Práctica 1 Búsqueda Local APC
Metaheurísticas
Algoritmos KNN, Relief y Búsqueda Local

Ignacio Aguilera Martos
DNI: 77448262V e-mail: nacheteam@correo.ugr.es
Grupo de prácticas 1 Lunes 17:30-19:30

Curso 2017-2018

Índice

1. Introducción del problema

Para el problema de clasificación partimos de un conjunto de datos dado por una serie de tuplas que contienen los valores de atributos para cada instancia. Esto es una n-tupla de valores reales en nuestro caso.
El objetivo del problema es obtener un vector de pesos que asocia un valor en el intervalo $[0, 1]$ indicativo de la relevancia de ese atributo. Esta relevancia va referida a lo importante que es en nuestro algoritmo clasificador ese atributo a la hora de computar la distancia entre elementos.
Resumiendo lo que tenemos es un algoritmo clasificador que utiliza el vector de pesos calculado para predecir la clase a la que pertenece una instancia dada. Este algoritmo clasificador es el KNN con k=1. Lo que hace es calcular según la distancia euclídea (o cualquier otra) la tupla más cercana a la que queremos clasificar ponderando cada atributo con el correspondiente peso del vector, es decir, la distancia entre dos elementos sería:

$$d(e, f) = \sqrt{\sum_{i=0}^{n} w_i * (e_i - f_i)}$$

Donde e y f son instancias del conjunto de datos, w el vector de pesos y n la longitud de e y f que es la misma.
La calificación que se le asigna al vector w depende de dos cosas: la tasa de aciertos y la simplicidad.
La tasa de aciertos se mide contando el número de aciertos al emplear el clasificador descrito y la simplicidad se mide como el número de elementos del vector de pesos que son menores que 0.2, ya que estos pesos no son empleados por el clasificador, o lo que es lo mismo, son sustituidos por cero. Por lo tanto las calificaciones siguen las fórmulas:

$$Tasa\_acierto = 100 \cdot \frac{n^o\ aciertos}{n^o\ datos}\ ,\ Tasa\_simplicidad = 100 \cdot \frac{n^o\ valores\ de\ w\ <\ 0{,}2}{n^o\ de\ atributos}$$

$$Tasa\_agregada = \frac{1}{2} \cdot Tasa\_acierto + \frac{1}{2} \cdot Tasa\_simplicidad$$

Cabe destacar que todas las tasas están expresadas en porcentajes, por lo tanto cuanto más cercano sea el valor a 100 mejor es la calificación.
De esta forma a través del algoritmo que obtiene el vector de pesos para el conjunto de datos dado y el clasificador obtenemos un programa que clasifica de forma automática las nuevas instancias de datos que se introduzcan.

2. Introducción de la práctica

En esta práctica he analizado el comportamiento de los algoritmos KNN con k=1, el algoritmo greedy Relief y una implementación del algoritmo de búsqueda local para el problema de obtención de un vector de pesos para clasificar un conjunto de datos. Así mismo he realizado la implementación del algoritmo KNN con k variable para poder estudiar si varían los resultados al aumentar el valor de K o por contra obtenemos demasiado ajuste.
Para empezar, al leer los ficheros de datos dados para el problema, me he dado cuenta de que tenemos tuplas repetidas, cosa que he tenido en cuenta para no usarlas en la clasificación, ya que siempre obtendríamos distancia 0 para dicha tupla. Para ello en vez de comprobar el índice dentro del vector he comprobado si las tuplas son iguales para no usarlas.
Para terminar, antes de analizar los datos, se debe considerar que los datos han sido redondeados a 4 decimales para no obtener tablas excesivamente largas. Si se desea obtener los datos completos se puede ejecutar el programa como se describe en la sección 7.

3. Descripción común a todos los algoritmos

Los algoritmos empleados han sido el KNN, el algoritmo greedy Relief y la metaheurística de búsqueda local.
Estos algoritmos comparten ciertos métodos y operadores que pasaré a explicar en esta sección.
Para empezar se debe destacar que la representación escogida para las soluciones es un vector de números reales, es decir, si n es el número de características:

$$w \in \mathbb{R}^n \ t.q. \ \forall i \ con \ 0 \leq i < n \ se \ tiene \ w_i \in [0,1]$$

O lo que es lo mismo, un vector de tamaño n con todas las posiciones rellenas con números del intervalo [0,1].
A estos números me referiré como pesos asociados a las características, ya que lo que nos indican es el grado de importancia de dicha característica a la hora de clasificar los datos, siendo 1 el máximo de relevancia y 0 el mínimo.
Así mismo cabe destacar que nuestra intención en este problema es obtener una buena calificación de dicho vector de pesos. Esto lo medimos mediante las tasas de acierto y simplicidad que se definen como:

$$Tasa\_acierto = 100 \cdot \frac{n^o \ aciertos}{n^o \ datos} \ , \ Tasa\_simplicidad = 100 \cdot \frac{n^o \ valores \ de \ w \ < \ 0{,}2}{n^o \ de \ atributos}$$

$$Tasa\_agregada = \frac{1}{2} \cdot Tasa\_acierto + \frac{1}{2} \cdot Tasa\_simplicidad$$

La tasa de aciertos lo que nos mide es en un porcentaje cuántas instancias hemos clasificado correctamente mediante el algoritmo KNN usando el vector de pesos w.
La tasa de simplicidad nos mide cuántos de los valores que tiene el vector de pesos son menores que 0.2. Esto se hace ya que, como imposición del problema, tenemos que si alguno de los pesos es menor que 0.2 no debemos usarlo, o lo que es lo mismo, debemos sustituirlo por un 0 en la función de la distancia que luego describiré. Midiendo esto obtenemos un dato de cuanto sobreajuste ha tenido nuestro algoritmo a la hora de obtener el vector de pesos. Cuantas menos características necesitemos para discernir la clase a la que pertenece una instancia de los datos, más simple será clasificar dicha instancia. Se expresa en porcentaje indicando 0 como ninguna simplicidad y 100 como la máxima simplicidad.
De esta forma combinando ambas tasas obtenemos la tasa agregada que nos hace la media entre ambas tasas, de forma que le asignamos la misma importancia a acertar en la clasificación de

las instancias y a la simplicidad en la solución. Cabe destacar que es imposible obtener una tasa de un 100 % a no ser que los datos se compongan únicamente de un punto ya que ello implicaría que la simplicidad ha de ser un 100 % (todos las posiciones del vector menores que 0.2) y por tanto la distancia sería 0 en todos los casos. De esta forma aspiraremos a una calificación lo mas alta posible pero teniendo en cuenta las restricciones de la función objetivo construida.
Las funciones y operadores de uso común los he agrupado en un fichero llamado auxiliar.py. Este fichero contiene las funciones de lectura de datos, distancias, una función que devuelve el elemento más común de una lista, la norma euclídea y una función para dividir los datos en el número de particiones que queramos manteniendo el porcentaje de elementos de cada clase que había en el conjunto de datos original.

3.1. Función de lectura de datos

La función de lectura de datos recibe la ruta del fichero arff y lee el contenido del mismo dando como resultado una lista de listas en la que cada una de ellas es una tupla o instancia de los datos.
El pseudocódigo de la función es:

Algorithm 1 lecturaDatos(nombre_fich)

```
data ← [ ]
for linea de nombre_fich do
  if se ha leído @data then
    data ← [data,linea]
  end if
end for
for tupla en data do
  for atributo en tupla do
    Normalizar.
  end for
end for
return data
```

Para esta implementación en concreto nos apoyamos en que Python tiene polimorfismo para todos los tipos de datos sin necesidad de declarar las variables, de forma que no nos importa que los datos sean numéricos o de tipo string.

3.2. Funciones de distancia

Las funciones de distancia siguen todas el mismo esquema de código, cambiando únicamente la fórmula empleada en cada caso para computar la distancia. Voy a describir las 3 distancias que he implementado teniendo esto en cuenta.
Se debe tener en cuenta que e1 y e2 son ambos dos tuplas del conjunto de datos de las que vamos a obtener la distancia y w es el vector de pesos que toma parte en el cómputo.
El resto de funciones de distancia siguen el mismo esquema pero cambiando la fórmula para obtener el valor de la misma.

3.3. Función MasComun

Esta función lo que hace es devolvernos el elemento que más se repite dentro de un vector, esto es utilizado en el algoritmo KNN para obtener la clase más común entre los k elementos con distancia mínima al dado.

Algorithm 2 distanciaEuclidea(e1,e2,w)

if longitud(e1)!=longitud(e2) **then**
  No se puede hallar la distancia.
else
  distancia = $\sqrt{\sum_{i=1}^{longitud(e1)} w_i \cdot (e1_i - e2_i)^2}$
end if

Algorithm 3 masComun(lista)

vector_repeticiones ← [ ]
for elemento en lista **do**
  Contar el número de veces que aparece e introducirlo en el vector de repeticiones.
end for
Obtener el elemento con mayor número de apariciones.

3.4. Función de norma euclídea

Esta función toma una lista de elementos y devuelve la norma euclídea asociada al vector que representa la lista.

Algorithm 4 normaEuclidea(e)

norma ← 0
for ei en e **do**
  norma ← norma+ei^2
end for
norma ← $\sqrt{norma}$
return norma

3.5. Función de división de datos

Esta función divide el conjunto de datos inicial en n particiones todas ellas respetando el porcentaje de ocurrencias de cada clase que tenía el conjunto de datos inicial.

Algorithm 5 divideDatosFCV(data,n)

```
particiones ← [ ]
tam_particion ← longitud(data) / n
clases ← Posibles clases del conjunto data.
proporciones_clases ← [ ]
for clase en clases do
  proporciones_clases ← [proporciones_clases, nueva_proporcion]
end for
for i en 0..n do
  particion ← [ ]
  for j en 0..numero_clases do
    numero_elementos_clase = proporciones_clases[j]*tam_particion
    for k en 0..longitud(data) do
      Introducir en particion el número de elementos de clase calculado para cada clase.
    end for
  end for
end for
Si han sobrado datos sin colocar los ponemos en la última partición.
return particiones
```

4. KNN

El algoritmo KNN ha sido utilizado en estas prácticas tanto como clasificador (y por tanto con el objetivo de obtener una valoración de un vector de pesos) como algoritmo de comparación. El guión de prácticas nos pedía implementar el algoritmo 1NN, pero yo he decidido implementarlo de forma genérica para poder comprobar cómo cambiaban los resultados cuando incrementamos el número de vecinos más cercanos.

