Universidad De Granada

Grado ingeniería informática (2017 – 2018)

METAHEURÍSTICAS

Práctica 3: APC (EF, BLR y ED)

Trabajo realizado por: Antonio Miguel Morillo Chica – 77379031Y - Grupo 3

Índice

1. Descripción del problema.

El problema expuesto para la realización de la práctica es el "**Problema del Aprendizaje de Pesos en Carácteriticas**". Este problema consiste en optimizar el rendimiento del clasificador a partir de una serie de pesos asociados a unos determinados ejemplos. Disponemos de una muestra de objetos clasificados $w_{0,} w_{1,} \ldots, w_n$ representados en función de una serie de atributos en un vector de características: $(x_1(w_i), \ldots, x_n(w_i))$ cada objeto pertenecerá a una clase asociada:

$w_k = (x_1(w_k), \ldots, x_n(w_k)) \rightarrow C_{ik}$

El objetivo será obtener un sistema que pueda clasificar todos los ejemplos de forma automática infiriendo esta clasificación através de las características. Para ello deberemos será entrenarnos sobre un conjunto de datos para aprender y posteriormente usaremos un conjunto de datos de prueba para validar el aprendizaje relizado.

El clasificador hace uso de la distancia, que se define como la similitud que tendrán dos ejemplos con relación a las características de los mismos. La menor distancia entre un ejemplo que se quiere clasificar y todos los de la muestra se usará para determinar la clasificación de este nuevo.

En la práctica generaremos muchos conjuntos de pesos através de diferentes técnicas para obtener el vector de pesos que mejor clasifique los ejemplos. Las distintas formas en las que obtendremos estos vectores serán:

- De forma aleatoria (aunque no lo usemos para ver los resultados)
- Todos los pesos a 1.
- Algoritmo Relief.
- Algoritmo aleatorio con Búsqueda local.

2. Aplicación de los algoritmos.

A continuación describiremos la descripción del esquema de representación de soluciones, la descripción de la función objetivo y la de los operadores comunes usados.

2.1. Descripción esquema de representación soluciones.

La solución la denotaremos con W que será un vector de pesos asociados, donde cada posición será el peso de la caracteristica asociada y que habrá tantas como atributos tenga el ejemplo: $W=(w_1, w_2, \ldots, w_{N-1}, w_N)$ dónde $w_i \in [0,1], N \in \mathbb{N}$

w_1	w_2	...	w_{n-1}	w_n

Si el valor asociado a la posición i es cero quiere decir que la característica no se usará en el cálculo, en cambio si es uno quiere decir que se usará completamente en el cálculo de la distancia. Los valores intermedios del intervalor indican el grado de importacia de cada característica.

2.2. Descripción de la función objetivo.

El objetivo en este problema es maximizar la función objetivo que tiene dos criterios, **precisión** y **simplicidad**, por un lado, la tasa de clasificación que mide el porcentaje de ojetos bien clasificados, es decir, la precisión, siguiendo la siguiente formula: $Tasa-Clasificación = 100 \cdot \frac{n^o\ bien\ clasificados}{n^o\ ejemplos}$

```
bienClasificados = 0
for i in numEjemplosClasificados
   if etiquetas[i] == NN1(datos[i])  then
      bienClasificados++
return bienclasificados/numClasificados
```

Por otro lado la tasa de reducción que mide el porcentaje de características descartadas, aquellas cuyo peso esté cercano a cero en W, con respecto a "n".

Consideraremos un umbral de 0.1 en w_i para considerar que se descarta una característica: $Tasa-Deducción=100\cdot\frac{n^o.valores\, w_i<0.2}{n^o\, características}$

```
NumDescartados = 0
for i in W
   if W[i] < 0.2 them
      numDescartados++;
return numDescartados/W.size()
```

La función objetivo es la combinación de ambos criterios donde ambos tendrán para nosotros la misma importancia por tanto lo que buscamos es maximizar el accierto a la vez que consideramos el mínimo númeor de características posibles.

$$F(W)=\alpha\cdot tasa-clasificación(W)+(1-\alpha)\cdot tasa-reducción(W)$$

2.3. Operadores comunes.

Los siguientes operadores han sido usados para los diferentes procedimientos de los algoritmos o para la preparación de los valores en las características.

a) **Generación de soluciones iniciales aleatorias:** Usado para el calcular una posible valoración inicial para los pesos. Además de ser usados en la parte relacionada a los algoritmos genéticos.

```
W = vector(num_caracteristicas)
for in W
   W[i] = randFloat(0,1)
return W
```

Nota: De igual forma este operador ha sido implementado con la inicializacióon de los pesos a 1 para el algoritmo 1NN.

b) **Mutación del vector de Pesos**: Usado en el algoritmo de Búsqueda Local para la mutación uno a uno del vector de características.

```
mutar_vector(vector<float> w)
   w = pesos_actual
   posMutar = siguiente_pos
   if (siguiente_pos == numAtributos) then
      siguiente_pos = 0
      posMutar = 0
   end
   w[posMutar] = w[posMutar] + DistribucionNormal(media,desviacion)
   w[posMutar] = truncar(w[posMutar])
return w;
```

