá un peso asociado, por lo que dado un vector de pesos w, la distancia entre dos vectores u y v será de la forma:

$$d(u, v) = \sqrt{\sum_i w_i(u_i - v_i)^2 + \sum_j w_j d_n(u_j, v_j)}$$

El aprendizaje de pesos en características consiste en hallar un vector de pesos que maximice la siguiente función:

$$F(w) = \alpha T_{clas}(w) + (1 - \alpha)T_{red}(w)$$

Donde

- T_{clas} es la función que indica cómo de bueno es nuestro clasificador, es decir, cuántos casos ha clasificado correctamente si entrenamos el clasificador usando el resto de datos ,la técnica *k-fold cross validation*, y dejando un elemento fuera (leave one out).
- T_{red} que es la función que nos indica cuántas características de un dato tienen un peso menor que un valor establecido, en nuestro caso 0.2.

2 Descripción de la aplicación de los algoritmos

2.1 Esquemas de representación

En nuestro problema, los datos de entrada poseen los elementos que siguen:

- **Clase del elemento (target)**: que es la categoría a la que corresponde el mismo, dependiendo de cada dataset
- **Ejemplo**: que es un par que tiene un vector de features y una clase (target).
- **Dataset**: que contendrá una lista de ejemplos
- **Vector de características**: vector de valores reales que trataremos de normalizar al intervalo $[0, 1]$ para trabajar con ellos.

A partir de los elementos previamente descritos, vamos a obtener nuestra solución al problema. Esta consistirá en un vector de pesos, cuyos valores reales se encontrarán también en el intervalo $[0, 1]$.

2.2 Operadores comunes

En esta sección, procederemos a describir aquellas funcionalidades que sean comunes para todos los algoritmos. En nuestro caso, solo es la que sigue:

La función *weighted_onenn* es la función que nos da el vecino más cercano a otro utilizando el algoritmo $1 - NN$, usando un vector de pesos dado. La utilizaremos dando diferentes vectores de pesos w para obtener diferentes resultados de clasificación.

1: **procedure** WEIGHTED-ONENN(weight, features, target, examples)
2: dist $\leftarrow \sqrt{\sum_i weight_i * (features_i - example_i)^2}$
3: **return** targets[minimo(dist)]

El resto de funcionalidades que se han utilizado se han hecho en cada algoritmo concreto, debido a la facilidad que nos da el lenguaje de programación escogido para la programación de estos operadores.

2.3 Función objetivo

En nuestro caso, se nos indica que tomemos como *alpha* el valor 0.5, así que en realidad lo que estamos haciendo es:

$$F = 0.5(T_{clas} + T_{red})$$

Calcularemos T_{class} de la siguiente manera:

1: **procedure** T-CLASS(w, features, targets)
2: rights = 0
3: *loop* : train index, test index in LeaveOneOut.split(features)
4: **if** weighted 1-NN clasifica correctamente **then** *rights* $\leftarrow$ *rights* + 1
5: *endloop*
6: **return** porcentaje de aciertos

Y calcularemos T_{red} así:

1: **procedure** T-RED(weight)
2: **return** (Nº Pesos < 0.2) / Nº Pesos Totales

3 Estructura del método de búsqueda

3.1 Algoritmo Greedy Relief

El algoritmo de Greedy Relief se encarga de recorrer todo el conjunto de datos, muestra por muestra. En cada uno de ellos, dependiendo del amigo y enemigo más cercanos, hace una modificación del vector de pesos. Entonces, lo que haremos será: por cada una de las muestras, obtener un conjunto de datos amigos (que pertenecen a su misma clase) y uno de enemigos (no pertenecen a su misma clase). Posteriormente, obtendremos el vecino más cercano de cada uno de ellos.