Este algoritmo toma como entrada un vector de pesos, un conjunto de datos de entrenamiento con sus etiquetas de clase, otro de test con el conjunto de etiquetas también, un valor k que nos indica cuántos vecinos más cercanos queremos tomar para clasificar cada dato y un booleano que nos indica si los dos conjuntos pasados son iguales (cosa que ocurre por ejemplo en la búsqueda local, genéticos y el memético) con la intención de aplicar el leave-one-out. El procedimiento consiste en tomar cada dato del conjunto de test y hallar los k datos del conjunto de entrenamiento que tienen distancia mínima con el dato tomado del conjunto de test. De ahí lo que hacemos es clasificar el dato con la clase más frecuente de entre los k vecinos más cercanos calculados. De esta forma, intuitivamente, asignamos la misma clase a datos que están próximos entre sí. Una vez obtenida la clase de estos elementos lo que hacemos es comprobar cuántas clases hemos acertado al clasificar de esta forma. Con esto obtenemos tras la ejecución del algoritmo un número entre 0 y 100 que nos indica cual ha sido nuestra tasa de acierto (100 todo correcto, 0 ninguna correcta).

Para emplear este algoritmo como comparación para los otros lo que hacemos es introducir como vector de pesos un vector con todas las posiciones a 1. De esta forma obtenemos la calificación que obtendríamos si no hubiéramos incluido en la fórmula de la distancia una ponderación con pesos.

Nótese que para esta implementación el vector de pesos pasado tiene, en las posiciones con valor inicial menor que 0.29, ceros. Esto se hace con la intención de mejorar la eficiencia del algoritmo.

El pseudocódigo del algoritmo es:

Cabe notar que el número que devolvemos está entre 0 y 1, por lo que en los algoritmos

Algorithm 6 KNN(w,datos_test,datos_entrenamiento, etiquetas_entrenamiento, etiquetas_test, k, mismos_conjuntos)

```
tam_datos_entrenamiento ← longitud(datos_entrenamiento)
clases ← []
for i=0,...,longitud(datos_test) do
  p ← datos_test[i]
  w_m ← Repetir el vector w tantas veces como datos haya en datos_entrenamiento.
  p_m ← Repetir el vector p tantas veces como datos haya en datos_entrenamiento.
  dist ← w_m · (p_m − datos_entrenamiento)^2
  if mismos_conjuntos then
    dist[i] ← ∞
  end if
  mins ← Los k índices correspondientes a las distancias más pequeñas.
  clases ← [clases, masComun(etiquetas_entrenamiento[mins])]
end for
return Numero de elementos de clases que han acertado con respecto a etiquetas_test / longitud(etiquetas_test)
```

de valoración debemos tener esto en cuenta para multiplicarlo por 100 y convertirlo en un porcentaje.

5. Relief

El algoritmo Relief es un algoritmo greedy que busca rapidez en detrimento de mejores resultados de clasificación.
El algoritmo toma como entrada únicamente el conjunto de datos. Para cada elemento del conjunto buscamos el elemento de la misma clase más cercano y el elemento de la clase contraria más cercano de forma que tenemos al amigo y enemigos más cercanos. En la literatura es común encontrar estos elementos como near hit y near miss.
Partiendo de un vector de pesos con todas las posiciones a 0 tomamos el amigo y enemigo más cercano y restamos a nuestro elemento del conjunto de datos tanto el amigo como el enemigo. Después de esto lo que hacemos es sumar la resta del enemigo y restar la resta del amigo, de forma que aumentamos la distancia que dicho vector de pesos produce para hacer que el amigo esté más cercano del dato que el enemigo.
Esto ocurre ya que sumamos la distancia del elemento al enemigo haciendo que los pesos sean mayores para el mismo y restamos la distancia del elemento al amigo haciendo que nos acerque en distancia a dicha tupla.
A continuación se describe el funcionamiento en pseudocódigo:
Tenemos en primer lugar la función elementoMinimaDistancia que nos devuelve el elemento de la lista más cercano al elemento e. De esta forma podemos hallar el amigo y enemigo más cercano para el algoritmo Relief. Cabe destacar que para que el vector de pesos esté con números entre 0 y 1 tenemos que poner al final del algoritmo los elementos del vector negativos a 0 y el resto los normalizamos dividiendo por el máximo elemento del vector de forma que todas las posiciones nos queden entre 0 y 1.

Algorithm 7 elementoMinimaDistancia(e,lista)

```
distancias ← [ ]
for l en lista do
  if l!=e then
    distancias ← [distancias, distancia(e,l,[1..1])]
  else
    distancias ← [distancias, max(distancias)]
  end if
end for
indice_menor_distancia ← índice del elemento de menor valor del vector distancias.
return lista[indice_menor_distancia]
```

Algorithm 8 Relief(data)

```
w ← vector de pesos a 0
for elemento en data do
  clase ← clase de elemento
  amigos ← [ ]
  enemigos ← [ ]
  for e en data do
    if e!=elemento AND e[longitud(e)-1]==clase then
      amigos ← [amigos, e]
    else
      enemigos ← [enemigos, e]
    end if
  end for
  amigo_cercano ← elementoMinimaDistancia(elemento, amigos)
  enemigo_cercano ← elementoMinimaDistancia(elemento, enemigos)
  resta_enemigo ← element-enemigo_cercano
  resta_amigo ← element-amigo_cercano
  w ← w + resta_enemigo - resta_amigo
  w_max ← máximo de w
end for
for i en [0..longitud(w)-1] do
  if w[i]<0 then
    w[i] ← 0
  else
    w[i] ← w[i] / w_max
  end if
end for
return w
```

6. Búsqueda Local

El algoritmo de búsqueda local tiene como intención, al igual que el anterior, proporcionarnos un vector de pesos que maximice la tasa agregada que obtenemos como ya he mencionado anteriormente.
Este algoritmo comienza con un vector de pesos aleatorio generado con una distribución uniforme en el intervalo [0,1], obteniendo a partir de ahí mejores vectores a cada iteración. La generación de vecinos viene dada por un operador que toma posiciones aleatorias del vector de pesos (sin repetición) y les suma un valor aleatorio generado con una distribución normal de media 0 y desviación 0.3. Tras esto hacemos igual que en el algoritmo Relief para normalizar el vector de pesos y que cumpla las restricciones del problema.
Dado el vector de pesos inicial obtenemos su valoración de la tasa agregada (incluye la simplicidad y la tasa de aciertos). Generamos un vecino mediante la forma mencionada anteriormente y comprobamos si su tasa es mayor que la del vector de pesos actual. Si esto es así descartamos nuestro vector de pesos y tomamos al vecino, en caso contrario continuaremos explorando el vecindario hasta encontrar uno que mejore el resultado, agotar las posibilidades de vecinos o llegar a 15.000 evaluaciones de la función objetivo o $20 \cdot n$ vecinos explorados donde n es el número de atributos de cada tupla de datos.
De esta manera otorgamos una forma de parar a nuestro algoritmo y no excederse en la búsqueda.
Las funciones que intervienen, en pseudocódigo, son las siguientes:

Algorithm 9 mutacion(w,vector_posiciones)

```
incremento ← random.gauss(0,0.3)
i ← random.int(0,longitud(vector_posiciones)-1)
vector_posiciones_aux ← [ ]
pos ← vector_posiciones[i]
w[pos] ← w[pos]+incremento
w_max ← maximo(w)
for i en [0..longitud(w)-1] do
  if w[i]¡0 then
    w[i] ← 0
  else
    w[i] ← w[i] / w_max
  end if
end for
for v en vector_posiciones do
  if v!=pos then
    vector_posiciones_aux ← [vector_posiciones_aux, v]
  end if
end for
return w,vector_posiciones_aux
```

Algorithm 10 primerVector(n)

```
w ← [ ]
for i en [0..n-1] do
  w ← [w, random.uniforme(0,1)]
end for
return w
```

Algorithm 11 busquedaLocal(data,k)

```
MAX_EVALUACIONES ← 15000
MAX_VECINOS ← 20 · longitud(data[0])
vecinos ← 0
evaluaciones ← 0
w ← primerVector(longitud(data[0]))
vector_posiciones ← [0..longitud(w)-1]
valoracion_actual ← Valoracion(data,data,k,w)
while evaluaciones¡MAX_EVALUACIONES AND vecinos¡MAX_VECINOS do
  evaluaciones ← evaluaciones+1
  vecinos ← vecinos+1
  vecino, vector_posiciones ← mutacion(w,vector_posiciones)
  valoracion_vecino ← Valoracion(data,data,k,vecino)
  if valoracion_vecino¿valoracion_actual then
    w ← vecino
    valoracion_actual ← valoracion_vecino
    vector_posiciones ← [0..longitud(w)-1]
  else if vector_posiciones==[ ] then
    return w
  end if
end while
return w
```

Debe tomarse en cuenta que la función Valoracion(data,data,k,w) devuelve la tasa agregada teniendo en cuenta la simplicidad y la tasa de acierto como se ha descrito en el algoritmo KNN. Este algoritmo, como ya discutiré en la parte de resultados, puede quedarse atrapado fácilmente en un máximo local y va a consumir mucho más tiempo, ya que cada llamada de la función valoración ejecuta el algoritmo KNN sobre dicho conjunto de datos sumando así mayor tiempo de ejecución que el de sus competidores.