- **Corregido de la práctica anterior, antes las posiciones a mutar eran aleatorias.**

c) **Distancia Euclidia**: entre dos ejemplos sería:

```
float distasnciaEuclidia(e1,e2){
   w = solucion_actual;
   resultado = 0
   for i in num_caracteristicas {
      resta = e1[i] - e2[i]
      resultado += w[i] * resta * resta
   }
   resultado = resultado
   return resultado
}
```

- Es importante tener en cuenta que las características estén normalizados para que al calcular las distancias no se prioricen unos sobre otros.
- **Eliminada la raiz cuadrada en la implementación, como me dijo en el correo.**

d) **Operador para el torneo de selección:** Usado en con los Algoritmos Geneticos.

```
vector torneoSeleccion(){
   selección = vector(numPadres)
   poblacion = poblacion_actual
   for i in numPadres {
      p1 = randomInt(1,tamPoblacion)
      p2 = randomInt(1,tamPoblacion)
      if existosa(poblacion[p1]) >= existosa(poblacion[p2]) then
         seleccion[i] = poblacion[p1]
      else
         seleccion[i] = poblacion[p2]
   }
   return seleccion
}
```

e) **Operador cruce BLX**: Usado en en las diferentes algoritmos genéticos, estacionario y generacional.

```
vector of vector BLX(){
   w1 = cromosoma1
   w2 = cromosoma2
   for i in numAtributos {
      maximo = max(w1[i],w2[i])
      minimo = min(w1[i],w2[i])
      dif = maximo -minimo;
      w1[i] = randomFloat(min - dif*alpha, maximo + dif*alpha)
      w2[i] = randomFloat(min - dif*alpha, maximo + dif*alpha)
      truncar(w1[i],0,1)
      truncar(w2[i],0,1)
   }
   return <w1,w2>
}
```

f) **Operador cruce aritmético**: Usado en en las diferentes algoritmos genéticos, estacionario y generacional.

```
vector CA(){
   w = cromosoma
   w1 = comosoma1
   w3 = comosoma2
   for i in numAtributes
      w[i] - (w1[i]+w2[i])/2
   return w
}
```

g) **Operador enfriar**: Usado en el enfriamiento simulado para el calculo de la probillidad de intrucir soluciones peores:

```
enfriar(t) {
   beta = (temperatura_inicial - temperatura final) /
           M*temperatura inicial*temperatura final

   t = t / (1 + beta*t)
}
```

h) **Operador cruce aleatorio:** Usado en el algoritmo Evolución Diferencial current-to-1.

```
p1 = padre1
p2 = padre2
p3 = padre3
return truncar(p1 +f*(p2-p3))
```

i) **Operador cruce mejor+aleatorio:** Usado en el algoritmo Evolución Diferencial best.

```
w = solucionActual
best = mejorSolucion
p1 = padre1
p2 = padre2
return truncar(w + f*(best-2) + f*(p1-p2))
```

3. Descripción de los algoritmos específicos.

A continuación mostraré y describiré los dos algoritmos usados para esta práctica a excepción del 1NN que lo explicaremos en el siguiente apartado, en la descrpción del algoritmo de compraración.

3.1. Algoritmo RELIEF.

El algoritmo RELIEF se utiliza para el calculo del vector de pesos asociado a las características. El algoritmo comieza con el vector de pesos w inicializado a 0. Por todos los ejemplos calcula su amigo y su enemigo, el amigo es un ejempllo cercano de su misma clase, el enemigo es un ejemplo de clase distinta más cercano.

Una vez calculado su amigo y enemigo recorre todos los pesos y los modifica sumandole la diferencia entre el peso amigo y enemigo más cercano. Tras esto se normalizan los valores de los pesos.

```
vector<float> RELIEF(){
   matriz = ejemplos_y_características
   w = vector(matriz.getNumAtributos(),0)
   for i in num_ejemplos {
      pos_enemigo = buscarEnemigo(matriz[i])
      pos_amigo = buscarAmigo(matriz[i])
      for j in num_caracteristicas {
         enemigo = matriz[i][j] - matriz[pos_enemigo][j]
         amigo = matriz[i][j]  matriz[pos_amigo][j]
         w[j] += enemy - friend
      }
   }
   w = normalizar(w)
   return w
}
```

- Los datos se normalizan por si aparecen valores que se salen de nuestro rango.
- Matriz contiene por filas los ejemplos y cada columna es su vector de características.

3.2. Algoritmo de Búsqueda Local.

El algoritmo de busqueda local transforma una solución inicial en otra atavés de una serie de mutaciones. El algoritmo necesita una serie de datos de entrada, numero de evaluaciones y vecinos, el primero porque el tamaño del entorno es infinito y necesitamos un límite de evaluaciones, el segundo, vecinos, se usa para saber cuantas mutaciones mínimas por característica se tiene que hacer si no se encontrase una solución mejor a la actual.