Algorithm 1 Greedy Relief

```
1:  procedure greedyRelief(features, targets)
2:      w ← [0, ...., 0]
3:      distances ← matriz cuadrada de distancias (euclídeas)
4:  loop: para cada muestra del conjunto de entrenamiento
5:      friends ← Amigos del sample i-ésimo
6:      enemies ← Enemigos del sample i-ésimo
7:
8:      closestFriend ← Amigo mas cercano
9:      closestEnemy ← Enemigo mas cercano
10:
11:     w ← w + |features(i) - closestEnemy| - |features(i)-closestFriend|
12: endloop
13:     w ← Truncamos valores negativos a 0
14:     w ← Normalizamos w
15:     return w
```

3.2 Búsqueda local

Para este algoritmo, debemos definir el algoritmo que hemos usado para obtener una mutación de un ejemplo. Al mutar, sumaremos a la componente $j-esima$ un valor aleatorio y truncaremos al intervalo $[0, 1]$ si el nuevo valor se nos escapara del intervalo:

```
1: procedure mute(w,sigma,j)
2:     w(j) ← w(j) + random(0, 1)
3:     w(j) ← Normalize(w(j))
4:     return w
```

En el procedimiento de cálculo de pesos mediante la búsqueda local intervendrá la función de evaluación, que será notada por $f(w)$.

Así, el procedimiento general para la generación de pesos para la búsqueda local sería:

Algorithm 2 Local Search

```
1:  procedure localSearch(initialWeight,features,targets)
2:      weight ← valor inicial para los pesos
3:      bestF ← valor inicial para la función objetivo
4:      index ← Ordenación de forma aleatoria de los índices
5:      improve ← False
6:  while (menos evaluaciones que 15000) and (Haya mejora antes de generar 20*n vecinos)
7:      w ← vector de pesos mutando componente j-esima
8:      newF ← Valor funcion objetivo para w
9:      if newF > bestF then
          bestF ← newF
          weight ← w
          notMuted ← 0
          improve ← True
10:     else
          notMuted ← notMuted + 1
11:     if (Hay mejora) or (Todos los componentes han sido ya mutados) then
          improve ← False
          index ← Ordenación de forma aleatoria de los índices
12: endloop
13:     return weight
```

4 Procedimiento considerado para desarrollar la práctica

Para desarrollar la práctica, he usado **Python** como lenguaje de programación, sin usar ningún framework de metaheurísticas.

Para poder ejecutar el código , hace falta tener instalado *Numpy, Scipy* y *Sklearn*. Este último es muy útil para la realización de prácticas de este estilo pues trae implementaciones de muchas funcionalidades básicas para *Machine learning*.

El fichero que hay que ejecutar dentro de la carpeta es el fichero ***main.py***. Para ello, basta con escribir en la terminal:

python main.py

Tras la ejecución, comenzará a ejecutar los algoritmos sobre los 3 ficheros de datos que tenemos, que se explicarán más adelante.

La lista de archivos que contiene la práctica son los siguientes:

- **main.py**: fichero principal a ejecutar para la ejecución de nuestro programa.
- **algorithms.py**: fichero en el que se encuentran los algoritmos y algunas funciones auxiliares programadas para la práctica.
- **metaheuristicas.ipynb**: notebook mediante el cual he generado las gráficas que aparecen en la parte final de la memoria.

- **Datasets**: carpeta en la que se encuentran los datasets almacenados.

5 Experimentos y análisis de resultados

5.1 Casos del problema

Vamos a comentar primero los ficheros que tenemos a analizar mediante nuestros algoritmos.

5.1.1 Ionosphere

Son los datos de radar recogidos por un sistema en Goose Bay, Labrador. Este sistema consiste en un conjunto de fases de 16 antenas de alta frecuencia con una potencia total transmitida del orden de 6.4 kilovatios Los objetivos son electrones libres en la ionosfera. Los "buenosretornos de radar son aquellos que muestran evidencia de algún tipo de estructura en la ionosfera.

- Consta de 352 ejemplos.
- Tiene 34 atributos.
- Presenta dos clases distintas, buenos y malos.