7. Procedimiento de desarrollo de la práctica

El desarrollo de esta práctica lo he realizado en Python 3.5 implementando desde cero todos los métodos necesarios por mi cuenta. Para implementar los tres algoritmos principales de la práctica me he valido de las explicaciones del seminario 2, completando la información para entender el algoritmo (si era necesario) mediante internet. En cualquier caso los algoritmos implementados reflejan los esquemas del seminario 2 y no otros.
Los archivos proporcionados de código son auxiliar.py donde se implementan los algoritmos y funciones comunes que se utilizan en todos o varios algoritmos con la intención de reutilizar las mismas. Así mismo el algoritmo KNN tiene su propio fichero donde está implementado y de igual forma lo tienen el algoritmo Relief y de búsqueda local. En cada uno de estos ficheros se incluye el algoritmo descrito en las secciones previas así como una función de valoración que se encarga de ejecutar el algoritmo para obtener el vector de pesos con 5 particiones y obtener las calificaciones obtenidas para cada una de las 5 particiones. De esta forma es más sencillo diseñar un script como el que se encuentra en el archivo resultados.py que toma dichas funciones y ejecuta para k=1,3,5 los algoritmos sobre los conjuntos de prueba. De esta forma obtenemos de manera automatizada todos los resultados que vemos en las tablas de la sección siguiente. Cabe destacar el hecho de que empleo siempre valores impares de k. Esta estrategia se sigue de que no queremos que se produzcan empates, es decir, si tomáramos k=4 podría darse el caso de obtener 2 elementos de una clase y otros 2 de la contraria teniendo que decidir finalmente de forma aleatoria o por orden. Es preferible por tanto tomar valores impares para evitar situaciones de empate.
Si se desea ejecutar el programa se recomienda compilar el código para que su ejecución sea mucho más rapida con el comando 'python3.5 -m compileall .'. Tras esto para ejecutar el script de resultados sólo tenemos que hacerlo como 'python3.5 resultados.py'.

8. Resultados

	Ozone				Parkinsons				Spectf-Heart			
	%_clas	%_red	Agr.	T (seg)	%_clas	%_red	Agr.	T (seg)	%_clas	%_red	Agr.	T (seg)
Partición 1	71.8750	0.0000	35.9375	0.4403	76.3158	0.0000	38.1579	0.0546	70.5882	0.0000	35.2941	0.2561
Partición 2	84.3750	0.0000	42.1875	0.4341	81.5789	0.0000	40.7895	0.0527	77.9412	0.0000	38.9706	0.2959
Partición 3	71.8750	0.0000	35.9375	0.4329	94.7368	0.0000	47.3684	0.0543	67.6471	0.0000	33.8235	0.3161
Partición 4	81.2500	0.0000	40.6250	0.4340	73.6842	0.0000	36.8421	0.0530	60.2941	0.0000	30.1471	0.2984
Partición 5	85.9375	0.0000	42.9688	0.4484	76.7442	0.0000	38.3721	0.0586	66.2338	0.0000	33.1169	0.2715
Media	79.0625	0.0000	39.5313	0.4380	80.6120	0.0000	40.3060	0.0546	68.5409	0.0000	34.2704	0.2876

Cuadro 1: Resultados 1NN

	Ozone				Parkinsons				Spectf-Heart			
	%_clas	%_red	Agr.	T (seg)	%_clas	%_red	Agr.	T (seg)	%_clas	%_red	Agr.	T (seg)
Partición 1	73.4375	0.0000	36.7188	0.4694	78.9474	0.0000	39.4737	0.0551	70.5882	0.0000	35.2941	0.2815
Partición 2	85.9375	0.0000	42.9688	0.4559	76.3158	0.0000	38.1579	0.0555	75.0000	0.0000	37.5000	0.3794
Partición 3	78.1250	0.0000	39.0625	0.4344	94.7368	0.0000	47.3684	0.0553	64.7059	0.0000	32.3529	0.3513
Partición 4	82.8125	0.0000	41.4063	0.4433	71.0526	0.0000	35.5263	0.0557	70.5882	0.0000	35.2941	0.3800
Partición 5	87.5000	0.0000	43.7500	0.4433	72.0930	0.0000	36.0565	0.0610	64.9351	0.0000	32.4975	0.3168
Media	81.5625	0.0000	40.7813	0.4493	78.6291	0.0000	39.3146	0.0565	69.1635	0.0000	34.5817	0.3418

Cuadro 2: Resultados 3NN

	Ozone				Parkinsons				Spectf-Heart			
	%_clas	%_red	Agr.	T (seg)	%_clas	%_red	Agr.	T (seg)	%_clas	%_red	Agr.	T (seg)
Partición 1	73.4375	0.0000	36.7188	0.5142	76.3158	0.0000	38.1579	0.0679	72.0588	0.0000	36.0294	0.3407
Partición 2	90.6250	0.0000	45.3125	0.5176	81.5789	0.0000	40.7895	0.0587	72.0588	0.0000	36.0294	0.4101
Partición 3	75.0000	0.0000	37.5000	0.5100	89.4737	0.0000	44.7368	0.0615	67.6471	0.0000	33.8235	0.3748
Partición 4	81.2500	0.0000	40.6250	0.5261	76.3158	0.0000	38.1579	0.0592	69.1176	0.0000	34.5588	0.3774
Partición 5	84.3750	0.0000	42.1875	0.4988	72.0930	0.0000	36.0465	0.0648	68.8312	0.00000	34.4156	0.3210
Media	80.9375	0.0000	40.4688	0.5133	79.1554	0.0000	39.5777	0.0624	69.9427	0.0000	34.9714	0.3648

Cuadro 3: Resultados 5NN

	Ozone				Parkinsons				Spectf-Heart			
	%_clas	%_red	Agr.	T (seg)	%_clas	%_red	Agr.	T (seg)	%_clas	%_red	Agr.	T (seg)
Partición 1	71.8750	98.6301	85.2526	0.9703	78.9474	47.8261	63.3867	0.1254	75.0000	46.6667	60.8333	0.4568
Partición 2	87.5000	97.2603	92.3801	1.0143	81.5789	26.0869	53.8330	0.1267	73.5294	53.3333	63.4314	0.6407
Partición 3	71.8750	97.2603	84.5676	1.0385	86.8421	47.8261	67.3341	0.1263	67.6471	60.0000	63.8235	0.7070
Partición 4	81.2500	95.8904	88.5702	1.0530	78.9474	52.1739	65.5606	0.1254	72.0588	31.1111	51.5850	0.6594
Partición 5	85.9375	98.6301	92.2838	0.9963	72.0930	43.4783	57.7856	0.1180	72.7273	48.8889	60.8081	0.3986
Media	79.6875	97.5342	88.6109	1.0145	79.6818	43.4783	61.5800	0.1244	72.1925	48.0000	60.0963	0.5725

Cuadro 4: Resultados Relief con K=1

	Ozone				Parkinsons				Spectf-Heart			
	%_clas	%_red	Agr.	T (seg)	%_clas	%_red	Agr.	T (seg)	%_clas	%_red	Agr.	T (seg)
Partición 1	73.4375	98.6301	86.0338	0.9413	78.9474	47.8261	63.3867	0.1317	66.1765	46.6667	56.4216	0.5166
Partición 2	85.9375	97.2603	91.5989	0.9480	78.9474	26.0870	52.5172	0.1304	80.8824	53.3333	67.1078	0.7211
Partición 3	78.1250	97.2603	87.6926	0.9244	94.7368	47.8261	71.2815	0.1322	58.8235	60.0000	59.4118	0.8060
Partición 4	81.2500	95.8904	88.5702	0.9436	68.4211	52.1739	60.2975	0.1321	70.5882	31.1111	50.8497	0.6745
Partición 5	87.5000	98.6301	93.0651	0.9589	72.0930	43.4783	57.7856	0.1242	71.4276	48.8889	60.1587	0.4445
Media	81.2500	97.5342	89.3921	0.9432	78.6291	43.4783	61.0537	0.1301	69.5798	48.0000	58.7899	0.6325

Cuadro 5: Resultados Relief con K=3

	Ozone				Parkinsons				Spectf-Heart			
	%_clas	%_red	Agr.	T (seg)	%_clas	%_red	Agr.	T (seg)	%_clas	%_red	Agr.	T (seg)
Partición 1	73.4375	98.6301	86.0338	1.0732	84.2105	47.8261	66.0183	0.1356	66.1765	46.6667	56.4216	0.5553
Partición 2	89.0625	97.2603	93.1614	1.0930	78.9474	26.0870	52.5172	0.1364	77.9412	53.3333	65.6373	0.7449
Partición 3	75.0000	97.2603	86.1301	1.0775	100.0000	47.8261	73.9130	0.1351	69.1176	60.0000	64.5588	0.8443
Partición 4	81.2500	95.8904	88.5705	1.0991	76.3158	52.1739	64.2449	0.1383	69.1176	31.1111	50.1144	0.8855
Partición 5	84.3750	98.6301	91.5026	1.0808	76.7442	43.4783	60.1112	0.1294	71.4286	48.8889	60.1587	0.5262
Media	80.6250	97.5342	89.0796	1.0847	83.2436	43.4783	63.3609	0.1350	70.7563	48.0000	59.3782	0.7112

Cuadro 6: Resultados Relief con K=5

	Ozone				Parkinsons				Spectf-Heart			
	%_clas	%_red	Agr.	T (seg)	%_clas	%_red	Agr.	T (seg)	%_clas	%_red	Agr.	T (seg)
Partición 1	71.8750	34.2466	53.0608	186.6400	78.9474	26.0870	52.5172	9.0305	70.5882	35.5556	53.0719	52.5939
Partición 2	76.5625	54.7945	65.6785	436.2914	81.5789	39.1304	60.3547	6.8897	82.3529	24.4444	53.3987	92.8766
Partición 3	70.3125	30.1370	50.2247	185.3929	86.8421	17.3913	52.1167	4.3802	70.5882	57.7778	64.1830	120.7886
Partición 4	82.8125	54.7945	68.8035	354.7868	78.9474	52.1739	65.5606	21.2622	63.2353	31.1111	47.1732	70.7596
Partición 5	84.3750	39.7260	62.0505	224.3229	74.4186	30.4348	52.4267	9.3806	64.9351	44.4444	54.6898	90.2908
Media	77.1875	42.7397	59.9636	277.4868	80.1469	33.0435	56.5952	10.1886	70.3400	38.6667	54.5033	85.4619

Cuadro 7: Resultados Búsqueda Local con K=1

	Ozone				Parkinsons				Spectf-Heart			
	%_clas	%_red	Agr.	T (seg)	%_clas	%_red	Agr.	T (seg)	%_clas	%_red	Agr.	T (seg)
Partición 1	73.4375	46.5753	60.0064	248.5178	76.3158	34.7826	55.5492	18.8915	66.1765	46.6667	56.4216	124.4612
Partición 2	85.9375	56.1644	71.0509	365.1825	78.9474	56.5217	67.7346	15.0689	75.0000	26.6667	50.8333	74.7928
Partición 3	78.1250	52.0548	65.0899	267.7560	92.1053	26.0870	59.0961	8.4903	69.1176	26.6667	47.8922	69.5649
Partición 4	79.6875	46.5753	63.1314	312.7265	68.4211	34.7826	51.6018	12.1651	70.5882	42.2222	56.4052	94.7792
Partición 5	89.0625	46.5753	67.8189	329.6873	69.7674	17.3913	43.5794	6.3797	64.9351	40.0000	52.4675	41.5187
Media	81.2500	49.5890	65.4195	304.7740	77.1114	33.9130	55.5122	12.1991	69.1635	36.4444	52.8040	81.0234