Por cada mutación se llama a la función de evaluación para saber si es mejor que la solución optima actual, en tal caso se actualizan los datos y se cambia la solución optima por la nueva. En caso de que ninguna mutación surja efecto, el vector devuelto debería ser el inicial, ninguna modificación se lleva acabo.

```
vector<float> BL(num_evals, vecinnos, desviacion){
   num_vecinos = 0 ; mejor = 0 ; evals = 0
   w = solucion_actual
   while (evals < num_evals && num_vecinos < vecinos*num_caracteristicas) {
      w_nuevo = w;
      w_nuevo = mutar(w_nuevo)
      if evaluar(w_nuevo) > mejor {
            mejor = evaluar(w_nuevo)
            w = w_nuevo
            num_vecionos = 0
      }
      evals++ ;  vecinos++
   }
   return w_nuevo
}
```

- El la práctica el numero de evaluaciones es de 1500.
- Sin embargo en número de vecinos es 20 que, en nuestro datos:
  - S-Heart: 45 características * 20 = 900 mutaciones como mínimo.
  - Parkinson: 23 características * 20 = 460 mutaciones como mínimo.
  - Ozone: 73 características * 20 = 1460 mutaciones como mínimo

3.3. Algoritmo Genético Generacional

El algoritmo comienza con un conjunto de comosomas aleatorios. En cada generación se cruza y mutan una serie de comoosomas con una probabilidad de hacerlo. El calculo de saber que cromosomas se cruzan o mutan es ineficiente por lo que calcula el número de comosomas exacto que se van a cruzar o mutar.

Aquí es donde tenemos en cuenta el tipo de cruce que vamos a realizar si BLX o Aritmético que hay que tenerlo en cuenta para la selección,ya que, si es el cruce BLX, la selección devuelve tantos padres como tamaño tenga la población. Si es el cruce aritmético, la selección devolverá el doble de padres del tamaño de la población, al generar en este cruce, un solo hijo por cada dos padres.

Otro aspecto importante es la comparación entre un cromosoma a otro ya existente en la población anterior que evaluar uno de ellos. Esto se hace cuando se han cruzado y mutado los correspondientes cromosomas y se comprar si existía anteriormente.

Para conservar el elitismo de la población, antes de terminar el tratamiento de la generación, se comprueba si el mejor de la población anterior es mejor que el peor de la población generada. En caso de que así sea, se elimina el peor de la nueva generación y se añade el mejor de la anterior.

```
poblacion = poblacionActual
poblacionExitosa = poblacionActualExitosa
numCruces = round(probCruzar * tamPoblacion)
numMutaciones = round(probMutar * tamPoblacion * numAtributos)
iEval = 0
while iEval < numEval {
   nuevaPobacion = seleccion()
   nuevaPoblacionExistosa = vector(tamPoblacion)
   for i in numCruces
      nuevaPoblacion[i] = cruzar(nuevaPoblacion[i])
   for i in numMutaciones
      nuevaPoblacion[i[ = mutar(nuevaPoblacion[i])
   for i in tamPoblacion {
      iguales = false
      j = 0
      while !iguales && j<tamPoblacion {
         if nuevaPoblacion[i] == poblacion[j]
            iguales = true
         j++
      }
      if iguales
         nuevaPoblacionExistosa[i] = poblacionExistosa[j-1]
      else
```

```
         nuevaPoblacionSatisdacible[i[ = evaluar(nuevaPoblacion[i])
   }
   if primero(poblacion) > ultimo(nuevaPoblacion)
      actualizar(primero(poblacion),ultimo(nuevaPoblacion))

   poblacion = nuevaPoblacion
   poblacionExistosa = nuevaPoblacionExistosa
   iEval += tamPoblacion
}

return mejorSolucion(), mejorExistosa
```

- **iEval**: número de evaluaciones que se realizan. Por cada iteración tantas evaluaciones como nuevos cromosomas hay en la población. En nuestro caso 30.
- **numEval**: número máximo de evaluaciones permitidad. En uestro caso llega a las 15.000 evaluaciones. Que provocan unas 500 generaciones de cromosomas nuevos.
- **poblacion**: vector con los cromosomas actuales que usará el algoritmo.
- **poblacionExitosa**: vector con la tasa de accierto de cada cromosoma.
- **numCruces**: numero máximo de cruces que se pueden realizar.
- **numMutaciones**: numeor máximo de mutaciones que se pueden realizar.

3.4. Algoritmo Genético Estacionario

El algoritmo comienza con un conjunto de comosomas aleatorios. En cada generacion se crean dos cromosomas y se cruzan. Debido a esto no es necesario calcular los cromosomas que se cruzarían. El número de mutaciones se calculan como en el **algoritmo generacional**. Por cada generacion se hace una selección que devolverá dos padres, en el caso de BLX y devolverá cuatro padres con el Aritmético.