5.1.2 Parkinson

contiene datos que se utilizan para distinguir entre la presencia y la ausencia de la enfermedad de Parkinson en una serie de pacientes a partir de medidas biomédicas de la voz

- Consta de 195 ejemplos
- Consta de 23 atributos (clase incluida)
- Consta de 2 clases. La distribución de ejemplos está desbalanceada (147 enfermos, 48 sanos)
- Atributos: Frecuencia mínima, máxima y media de la voz, medidas absolutas y porcentuales de variación de la voz, medidas de ratio de ruido en las componentes tonales, ...

5.1.3 Specft-Heart

contiene atributos calculados a partir de imágenes médicas de tomografía computerizada (SPECT) del corazón de pacientes humanos. La tarea consiste en determinar si la fisiología del corazón analizado es correcta o no.

- Consta de 267 ejemplos.
- Consta de 45 atributos enteros (clase incluida).
- Consta de 2 clases (paciente sano o patología cardiaca)
- Atributos: 2 características (en reposo y en esfuerzo) de 22 regiones de interés de las imágenes

5.2 Resultados

Vamos a comentar los resultados que se han obtenido. Hay que indicar que he redondeado a dos decimales, excepto en la columna del tiempo de ejecución. Esto úlitmo debido a que para los algoritmos de 1-NN y Greedy Relief los tiempos de ejecución en segundos son muy pequeños. Introduzco las siglas de la tabla, y a continuación los resultados obtenidos:

- T_{class} es la tasa de clasificación, porcentaje de acierto
- T_{red} es la tasa de reducción
- Agr es el valor de la función objetivo con $\alpha = 0.5$
- T es el tiempo en segundos

```
Results for 1NN
Partition        Ionosphere                        Parkinsons                        Spectf-heart
        T_clas - T_red - Agr - T          T_clas - T_red - Agr - T          T_clas - T_red - Agr - T
0        0.85 - 0.00 - 0.42 - 0.00282 -    0.72 - 0.00 - 0.36 - 0.00104 -    0.99 - 0.00 - 0.49 - 0.00322 -
1        0.77 - 0.00 - 0.39 - 0.00280 -    0.82 - 0.00 - 0.41 - 0.00103 -    0.93 - 0.00 - 0.46 - 0.00320 -
2        0.83 - 0.00 - 0.41 - 0.00280 -    0.95 - 0.00 - 0.47 - 0.00104 -    0.77 - 0.00 - 0.39 - 0.00324 -
3        0.91 - 0.00 - 0.46 - 0.00278 -    0.74 - 0.00 - 0.37 - 0.00104 -    0.74 - 0.00 - 0.37 - 0.00311 -
4        0.86 - 0.00 - 0.43 - 0.00504 -    0.67 - 0.00 - 0.33 - 0.00106 -    1.00 - 0.00 - 0.50 - 0.00315 -