Cuadro 8: Resultados Búsqueda Local con K=3

	Ozone				Parkinsons				Spectf-Heart			
	%_clas	%_red	Agr.	T (seg)	%_clas	%_red	Agr.	T (seg)	%_clas	%_red	Agr.	T (seg)
Partición 1	71.8750	45.2055	58.5402	201.1896	73.6842	52.1737	62.9291	16.7641	82.3529	48.8889	65.6209	172.0540
Partición 2	89.0625	36.9863	63.0244	274.0396	81.5789	26.0870	53.8330	11.6506	79.4118	57.7778	68.5948	143.7965
Partición 3	75.0000	34.2466	54.6233	192.7606	89.4737	34.7826	62.1281	20.1932	69.1176	57.7778	63.4477	183.0782
Partición 4	84.3750	43.8356	64.1053	363.2454	73.6842	43.4783	58.5812	11.8747	60.2941	53.3333	56.8137	199.8569
Partición 5	89.0625	61.6438	75.3532	593.0892	72.0930	47.8261	59.9596	13.0924	67.5325	60.0000	63.7662	115.5035
Media	81.8750	44.3836	63.1293	324.8649	78.1028	40.8696	59.4862	14.7150	71.7418	55.5556	63.6487	162.8578

Cuadro 9: Resultados Búsqueda Local con K=5

	Ozone				Parkinsons				Spectf-Heart			
	%_clas	%_red	Agr.	T (seg)	%_clas	%_red	Agr.	T (seg)	%_clas	%_red	Agr.	T (seg)
1-NN	79.0625	0.0000	39.5313	0.4380	80.6120	0.0000	40.3060	0.0546	68.5409	0.0000	34.2704	0.2876
Relief	79.6875	97.5342	88.6109	1.0145	79.6818	43.4783	61.5800	0.1244	72.1925	48.0000	60.0963	0.5725
BL	77.1875	42.7397	59.9636	277.4868	80.1469	33.0435	56.5952	10.1886	70.3400	38.6667	54.5033	85.4619

Cuadro 10: Resultados globales con K=1

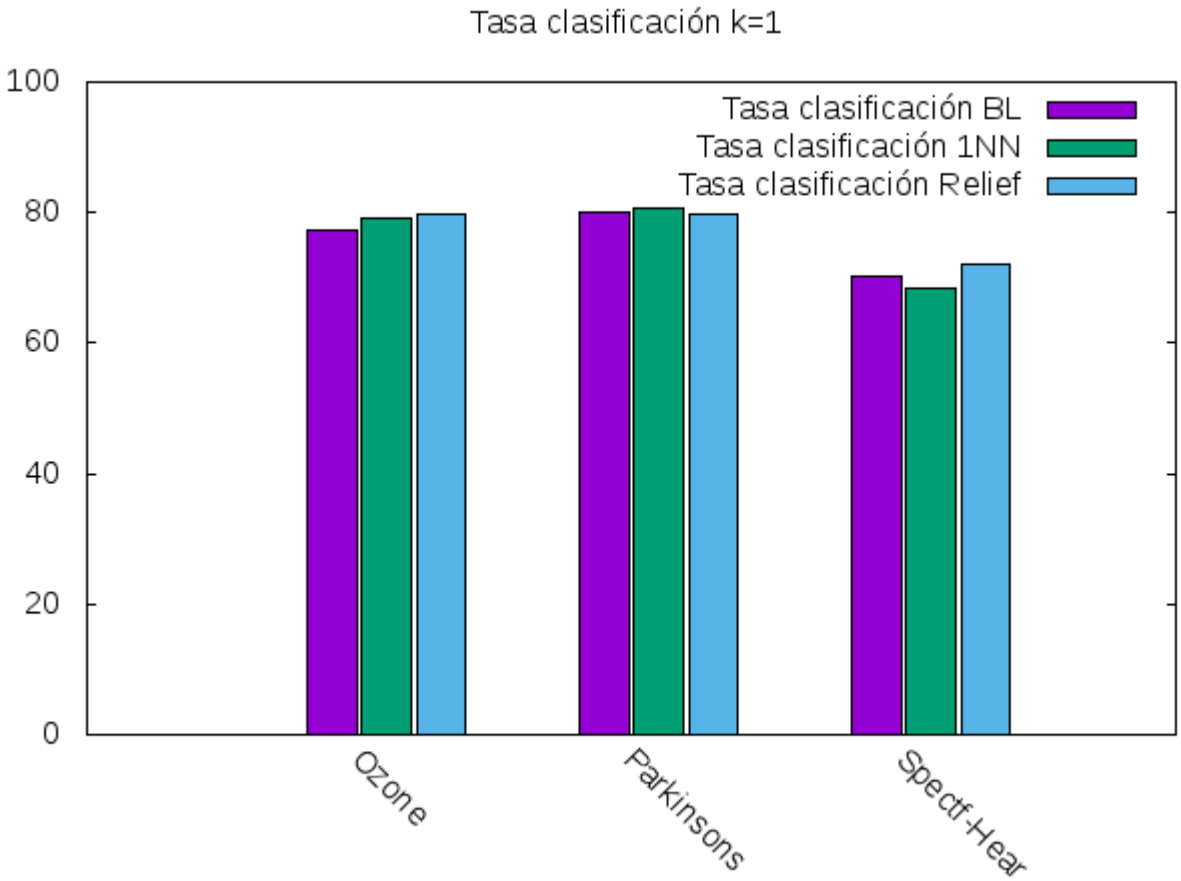
	Ozone				Parkinsons				Spectf-Heart			
	%_clas	%_red	Agr.	T (seg)	%_clas	%_red	Agr.	T (seg)	%_clas	%_red	Agr.	T (seg)
3-NN	81.5625	0.0000	40.7813	0.4493	78.6291	0.0000	39.3146	0.0565	69.1635	0.0000	34.5817	0.3418
Relief	81.2500	97.5342	89.3921	0.9432	78.6291	43.4783	61.0537	0.1301	69.5798	48.0000	58.7899	0.6325
BL	81.2500	49.5890	65.4195	304.7740	77.1114	33.9130	55.5122	12.1991	69.1635	36.4444	52.8040	81.0234