Cuando ya se tienen los dos cromosomas cruzados, mutados y evaluados, se comparan con los dos peores de la población anterior. La nueva generación la componen todos los cromosomas de la generación anterior, excepto los dos últimos. Estos dos últimos se compararán con los dos nuevos cromosomas generados y solo los dos mejores de los cuatro que se comparan, forman parte de la nueva generación.

```
poblacion = poblacionActual
poblacionExitosa = poblacionActualExitosa
numCruces = round(probCruzar * tamPoblacion)
numMutaciones = round(probMutar * tamPoblacion * numAtributos)
iEval = 0
while iEval < numEval {
   nuevaPobacion = seleccion()
   nuevaPoblacionExistosa = vector(tamPoblacion)
   for i in numMutaciones
      nuevaPoblacion[i[ = mutar(nuevaPoblacion[i])
   for i in tamPoblacion {
      iguales = false
      j = 0
      while !iguales && j<tamPoblacion {
         if nuevaPoblacion[i] == poblacion[j]
            iguales = true
         j++
      }
      if (iguales) then
         nuevaPoblacionExistosa[i] = poblacionExistosa[j-1]
      else
         nuevaPoblacionSatisdacible[i[ = evaluar(nuevaPoblacion[i])
   }
   if primero(poblacion) > ultimo(nuevaPoblacion)
      actualizar(primero(poblacion),ultimo(nuevaPoblacion))

   poblacion = nuevaPoblacion
   poblacionExistosa = nuevaPoblacionExistosa
   iEval += tamPoblacion
}

return mejorSolucion(), mejorExistosa
```

3.5. Algoritmo Memético

Los Algoritmos Meméticos son Algoritmos Genéticos a los que se le aplica una Búsqueda Local. En este caso, el esquema del Algoritmo Memético es similar al del Algoritmo Genético Generacional, salvo que después de conseguir las evaluaciones de los cromosomas y antes de realizar la operación de conservación del elitismo de la población, se aplica la Búsqueda Local a una serie de cromosomas.

A qué serie de cromosomas se le aplica la Búsqueda Local, depende del tipo de Algoritmo Memético que se quiera realizar.

```
[…]
if (iguales) then
   nuevaPoblacionExistosa[i] = poblacionExistosa[j-1]
else
   nuevaPoblacionSatisdacible[i[ = evaluar(nuevaPoblacion[i])
evaluaciones = BusquedaLocal(nuevaPoblacion, nuevaPoblacionExistosa)
iEval += evaluaciones
if primero(poblacion) > ultimo(nuevaPoblacion)
   actualizar(primero(poblacion),ultimo(nuevaPoblacion))
[...]
```

Las evaluaciones realizadas por la busqueda local se suman a las evaluaciones del Memetico. A grandes rasgos y suponiendo que se dispone de un contenedor que indique la posición del cromosoma al que se le va a aplicar la Búsqueda Local sería:

```
numEval = 0
for i in tamNuevaPoblacion{
   if posBL[i] == 0 then
      nuevaPoblacion[i] = getSolucionBL(nuevaPoblacion[i])
      nuevaPoblacionExistosa[i] = getExistosaBL(nuevaPoblacion[i])
      numEval += numEvaluacionesBL(nuevaPoblacion[i])
   }
}
return numEval
```

3.6. Algoritmo Enfriamiento Simulado.

El enfriamiento simulado trabaja con todo el tiempo con tres soluciones. Al principio genera una solución aleatoria, la segunda que se utiliza para generar las mutaciones y ver si es mejor que la solución anterior y una última que reserva que es la mejor solución hasta el momento.

El algoritmo se basa en comenzar con un valor de temparatura muy alto, la temparatura designa lo fuerte que será la mutación, al princìo es alta y va disminullendo simulando la idea del temple extraida de la metalurgia. El metal está al rojo y se le da su mayor forma, luego mientras disminulle la temperatura se "corriguen" las imperfecciones.

En cada iteraacción se generan una serie de soluciones mediante mutaciones de la solución actual. Esta mutacióon puede ser mejor o peor que la original pero se sustituirá dependiendo de:

- Si la solución mutada mejora a la actual.
- Mediante una probabilidad de sustituir cuando la solución es peor que la original.

Debido que existe la posibilidad de guardar soluciones peores es necesario ir guardando la mejor solución hasta el momento.

```
Enfrimiento_Simulado(w, maxEval, maxExito){
   w = SolucionAleatoria()
   wEval = evaluar(w)
   mejorW = w
   mejorEval = wEval
   t = temperaturaInicial(wEval)
   numEval = 1
   éxito = inf
   while (numEval < maxEval && éxito != 0) {
      vecinos = 0
      éxito = 0
      while (vecinos < maxVecino && éxito < maxExito) {
         wAux = mutar(w)
         wEvalAux = evaluar(wAux)
         numEval++
         diferencia = wEvalAux - wEval
         if (diferencia > 0 || random(0,1) >= exp(diferencia/t)) {
            w = wAux
            wEval = wEvalAux
            éxito++
```

```
            if (mejorW < wEval) {
               mejorW = w
               mejorEval = wEval
            }
         }
         vecionos++;
      }
      enfriar(t)
   }
   return mejorW, mejorEval
}
```

El calculo de la temperatura inicial se consigue de la siguiente forma:

```
var temperaInicial(eval){
   return -mu*eval / ln(phi)
}
```

Para el calculo del enfriamiento de la temperatura lo he realizado de la siguiente forma:

```
enfriar(t) {
   beta = (temperatura_inicial - temperatura final) /
           M*temperatura inicial*temperatura final

   t = t / (1 + beta*t)
}
```