Media    0.84 - 0.00 - 0.42 - 0.00325 -    0.78 - 0.00 - 0.39 - 0.00104 -    0.89 - 0.00 - 0.44 - 0.00319 -
Std      0.05 - 0.00 - 0.02 - 0.00089 -    0.10 - 0.00 - 0.05 - 0.00001 -    0.11 - 0.00 - 0.05 - 0.00005 -

Results for Greedy Relief
Partition        Ionosphere                        Parkinsons                        Spectf-heart
        T_clas - T_red - Agr - T          T_clas - T_red - Agr - T          T_clas - T_red - Agr - T
0        0.86 - 0.03 - 0.44 - 0.01326 -    0.72 - 0.05 - 0.38 - 0.00844 -    0.97 - 0.16 - 0.57 - 0.01327 -
1        0.79 - 0.03 - 0.41 - 0.01289 -    0.82 - 0.00 - 0.41 - 0.00489 -    0.91 - 0.00 - 0.46 - 0.01364 -
2        0.80 - 0.09 - 0.44 - 0.01444 -    0.95 - 0.05 - 0.50 - 0.00490 -    0.79 - 0.00 - 0.39 - 0.01332 -
3        0.91 - 0.03 - 0.47 - 0.01307 -    0.69 - 0.05 - 0.37 - 0.00495 -    0.77 - 0.00 - 0.39 - 0.01336 -
4        0.86 - 0.03 - 0.44 - 0.01562 -    0.67 - 0.00 - 0.33 - 0.00493 -    0.97 - 0.18 - 0.58 - 0.01304 -
Media    0.84 - 0.04 - 0.44 - 0.01386 -    0.77 - 0.03 - 0.40 - 0.00562 -    0.88 - 0.07 - 0.48 - 0.01332 -
Std      0.05 - 0.02 - 0.02 - 0.00103 -    0.10 - 0.02 - 0.06 - 0.00141 -    0.09 - 0.08 - 0.08 - 0.00019 -

Results for Local Search
Partition        Ionosphere                        Parkinsons                        Spectf-heart
        T_clas - T_red - Agr - T          T_clas - T_red - Agr - T          T_clas - T_red - Agr - T
0        0.89 - 0.79 - 0.84 - 37.02041 -   0.72 - 1.00 - 0.86 - 9.79968 -    0.99 - 0.70 - 0.85 - 33.86703 -
1        0.83 - 0.76 - 0.80 - 38.89940 -   0.82 - 0.91 - 0.86 - 8.07744 -    0.93 - 0.80 - 0.86 - 74.65958 -
2        0.83 - 0.82 - 0.83 - 46.61394 -   0.90 - 0.77 - 0.84 - 4.14633 -    0.69 - 0.98 - 0.83 - 138.09329 -
3        0.89 - 0.97 - 0.93 - 65.29625 -   0.79 - 0.95 - 0.87 - 6.80805 -    0.89 - 0.84 - 0.86 - 36.63272 -
4        0.87 - 0.79 - 0.83 - 31.15989 -   0.69 - 0.82 - 0.76 - 9.42441 -    0.99 - 0.93 - 0.96 - 78.72949 -
Media    0.86 - 0.83 - 0.84 - 43.79798 -   0.78 - 0.89 - 0.84 - 7.65118 -    0.89 - 0.85 - 0.87 - 72.39642 -
Std      0.03 - 0.07 - 0.04 - 11.83011 -   0.07 - 0.08 - 0.04 - 2.04631 -    0.11 - 0.10 - 0.04 - 37.74881 -
(base) javier@javier-Lenovo-ideapad-510-15IKB:~/Escritorio/UGR/DGIIM/4º/Cuatri II/MH/P1_definitiva/Sin_sklearn$
```