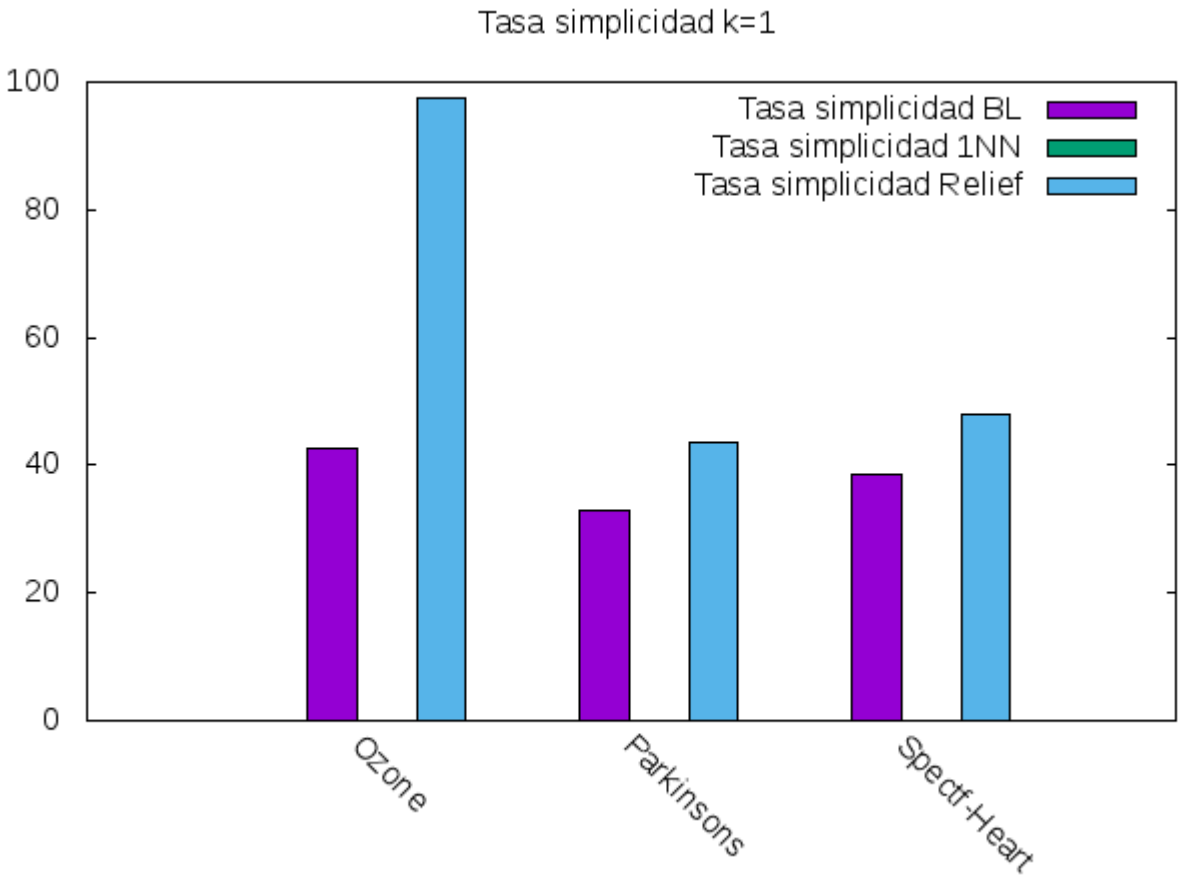
Cuadro 11: Resultados globales con K=3

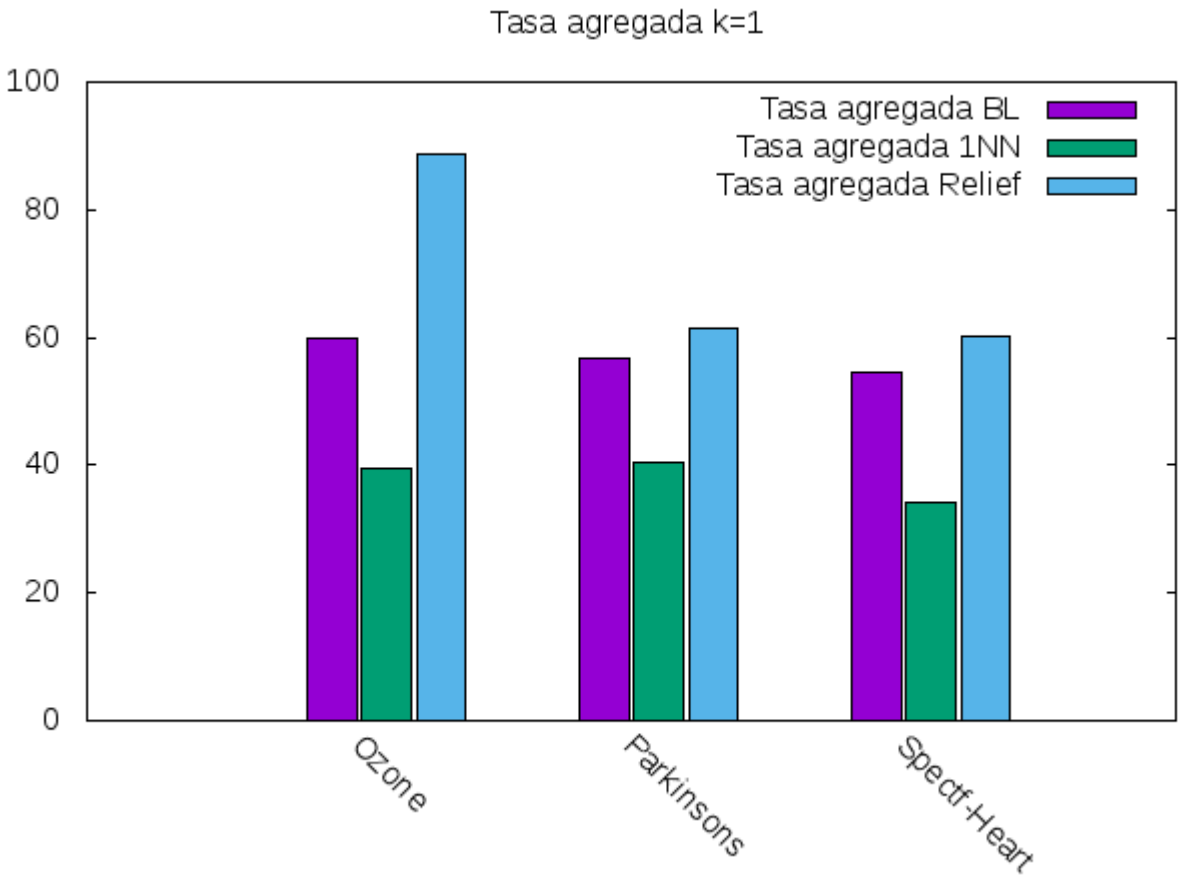
	Ozone				Parkinsons				Spectf-Heart			
	%_clas	%_red	Agr.	T (seg)	%_clas	%_red	Agr.	T (seg)	%_clas	%_red	Agr.	T (seg)
5-NN	80.9375	0.0000	40.4688	0.5133	79.1554	0.0000	39.5777	0.0624	69.9427	0.0000	34.9714	0.3648
Relief	80.6250	97.5342	89.0796	1.0847	83.2436	43.4783	63.3609	0.1350	70.7563	48.0000	59.3782	0.7112
BL	81.8750	44.3836	63.1293	324.8649	78.1028	40.8696	59.4862	14.7150	71.7418	55.5556	63.6487	162.8578

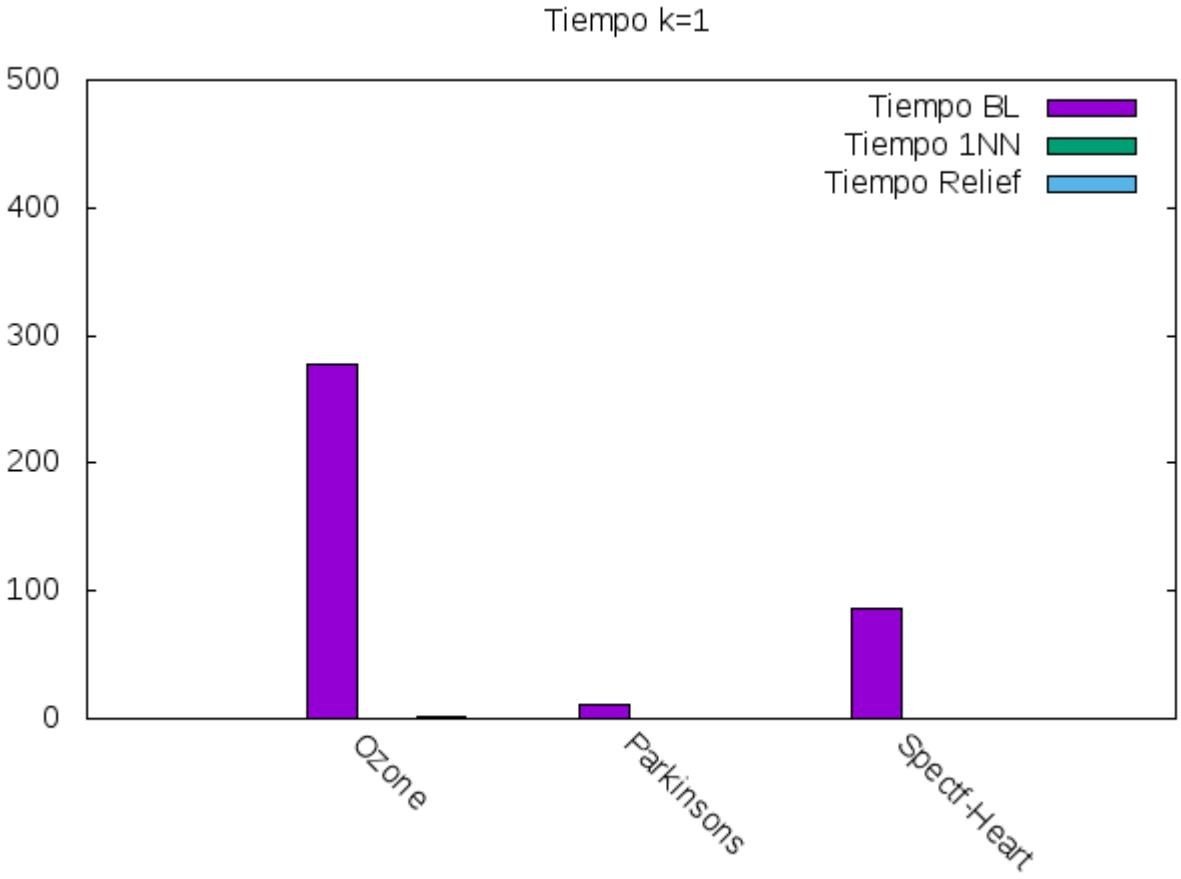
Cuadro 12: Resultados globales con K=5

Como se puede comprobar en las tablas los parámetros de ejecución de los casos han sido obtenidos variando el valor de k con 1,3,5 para comprobar si los resultados y tiempos variaban con mejor o peor tendencia. En cada tabla se puede observar que he obtenido la valoración de la tasa de simplicidad, la tasa de aciertos, la tasa agregada y el tiempo para cada algoritmo implementado y para cada conjunto de datos proporcionado en el guión de la práctica. Así mismo se proporciona la media de los resultados al final de cada tabla, siendo además las tres últimas tablas una recopilación de las medias para cada valor de k.
Cabe destacar que con el objeto de que el código y resultados sean replicables se ha fijado la semilla de números aleatorios con el valor 123456789.
Para el análisis comparativo de los tres algoritmos he elaborado gráficas comparativas de los 3 en las tasas de clasificación, simplicidad y agregada así como en el tiempo.


Tasa clasificación k=1
Tasa clasificación BL
Tasa clasificación 1NN
Tasa clasificación Relief
100
80
60
40
20
0
Ozone
Parkinsons
Spectf-Hear


Tasa simplicidad k=1
Tasa simplicidad BL
Tasa simplicidad 1NN
Tasa simplicidad Relief
100
80
60
40
20
0
Ozone
Parkinsons
Spectf-Heart


Tasa agregada k=1
Tasa agregada BL
Tasa agregada 1NN
Tasa agregada Relief
100
80
60
40
20
0
Ozone
Parkinsons
Spectf-Heart


Tiempo k=1
Tiempo BL
Tiempo 1NN
Tiempo Relief
500
400
300
200
100
0
Ozone
Parkinsons
Spectf-Heart