3.7. Algoritmo Búsqueda Local Iterativa.

El algoritmo de BL Irerativa es muy sencillo. Se basa en generar una solución inicial aleatoria a la que se le aplica una búsqueda local. Tras esto lo que hacemos es ir mutando ciertas caracteristicas y volvemos a aplicar una busqueda local.

```
Busqueda_Local_Iterativa(w, maxEval){
   t = atributosCambiar()
   w = generarSolucionInicial
   wEval = BL(w)
   for i in maxEval {
      wAux = mutar(w, t)
      evalAux = BL(wAux)
      if(evalAux > wEval) {
         w = wAux
         wEval = evalAux
      }
   }
   return w, wEval
}
```

La mutacion realizada la realizaremos de la siguiente forma. Hay que tener en cuenta que "t" indica el número de caracteristicas a mutar y que en nuestro caso sería de un 10%.

```
mutar(solucionActual,t){
   w = solucion actual
   posMutar = generarPosición sin repeticiones(t,1,numAtributos)
   for i in posMutar {
      w[i] = w[i[ + distribuciónNormal(media, desviacionStandar)
      truncar(w[i[,0,1)
   }
}
```

3.8. Algoritmo Evolución Diferencial.

El algoritmo de Evolución Diferencial comienza partiendo de una población aleatoria de un tamaño determinado. Acontinuación evalua e iiterta hasta que no pueda realizar más evaluaciones.

Por iteracción se eligen tantas soluciones aleatorias como tamaño tenga la población y para cada una de ellas realiza un cruce. Al terminar el cruce comprueba cual de los dos es mejor y conserva el mejor. Finalmente acaba devolviendo la mejor solución. Este algoritmo conserva el elitismo.

```
Poblacion = generarPoblacionAleatoria()
poblacionExitosa = evaluar(poblacion)
numEval = size(poblacion)
numPadres = 2 || 3 // depende del tipo de cruce
while (numEval < maxEval){
   for i in size(poblacion){
      padres = getPadres(poblacion, numPadres)
      posW = random(1, tamPoblacion)
      w = poblacion[posW]
      wEval = poblacionExistosa[posW]
      for j in numAtributos {
         if (random(0,1) <= pobabilidadCrueze)
            wAux[j] = cruze(w[j], padres[1][j[, padres[2][j], padres[3][j])
      }
      wEvalAux = evaluar(wAux)
      numEval++
      if (wEval < wEvalAux){
         poblacion[posW] = wAux
         poblacionExitosa[posW] = wEvalAux
      }
   }
   ordenarPoblacion()
}
return poblacion[0[, poblacionExitosa[0]
```

Los cruces implementados se ven en la sección de los operadores.

4. Descipción del Algoritmo de Comparación.

El algoritmo usado para la comparación es el algoritmo KNN en nuestro caso es el 1NN ya que solo usaremos el elemento más cercano y no los k más cercanos. Para determinar la cercania usaremos la distancia euclídia ya que los valores de las variables no son nominales, en tal caso usariamos hamilton por ejmplo.

```
pair<string, float> 1NN( ejemplos_y_caracteristicas, clasificacion,
ejemplo_actual) {
   etiqueta_resultado = v.etiquetas[0]
   distancia_minima = distanciaEucludia(v.etiqueta[0], ejemplo)
   for i in num_ejemplos{
      aux = distanciaEuclidia(matriz[i], ejemplo)
      if aux < distancia_minima them {
         distancia_minima = aux
         etiqueta_resultado = v_etiquetas[i]
      }
   }
   return <etiqueta_resultado, distancia_minima>
}
```

El algoritmo 1NN se utilida para clacular la clase de un ejemplo que en principio se desconocía. Devuelve la etiqueta de la clase comparando con todos los ejemplos de la muestra. Se usa Leave-One-Out para no compararlo consigo mismo ya que sino la distancia con el mismo sería 0.

Al principio guardamos la distancia y la etiqueta del primer ejemplo. Recorremos el vector completo calculando la distancia con cada uno, si es menor actualizamos los valores distancia_mínima y la etiqueta_resultado hasta revisar todos los ejemplos.

Finalmente se devolverá la etiqueta de distancia mínima con respecto al ejemplo y su ditancia que será usada para otros algoritmos.

5. Desarrollo de la práctica.

La práctica ha sido desarrollada en C++ sin ningún uso de libreerías externas salvo las aportadas y las típicas de la STL. Sin embargo al compiezdo del desarrollo se iba a usar la librería Arff encontrada en github, una serie de clases que sirven para lectura de estos archivos. Finalmente fue descartado porque daban errores con los archivos propuestos y la lectura de los mismos se realizó a mano. El código de evaluación de los algoritmos ha sido paralelizado gracias a OpenMP visto y usado en asignaturas como Estructura de los Computadores. Además el código se compila con optimización O3.

La practica está compuesta por cuatro clases principales y dos ficheros con funciones "amigas" para randoms y medida de tiempos en las ejecuciones.