Figura 1: Resultados Ejecución main.py

5.3 Análisis General. Explicación Resultados Obtenidos

En primer lugar, comenzaremos con el algoritmo 1-NN. Podemos observar que obtiene tasas altas de clasificación, según el dataset en cuestión. Sin embargo, puesto que la tasa de reducción es siempre cero, no obtiene valores altos para la función objetivo. Esto último sobre la tasa de reducción es razonable, pues este algoritmo tiene en cuenta todos los datos para la clasificación.

Pasamos ahora a comentar los resultados obtenidos con el algoritmo Greedy Relief. Este algoritmo es posible que no proporcione a veces la solución más óptima al tratarse de un algoritmo greedy. Tenemos tasas de reducción que ya no se anulan. Esto favorecerá la simplicidad del clasificador, y por consecuente, la posibilidad de obtener valores algo mayores para la función objetivo (aunque no sean nulos, siguen siendo extremadamente bajos). En cuanto a la tasa de clasificación vemos que es bastante buena.

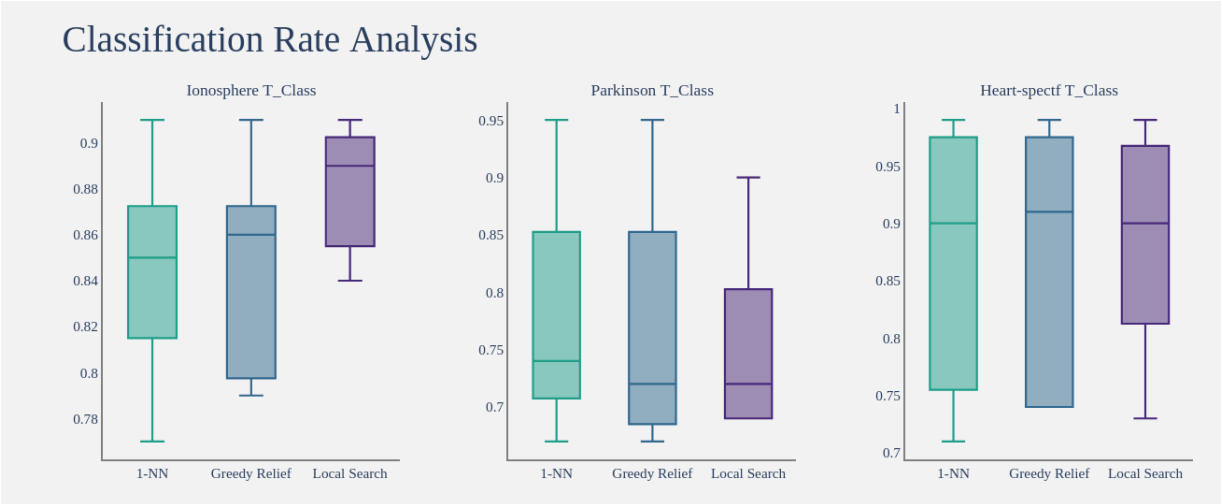
Por todo lo anterior expuesto para el algoritmo Greedy Relief, es razonable que el valor de la función objetivo que obtiene este algoritmo sea mejor que la obtenida por el clasificador sin pesos. En efecto, la teoría se cumple en la práctica. Con todo, la diferencia no es muy significante ya que las tasas de reducción siguen siendo extremadamente bajas.

Ahora, tratamos el algoritmo de Búsqueda Local. A diferencia de los algoritmos anteriores, este consigue tasas de reducción muy altas. Esto, combinado con las buenas tasas de clasificación justifica los altos valores obtenidos para la función objetivo con este algoritmo. Esto, nos hace ver que nuestros vectores de pesos son bastante buenos a la hora de clasificar datos en los datasets que se nos presentan. Con todo, cabe comentar que al tratarse de un algoritmo de búsqueda local usualmente no conseguiremos la solución óptima. Esto es debido a que el algoritmo, aunque pueda tener periodos de estancamiento, suele tender hacia óptimos locales.

Por último, podemos observar como aunque el algoritmo de búsqueda local nos proporcione un clasificador mucho más sencillo, éste es el que más tiempo de ejecución presenta (llegando a tardar en ocasiones 100 veces más por cada partición que el resto). Esto puede ser debido a la suma de la probable ineficiencia de programación del algoritmo y la ineficiencia que nos aporta el lenguaje de programación escogido para programar estos algoritmos.

5.4 Análisis Datasets

Antes de empezar, comentar brevemente que las siguientes gráficas que aparecen han sido creadas mediante el uso de las librerías de **Pandas** y **Plotly**. En caso de querer consultar el código correspondiente, consultar **metaheuristicas.ipynb**. Dicho esto, comenzaremos analizando los resultados que hemos obtenido para la **Tasa de Clasificación**.



5.4.1 Tasa Clasificación

- **Ionosphere**: Podemos observar como los mejores resultados los ofrece el algoritmo de búsqueda local. Concluimos esto porque, es el que presenta un valor mayor para cada uno de los cuartiles.