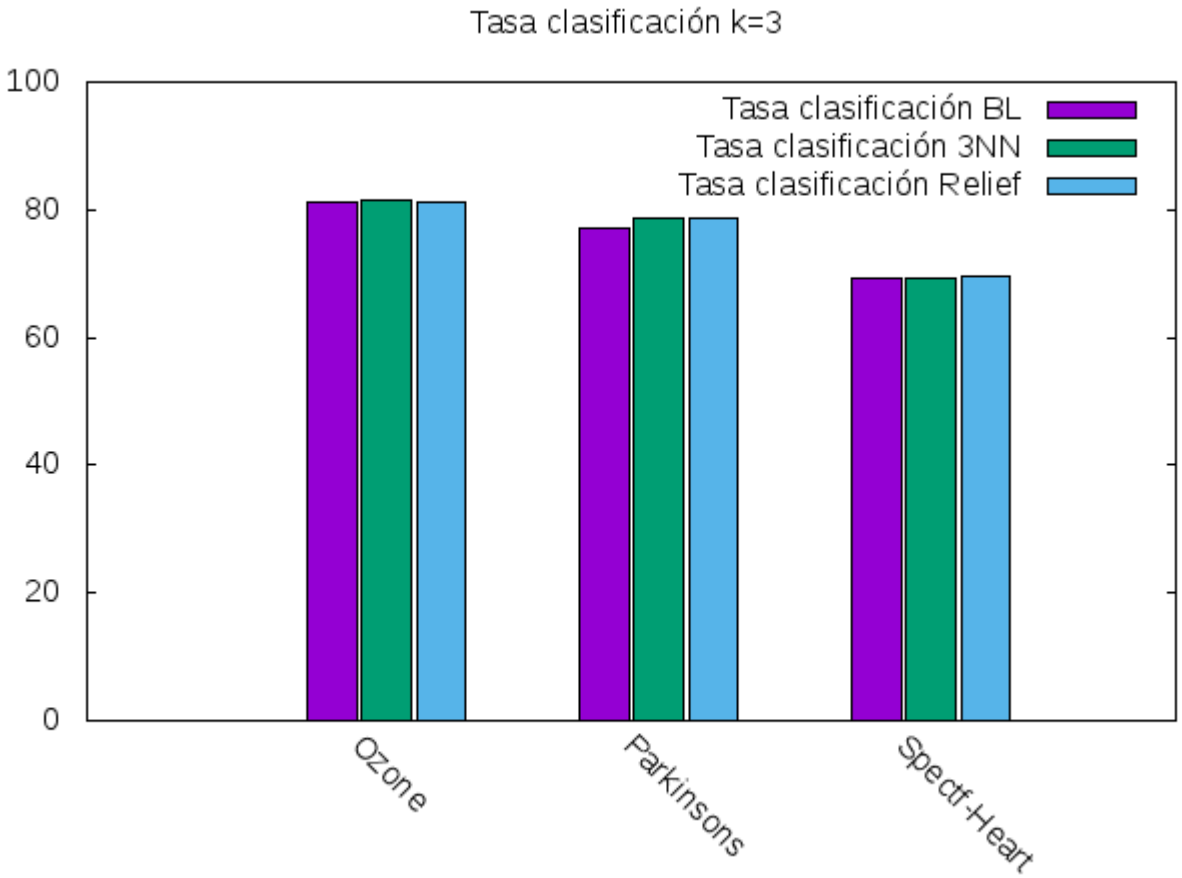
Estas gráficas son para K=1. En las mismas podemos observar que el peor algoritmo es el 1NN, cosa que era de esperar ya que su calificación en simplicidad siempre es cero al ser el vector todo unos.
En la tasa de clasificación podemos comprobar que todos están parejos aunque los dos mejores son Relief y búsqueda local, siendo el primero mejor en los conjuntos Ozone y Spectf-Heart y el segundo emjor en el conjunto Parkinsons.
En la tasa de simplicidad podemos ver que el mejor algoritmo sin duda es Relief qu siempre obtiene mejores valores que BL.
Por último en el tiempo podemos ver que incluso 1NN y Relief se quedan casi fuera de la escala, ya que BL tarda mucho más en completarse que Relief y 1NN. Por lo tanto vamos a analizar estos datos un poco más en profundidad buscando una explicación e intentando quedarnos con el mejor de los algoritmos para estos conjuntos.
En cuanto a la tasa de clasificación no podemos tomar una decisión clara, ya que los tres algoritmos tienen una valoración muy pareja por lo que debemos basarnos en el resto de gráficas para tomar una decisión. En cuanto a la tasa de simplicidad claramente el ganador es el algoritmo Relief. Esto puede producirse porque las clases están muy diferenciadas en dos clusters diferentes y por tanto al hacer la operación de resta con el enemigo más cercano hacemos una resta de una distancia muy grande produciendo que nuestro vector tenga muchos valores próximos a cero y por tanto obteniendo una tasa de simplicidad muy alta.
Si observamos la tasa agregada para hacer una comparativa conjunta vemos que 1NN queda muy atrás del resto, indicándonos que hemos mejorado los resultados con respecto al vector con todas las posiciones a uno. En cuanto a BL y Relief vemos que en Spectf-Heart y Parkinsons están más parejos en cuanto a resultados, pero en Ozone vemos una gran mejora comparando Relief con BL. Esto si nos remitimos a las gráficas anteriores observamos que es gracias a la tasa de simplicidad que dicho algoritmo ha conseguido en el conjunto de datos.
Hasta este punto vemos que el algoritmo más ventajoso es Relief en cuanto a resultados, lo que nos lleva a mirar el tiempo para ver qué algoritmo es el mejor. Aquí observamos que el peor con diferencia es BL, cuyos resultados únicamente están próximos al resto de algoritmos en el conjunto Parkinsons, siendo en los otros dos conjuntos muy superiores al resto.
Por lo tanto podemos sacar las siguientes conclusiones de los algoritmos:

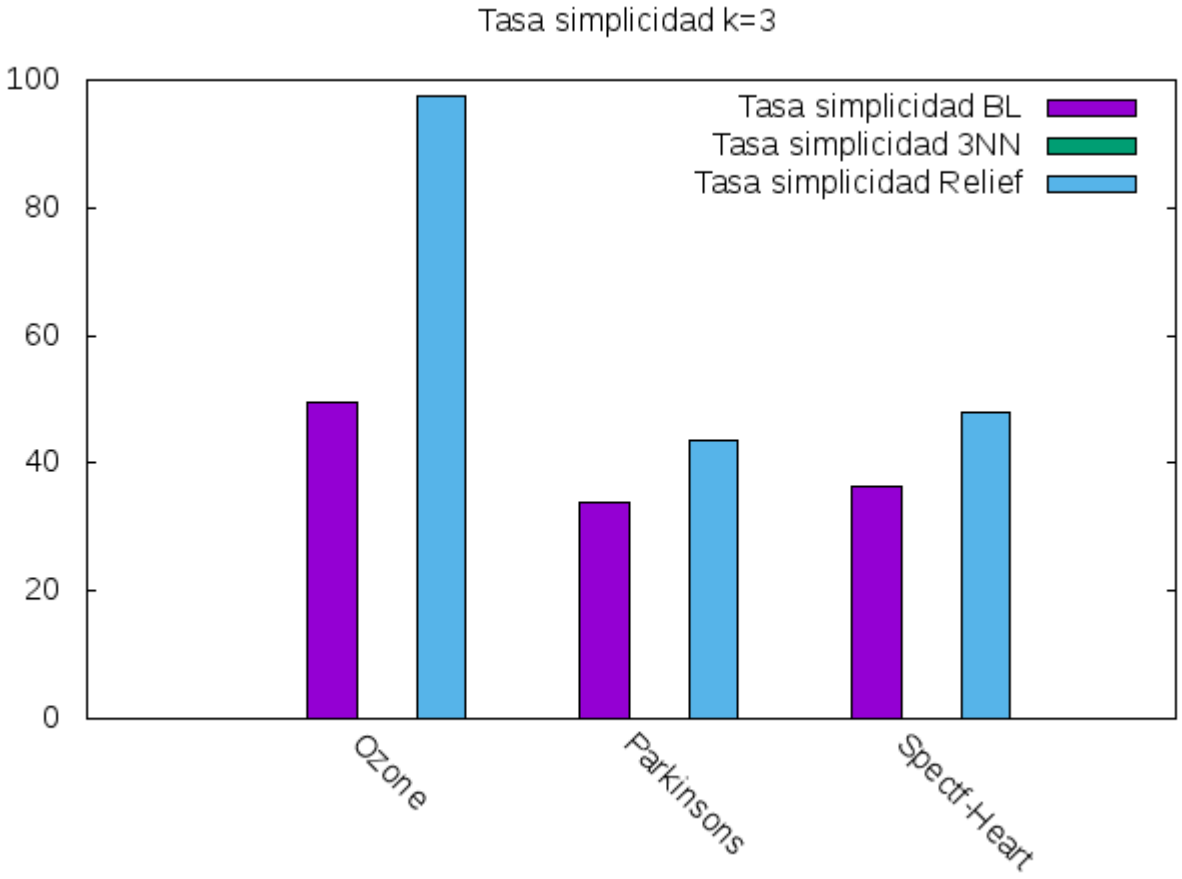
1. Relief obtiene mayor tasa de simplicidad que el resto de algoritmos gracias a que los clusters correspondientes a cada clase están separados entre sí.

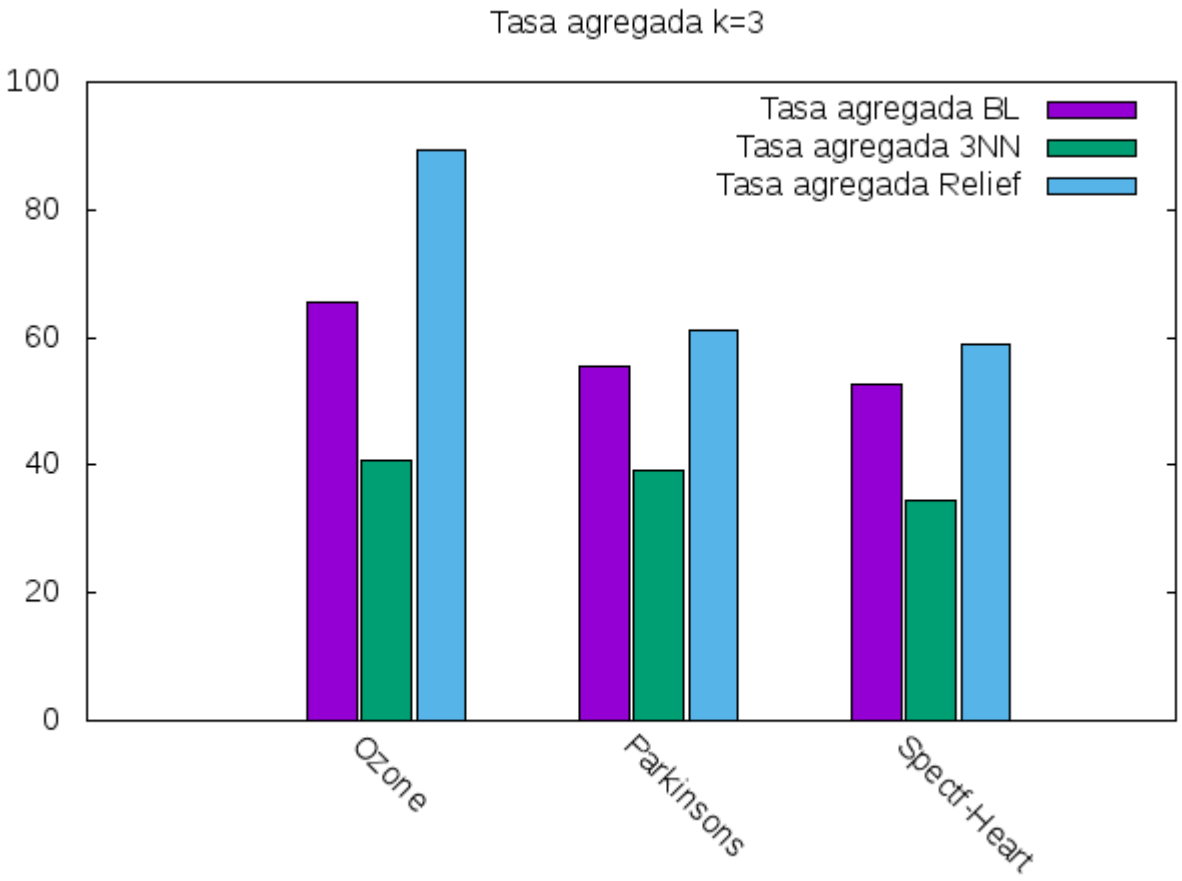
2. Búsqueda local es un algoritmo mucho más costoso que Relief en tiempo.