- **Datos**: Se usa para la lectura de los datos de entrada así como el procedimiento para la división entre los datos de entrenamiento y los datos de prueba de forma aleatoria en dos mitades iguales. Además contendrá los vectores de las etiquetas, de entrenamiento y prueba y el vector de pesos asociado a las características.
- **Clasificador**: Contiene los algortirmos de comparación y RELIEF para el calculo de los pesos. Además será donde he impleentado el operador de la distancia euclídia y la función de evaluación.
- **Búsqueda Local**: Contine el clasificado rque se ha usado ya que este encapsula un objeto datos que contiene el vector a mutar, además contiene las operaciones de mutación para el vector.
- **Genéticos**: Contine los algoritmos genéticos estacional, generacional, y memético además de los cruces necesarios para ejecutar los algorimos.
- **TSA**: Contiene los algoritmos de Enfiamiento Simulado y el algoritmo de Busqueda Local Iterativa con sus respectivas mutaciones y operadores.
- **DE**: Contiene la implementación de los dos variantes del Evolución Diferencial con sus respectivos operadores.
- **Main**: Será el encargado de medir los tiempos y mostrar los resultados de la tasa de clasificación, de red y el resultado de los tiempo.

5.1. Manual de la práctica.

La práctica contiene un makefile así que unicamente hay que ejecutar la orden make por terminal para compilar el proyecto. Para la ejecución:

```
$ ./bin/<ejecutable>    data/<fichero>.arff    <semilla>
```

6. Experimentos y analisis de los resultados.

6.1. Descripción de los problemas y valores parámetros.

Los problemas a los que nos hemos enfrentado son tres de variables numéricas donde deberemos de clasificar los ejemplos.

1. **Ozone**: es una base de datos para la detección del nivel de ozono según las mediciones realizadas a lo largo del tiempo. Tiene 320 ejemplos y consta de 73 atribustos y dos clases, dia normal o dia de alta concentración de ozono. **Deberemos de averiguar si los nuevos ejemplos será días con alto o baja concentración de ozono.** La ejecución para este problema es la siguiente:

   ```
   $ ./bin/p3    data/ozone.arff       62
   ```

2. **Parkinsons**: es una base de datos que 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. Tiene 195 ejemplos de 23 atributos y de dos clases, tiene parkinson o no tiene. **Deberemos de averiguar si los nuevos ejemplos serán personas con o sin parkinsons.** La ejecución para este problema es la siguiente:

   ```
   $ ./bin/p3    data/parkinsons.arff      94
   ```

3. **Spectf-heart**: es una base de datos que 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 de

45 atributos enteros y dos clases, paciente sano o patología. **Deberemos de averiguar si los nuevos ejemplos serán personas con o sin patologías**. La ejecución para este problema es la siguiente:

```
$ ./bin/p3    data/spectf-heart.arff    55
```

6.2 Resultados optenidos.

La tabla de los resultados se ajustan a los pedidos según la tabla aportada en la página web de la asignatura.

- Particiones 1NN:

	Ozone				Parkinsons				Spectf-heart			
	%_clas	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T
Partición 1	81,13	0,00	40,57	0,03	81,63	0,00	40,82	0,01	76,57	0,00	38,29	0,02
Partición 2	79,50	0,00	39,75	0,02	89,69	0,00	44,85	0,00	81,03	0,00	40,52	0,02
Partición 3	71,43	0,00	35,71	0,02	83,51	0,00	41,75	0,01	74,86	0,00	37,43	0,02
Partición 4	81,37	0,00	40,68	0,01	84,69	0,00	42,35	0,01	74,71	0,00	37,36	0,01
Partición 5	76,40	0,00	38,20	0,01	86,60	0,00	43,30	0,01	76,57	0,00	38,29	0,02
Media	77,97	0,00	38,98	0,02	85,22	0,00	42,61	0,01	76,75	0,00	38,37	0,02

- Particiones RELIEF:

	Ozone				Parkinsons				Spectf-heart			
	%_clas	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T
Partición 1	79,25	20,75	50,00	0,01	89,69	0,00	44,85	0,00	79,43	18,86	49,14	0,01
Partición 2	78,88	12,42	45,65	0,01	83,51	0,00	41,75	0,00	65,52	22,99	44,25	0,01
Partición 3	75,16	11,80	43,48	0,01	84,69	0,00	42,35	0,00	68,00	20,57	44,29	0,01
Partición 4	81,99	12,42	47,21	0,01	86,60	0,00	43,30	0,01	76,44	18,97	47,70	0,01
Partición 5	77,64	13,04	45,34	0,01	81,63	0,00	40,82	0,01	68,00	22,86	45,43	0,01
Media	78,58	14,09	46,34	0,01	85,22	0,00	42,61	0,00	71,48	20,85	46,16	0,01

- Particiones BL:

	Ozone				Parkinsons				Spectf-heart			
	%_clas	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T
Partición 1	87,58	39,13	63,35	0,00	86,73	21,43	54,08	0,00	73,56	25,29	49,43	0,00
Partición 2	78,62	39,62	59,12	0,00	93,88	19,39	56,63	0,00	73,14	25,14	49,14	0,00
Partición 3	93,71	38,99	66,35	0,00	95,88	17,53	56,70	0,00	90,80	22,99	56,90	0,00
Partición 4	83,02	39,62	61,32	0,00	94,90	16,33	55,61	0,00	76,57	25,14	50,86	0,00
Partición 5	85,53	37,74	61,64	0,00	86,60	19,59	53,09	0,00	88,51	23,56	56,03	0,00
Media	85,69	39,02	62,36	0,00	91,60	18,85	55,22	0,00	80,52	24,42	52,47	0,00