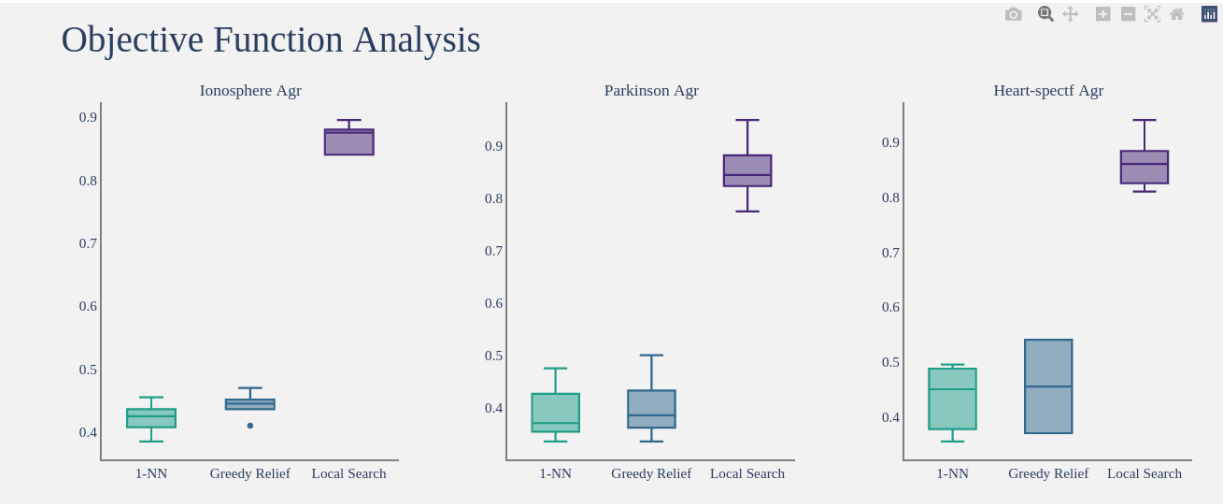
Nos centramos ahora en los algoritmos de 1-NN y Greedy Relief:

- Ambos presentan un valor para la mediana similar.
- El valor del cuartil 1 es superior para el algoritmo de 1-NN
- Más similitud entre los valores obtenidos en los distintos KFold del Cross Validation para el algoritmo de Greedy Relief.

- **Parkinson**: para este dataset podemos observar como las tasas de clasificación de los distintos algoritmos se asemejan mucho más que en el caso anterior. Llama la atención, que sea el algoritmo de 1-NN el que obtenga un valor de mediana más alta. Por último, destacar que el rango de valores para el algoritmo de Búsqueda Local es más reducido que para el resto, razonable pues como podemos apreciar en la Figura 1 (resultados ejecución) la desviación de los resultados es más pequeña para este algoritmo.
- **Spectf Heart**: para este último dataset, es el algoritmo de Greedy Relief el que presentan un mejor valor de mediana. Así mismo, vemos como el algoritmo 1-NN presenta una mayor desviacion en cuanto a los resultados obtenidos en cada KFold.

5.4.2 Función Objetivo

Como bien sabemos, la función objetivo tiene en cuenta tanto la tasa de clasificación como la de reducción. Si nos paramos a observar detenidamente los valores obtenidos en las ejecuciones (previamente mostradas) vemos como la tasa de reducción es significantemente baja para los algoritmos de 1-NN y Greedy Relief. Esto conduce a que, tal y como se puede observar en la gráfica posterior, los valores obtenidos para la función objetivo son mucho mayores en el caso de Local Search. Por su parte, para los otros dos algoritmos restantes, presenta un valor similar.



5.4.3 Tiempo de Ejecución

En esta última sección, haremos un breve análisis del tiempo de ejecución en función del dataset correspondiente.

Como podemos apreciar, el dataset que presenta unos menores tiempos de ejecución en todas las particiones y algoritmos es el de **Parkinson**. Esto puede tener su explicación en que dicho dataset es el de menor tamaño de los tres (solo 195 ejemplos). Así mismo, presenta un menor número de atributos que el resto.

En cuanto a los otros dos, para los algoritmos de 1-NN y Greedy Relief es Ionosphere el dataset con el que más se tarda. Sin embargo, para el algoritmo de Búsqueda Local es el de Spectf Heart, llegando en ocasiones a ser ser significante la diferencia de tiempo con respecto a los demás conjuntos de datos.