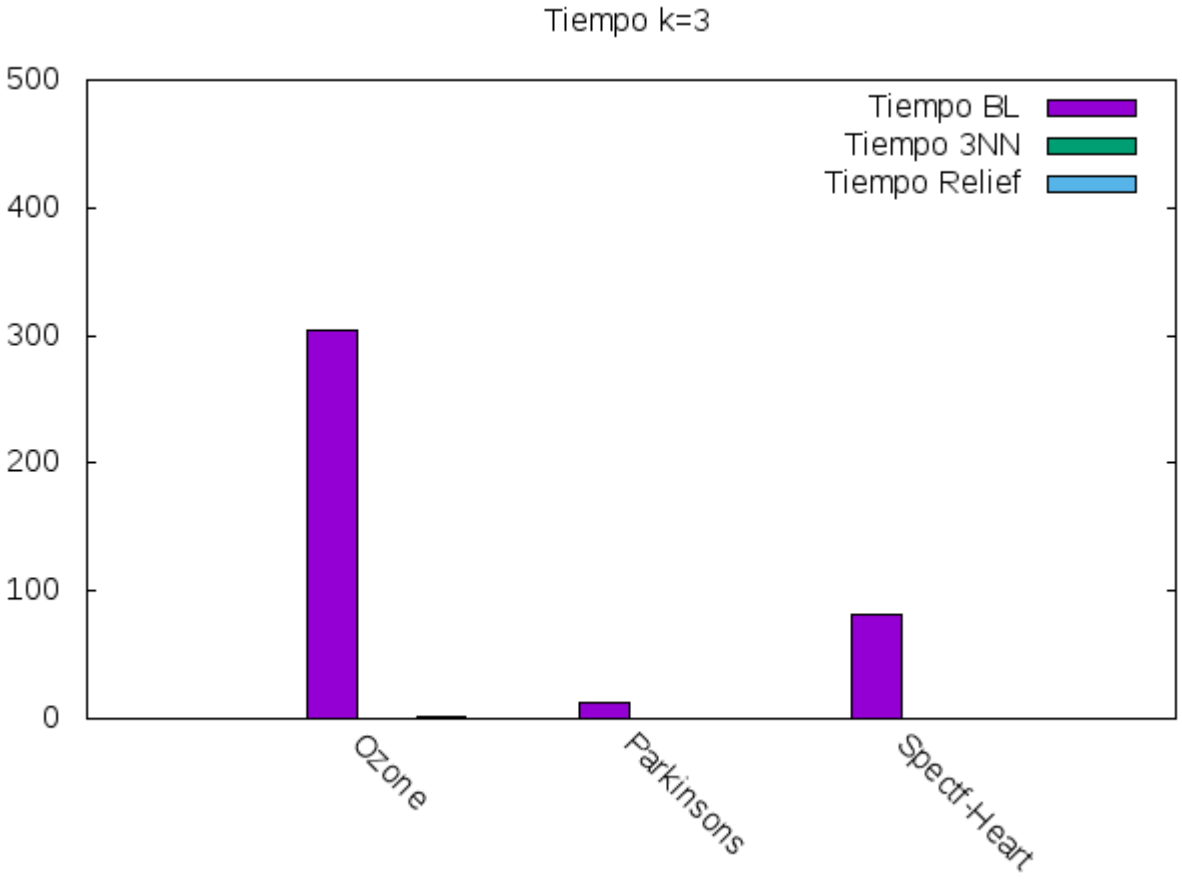
3. Búsqueda local se queda demasiado tiempo buscando soluciones, cosa que puede deberse a que se quede oscilando entorno a un máximo local de la función objetivo la mayoría del tiempo.

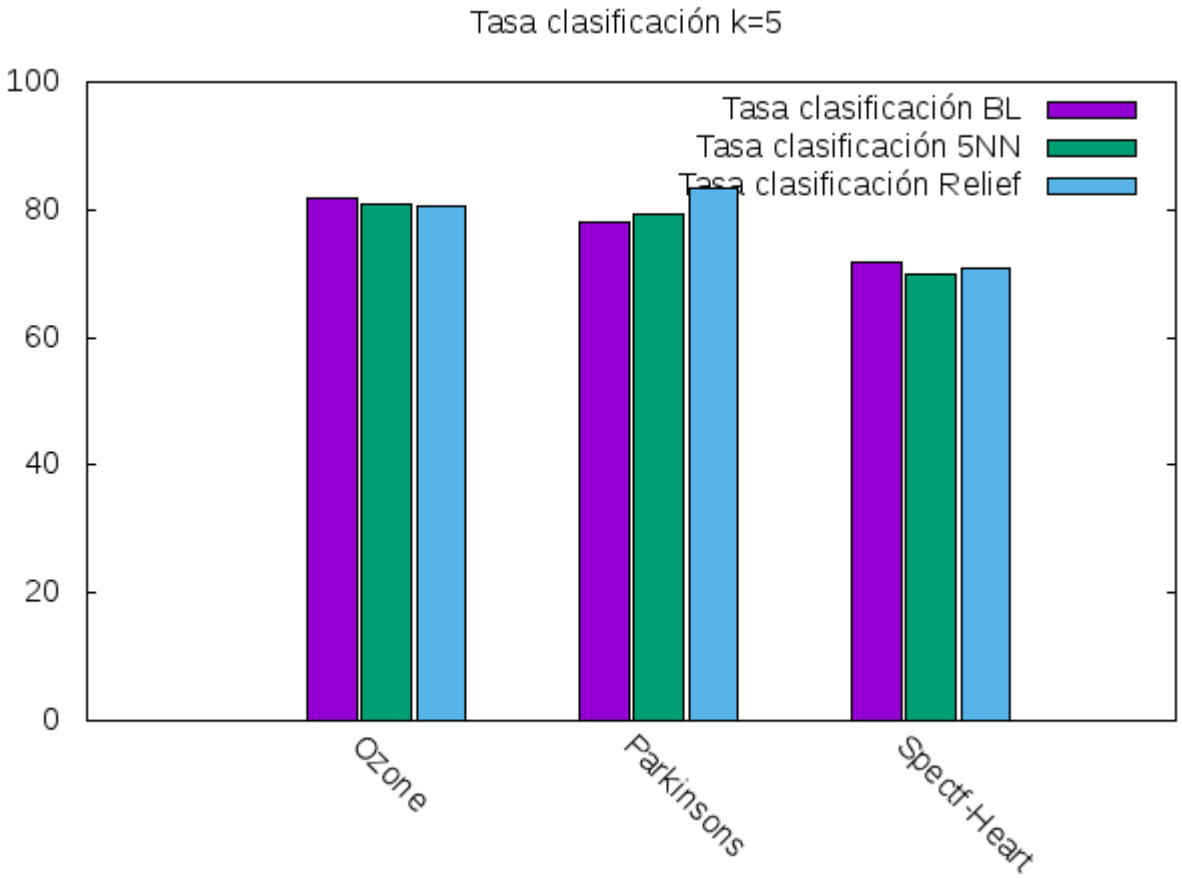
Por tanto para estos conjuntos de datos tenemos que el mejor algoritmo en términos generales es Relief, obteniendo mejores resultados en muchas de las situaciones a un coste en tiempo mucho menor que búsqueda local.