- Particiones AGG-BLX

	Ozone				Parkinsons				Spectf-heart			
	%_clas	*%red*	*Agr,*	T	*%_clas*	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T
Partición 1	86,16	16,98	51,57	23,11	93,81	11,34	52,58	8,36	92,57	7,43	50,00	35,97
Partición 2	90,68	20,50	55,59	23,23	93,81	14,43	54,12	8,03	91,38	6,32	48,85	35,83
Partición 3	84,47	19,25	51,86	23,02	95,92	12,24	54,08	8,20	86,86	9,14	48,00	34,68
Partición 4	86,96	21,12	54,04	22,94	92,78	12,37	52,58	8,14	89,66	8,05	48,85	35,30
Partición 5	85,71	20,50	53,11	23,22	94,90	12,24	53,57	8,21	90,29	8,57	49,43	35,14
Media	86,80	19,67	53,23	23,10	94,25	12,53	53,39	8,19	90,15	7,90	49,03	35,39

- Particiones AGG-CA

	Ozone				Parkinsons				Spectf-heart			
	%_clas	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T
Partición 1	73,58	44,03	58,81	10,57	81,44	21,65	51,55	2,28	74,86	24,57	49,71	8,61
Partición 2	71,43	41,61	56,52	10,43	80,41	21,65	51,03	2,21	72,41	25,29	48,85	6,61
Partición 3	73,91	40,99	57,45	10,42	81,63	21,43	51,53	2,18	72,00	25,14	48,57	6,81
Partición 4	72,05	44,10	58,07	10,55	83,51	21,65	52,58	2,09	74,14	24,71	49,43	8,05
Partición 5	72,67	44,10	58,39	10,34	80,61	22,45	51,53	1,51	77,71	24,57	51,14	8,27
Media	72,73	42,97	57,85	10,46	81,52	21,77	51,64	2,05	74,22	24,86	49,54	7,67

- Particiones AGE-BLX

	Ozone				Parkinsons				Spectf-heart			
	%_clas	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T
Partición 1	91,20	25,79	58,49	61,76	92,78	13,40	53,09	14,95	92,57	9,71	51,14	72,23
Partición 2	93,17	28,57	60,87	55,36	94,85	16,49	55,67	14,62	95,98	16,09	56,03	59,36
Partición 3	87,58	27,95	57,76	54,61	95,92	14,29	55,10	14,83	92,57	12,00	52,29	56,98
Partición 4	90,06	28,57	59,32	55,92	92,78	16,49	54,64	14,55	90,80	10,92	50,86	59,19
Partición 5	88,20	26,09	57,14	54,68	96,94	16,33	56,63	14,33	90,29	13,71	52,00	57,62
Media	90,04	27,39	58,72	56,47	94,65	15,40	55,03	14,66	92,44	12,49	52,47	61,07

- Particiones AGE-CA

	Ozone				Parkinsons				Spectf-heart			
	%_clas	*%red*	*Agr,*	T	*%_clas*	*%red*	*Agr,*	T	*%_clas*	*%red*	*Agr,*	T
Partición 1	87,42	25,79	56,60	87,54	92,78	9,28	51,03	18,29	89,14	7,43	48,29	83,33
Partición 2	90,68	23,60	57,14	87,24	91,75	13,40	52,58	19,44	90,23	12,64	51,44	78,47
Partición 3	84,47	24,84	54,66	88,19	92,86	10,20	51,53	19,94	86,29	8,00	47,14	79,87
Partición 4	90,68	31,06	60,87	86,97	92,78	14,43	53,61	19,69	88,51	13,79	51,15	75,02
Partición 5	86,34	26,71	56,52	86,34	89,80	9,18	49,49	20,07	88,57	4,57	46,57	78,50
Media	87,92	26,40	57,16	87,26	91,99	11,30	51,65	19,49	88,55	9,29	48,92	79,04

- AME1

	Ozone				Parkinsons				Spectf-heart			
	%_clas	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T
Partición 1	86,16	22,01	54,09	23,13	92,78	12,37	52,58	8,63	92,57	10,86	51,71	34,61
Partición 2	87,58	19,88	53,73	22,67	89,69	13,40	51,55	8,11	90,23	10,34	50,29	33,75
Partición 3	86,96	18,63	52,80	23,61	95,92	12,24	54,08	8,17	90,86	10,29	50,57	33,30
Partición 4	89,44	24,84	57,14	22,71	91,75	13,40	52,58	8,08	87,36	6,90	47,13	35,94
Partición 5	84,47	23,60	54,04	22,70	93,88	12,24	53,06	8,19	88,57	6,86	47,71	45,46
Media	86,92	21,79	54,36	22,96	92,80	12,73	52,77	8,24	89,92	9,05	49,48	36,61