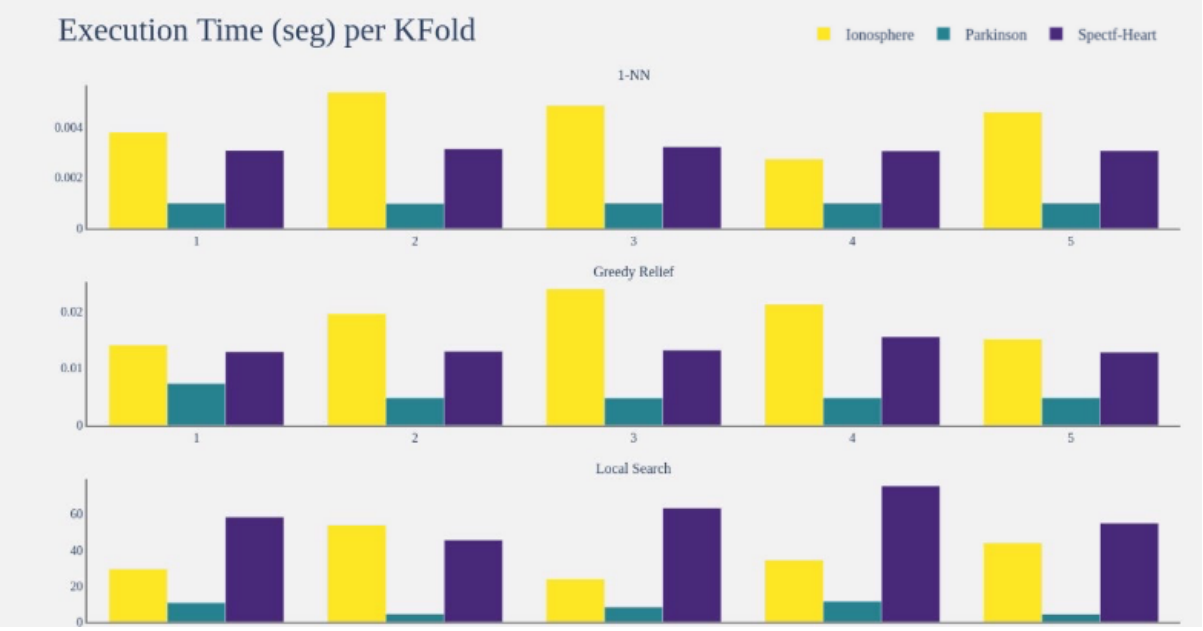


Figura 2: Análisis de los tiempos de ejecución de cada algoritmo, por cada KFold en los datasets

5.4.4 Comparativa Tabla Medias

Como última comparativa, podemos mostrar la tabla de medias :

Ionosphere				Parkinson				Spectf-Heart			
T_clas	T_red	Agr	Tiempo	T_clas	T_red	Agr	Tiempo	T_clas	T_red	Agr	Tiempo
0.84	0	0.42	0.00089	0.78	0	0.39	0.00104	0.89	0	0.44	0.00319
0.84	0.04	0.44	0.01386	0.77	0.03	0.4	0.00562	0.88	0.07	0.48	0.00019
0.86	0.83	0.84	43.79798	0.78	0.89	0.84	7.65118	0.89	0.85	0.87	72.39642

En esta tabla podemos ver claramente las diferencias de tiempo que hay y la gran brecha que crea la **búsqueda local**(la última fila) respecto a Greedy(2ª fila) y 1-nn(1ª fila). Además, se muestra también cómo aumentan la tasa de reducción y el valor de la función objetivo cuando aumenta la complejidad de nuestro algoritmo, lo que lo hace en el caso general mucho más efectivo aunque el coste computacional y de tiempo sea mucho mayor.

5.4.5 Conclusión sobre Datasets

Podemos obtener las siguientes conclusiones a cerca de cada uno de los conjuntos de datos:

- *Ionosphere, Spectf-Heart* nos da un porcentaje de clasificación alto en todos los algoritmos, a pesar de ser

los que más tipos de atributos diferentes tiene, y más ejemplos. Probablemente esto sea porque las clases tienen características bien diferenciadas.

- *Parkinson* es un conjunto de datos más pequeño, pero algo más complicado de clasificar comparado con los otros dos. Podría ser por tener menos ejemplos que los otros dos conjuntos o quizá porque los datos estén más dispersos y los ejemplos no representen bien a los representantes de cada clase.