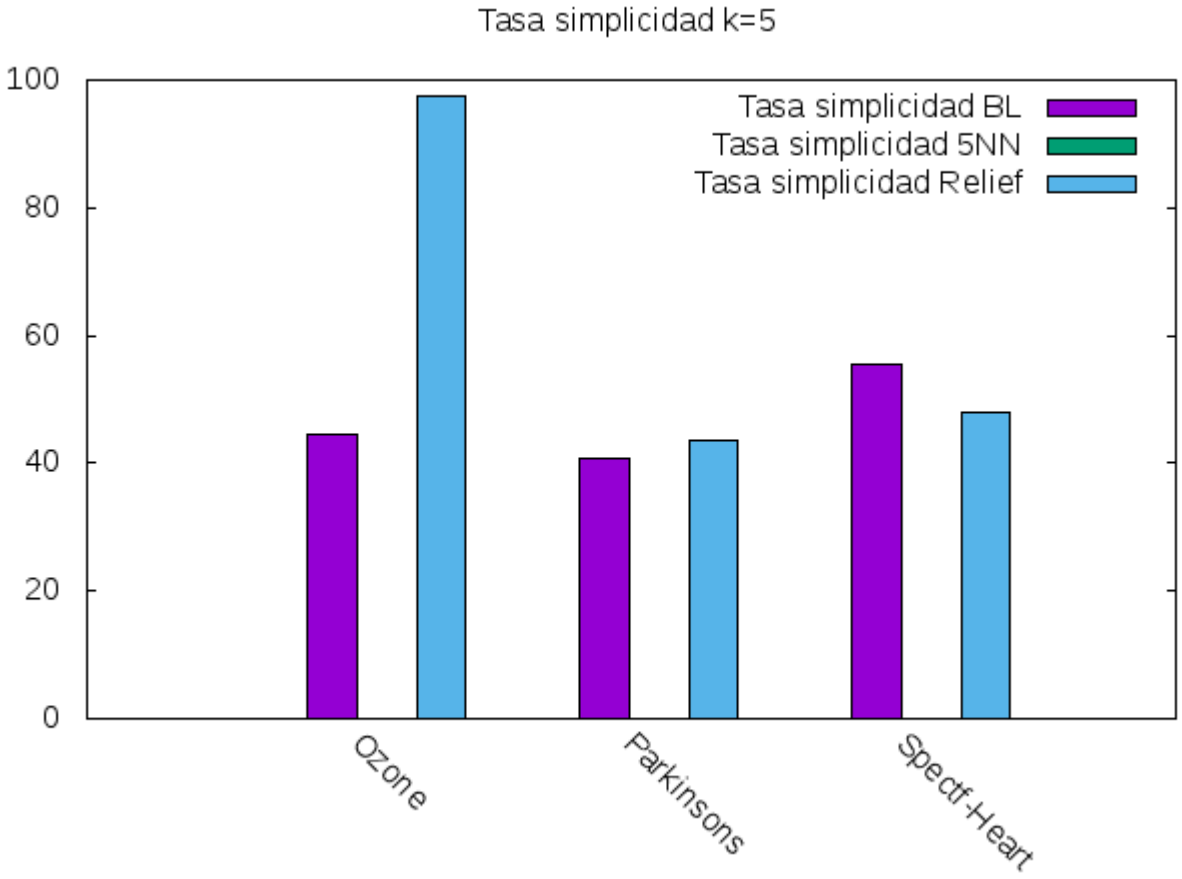

Tasa clasificación k=3
Tasa clasificación BL
Tasa clasificación 3NN
Tasa clasificación Relief
100
80
60
40
20
0
Ozone
Parkinsons
Spectf-Heart

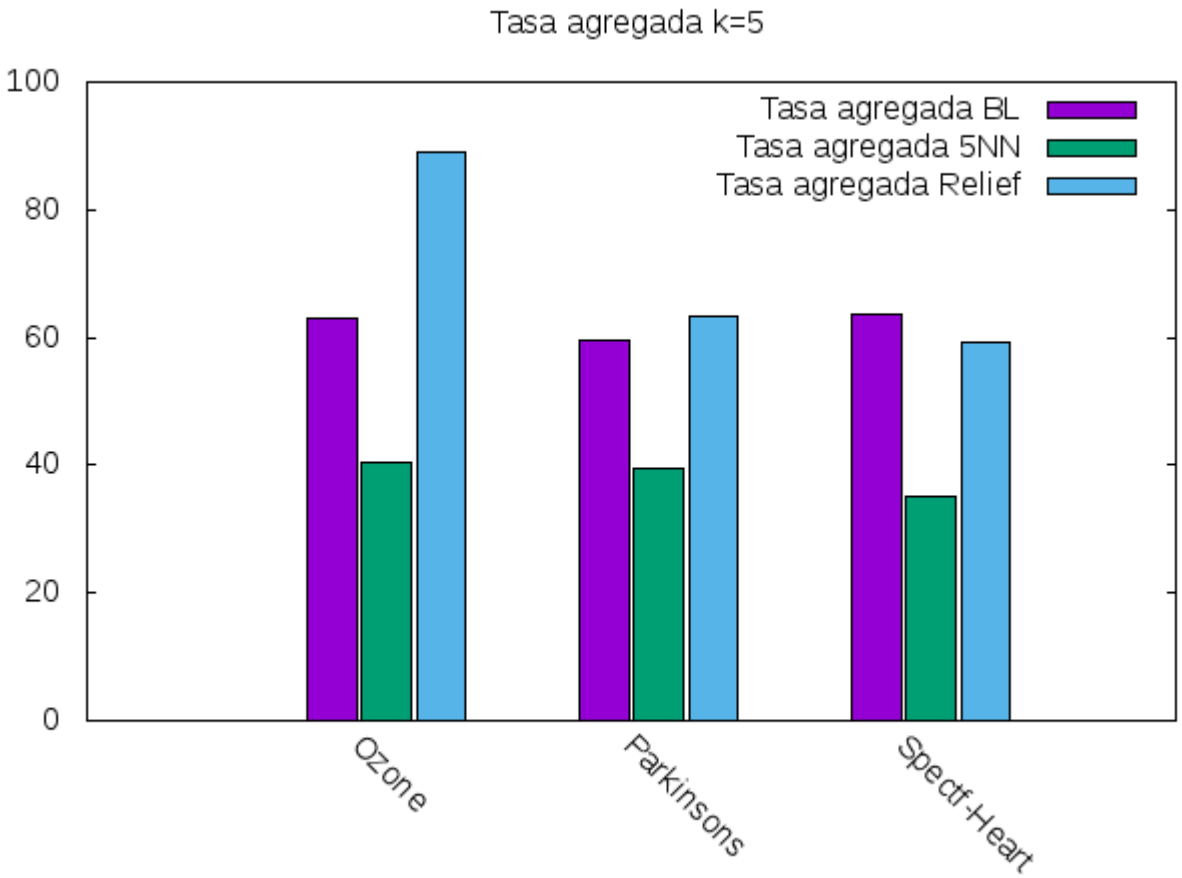

Tasa simplicidad k=3
Tasa simplicidad BL
Tasa simplicidad 3NN
Tasa simplicidad Relief
100
80
60
40
20
0
Ozone
Parkinsons
Spectf-Heart

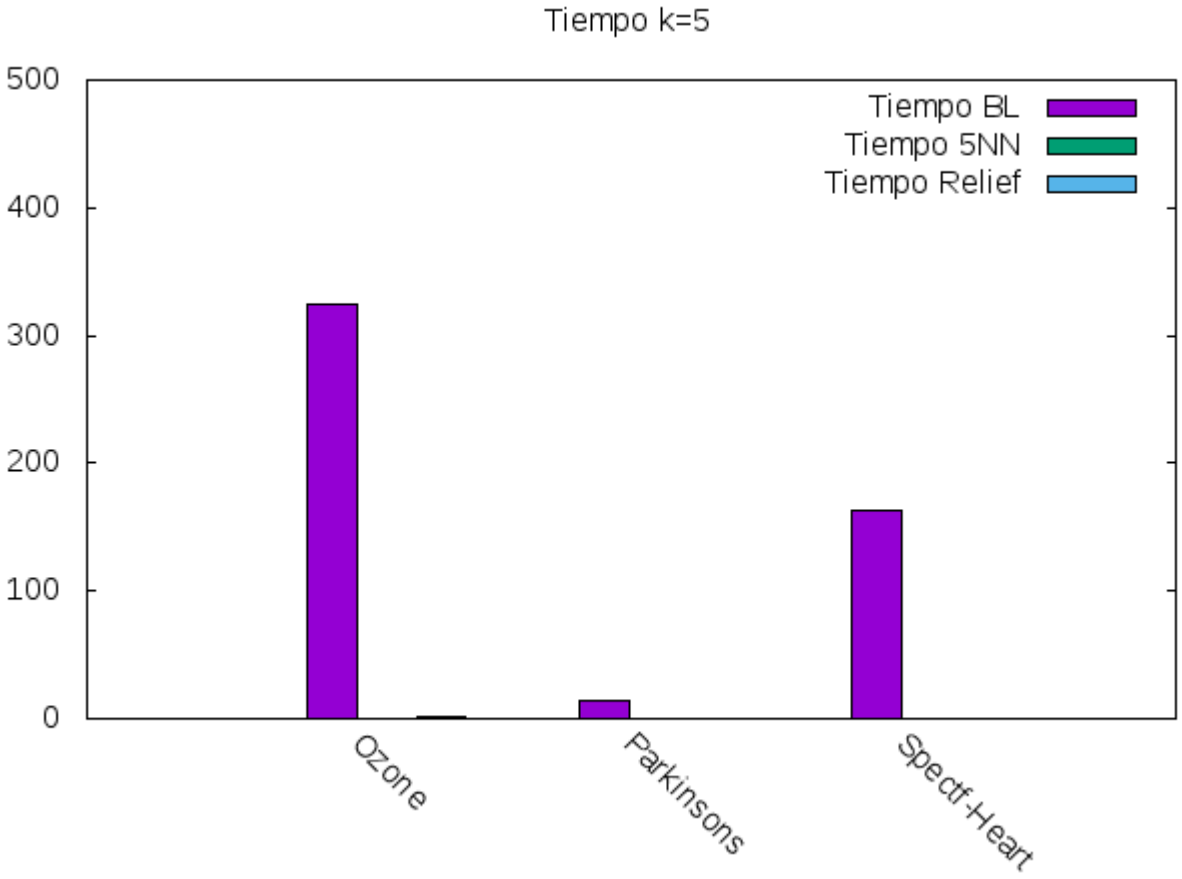

Tasa agregada k=3
Tasa agregada BL
Tasa agregada 3NN
Tasa agregada Relief
100
80
60
40
20
0
Ozone
Parkinsons
Spectf-Heart


Tiempo k=3
Tiempo BL
Tiempo 3NN
Tiempo Relief
500
400
300
200
100
0
Ozone
Parkinsons
Spectf-Heart


Tasa clasificación k=5
Tasa clasificación BL
Tasa clasificación 5NN
Tasa clasificación Relief
100
80
60
40
20
0
Ozone
Parkinsons
Spectf-Heart


Tasa simplicidad k=5
Tasa simplicidad BL
Tasa simplicidad 5NN
Tasa simplicidad Relief
100
80
60
40
20
0
Ozone
Parkinsons
Spectf-Heart


Tasa agregada k=5
Tasa agregada BL
Tasa agregada 5NN
Tasa agregada Relief
100
80
60
40
20
0
Ozone
Parkinsons
Spectf-Heart


Tiempo k=5
Tiempo BL
Tiempo 5NN
Tiempo Relief
500
400
300
200
100
0
Ozone
Parkinsons
Spectf-Heart

Si observamos el resto de gráficas para k=3,5 podemos observar que las conclusiones no varían ya que se mantienen los mismos resultados. Si podemos ver en las gráficas que conforme subimos el valor de k la tasa de clasificación de todos los algoritmos aumenta levemente.