- AME2

	Ozone				Parkinsons				Spectf-heart			
	%_clas	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T
Partición 1	86,16	20,75	53,46	22,84	94,85	10,31	52,58	8,37	92,57	6,29	49,43	35,26
Partición 2	87,58	14,91	51,24	23,35	93,81	14,43	54,12	8,09	89,66	8,62	49,14	34,65
Partición 3	81,99	14,91	48,45	23,51	94,90	13,27	54,08	8,06	86,29	12,57	49,43	32,72
Partición 4	89,44	24,22	56,83	22,91	91,75	12,37	52,06	8,18	89,66	8,62	49,14	34,51
Partición 5	86,96	20,50	53,73	23,16	93,88	12,24	53,06	8,20	88,57	8,00	48,29	45,97
Media	86,43	19,06	52,74	23,15	93,84	12,52	53,18	8,18	89,35	8,82	49,08	36,62

- AME3

	Ozone				Parkinsons				Spectf-heart			
	%_clas	*11,2661*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T
Partición 1	89,31	25,16	57,23	22,81	93,81	14,43	54,12	8,15	92,00	6,86	49,43	35,70
Partición 2	92,55	18,63	55,59	23,27	93,81	13,40	53,61	8,11	88,51	9,20	48,85	34,41
Partición 3	83,23	21,12	52,17	22,99	96,94	9,18	53,06	8,33	91,43	8,57	50,00	34,42
Partición 4	89,44	19,25	54,35	23,00	92,78	17,53	55,15	7,89	86,21	6,90	46,55	34,75
Partición 5	86,34	16,15	51,24	23,10	96,94	8,16	52,55	8,51	90,86	8,57	49,71	45,83
Media	88,17	20,06	54,12	23,03	94,86	12,54	53,70	8,20	89,80	8,02	48,91	37,02

- ES

	Ozone				Parkinsons				Spectf-heart			
	%_clas	*11,2661*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T
Partición 1	75,47	41,51	58,49	20,46	84,54	14,43	49,48	9,15	80,57	18,86	49,71	33,37
Partición 2	74,53	41,61	58,07	20,59	81,44	20,62	51,03	9,09	84,48	19,54	52,01	32,58
Partición 3	77,02	34,16	55,59	20,42	91,84	15,31	53,57	9,15	83,43	18,29	50,86	32,92
Partición 4	78,88	40,99	59,94	20,62	87,63	15,46	51,55	9,08	79,89	22,99	51,44	32,65
Partición 5	78,26	42,86	60,56	20,58	80,61	22,45	51,53	9,13	86,86	16,00	51,43	44,59
Media	76,83	40,23	58,53	20,53	85,21	17,65	51,43	9,12	83,05	19,13	51,09	35,22

- ISL

	Ozone				Parkinsons				Spectf-heart			
	%_clas	*11,2661*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T
Partición 1	54,04	44,72	49,38	20,11	83,67	22,45	53,06	4,00	73,56	25,29	49,43	29,37
Partición 2	53,42	44,72	49,07	19,86	80,61	22,45	51,53	3,98	73,14	25,14	49,14	29,03
Partición 3	52,80	44,72	48,76	20,26	78,35	22,68	50,52	3,97	73,56	25,29	49,43	29,09
Partición 4	50,93	44,72	47,83	20,01	77,32	22,68	50,00	3,98	76,57	25,14	50,86	29,05
Partición 5	53,42	44,72	49,07	20,12	80,61	22,45	51,53	4,02	75,43	25,14	50,29	39,65
Media	52,92	44,72	48,82	20,07	80,11	22,54	51,33	3,99	74,45	25,20	49,83	31,24

- Rand-current-to-1

	Ozone				Parkinsons				Spectf-heart			
	%_clas	*11,2661*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T
Partición 1	91,30	34,16	62,73	20,80	98,98	17,35	58,16	9,29	94,83	14,94	54,89	39,12
Partición 2	90,57	35,22	62,89	19,86	97,96	18,37	58,16	9,72	89,71	15,43	52,57	37,89
Partición 3	91,82	35,85	63,84	19,93	96,91	18,56	57,73	9,20	94,25	18,39	56,32	37,48
Partición 4	88,68	33,96	61,32	19,98	98,98	17,35	58,16	9,34	94,86	18,86	56,86	36,79
Partición 5	86,79	36,48	61,64	20,16	90,72	18,56	54,64	9,19	95,40	17,24	56,32	50,32
Media	89,83	35,13	62,48	20,15	96,71	18,03	57,37	9,35	93,81	16,97	55,39	40,32

- Rand-Best

	Ozone				Parkinsons				Spectf-heart			
	%_clas	*11,2661*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T
Partición 1	86,96	21,12	54,04	21,79	98,98	11,22	55,10	9,87	91,95	11,49	51,72	40,83
Partición 2	85,53	23,27	54,40	20,87	98,98	12,24	55,61	9,90	89,14	11,43	50,29	39,33
Partición 3	85,53	18,87	52,20	20,90	94,85	11,34	53,09	9,85	91,38	8,05	49,71	40,75
Partición 4	82,39	18,87	50,63	20,98	93,88	11,22	52,55	9,80	84,57	10,86	47,71	40,65
Partición 5	82,39	17,61	50,00	21,11	89,69	13,40	51,55	9,63	87,93	9,20	48,56	53,85
Media	84,56	19,95	52,25	21,13	95,27	11,89	53,58	9,81	89,00	10,20	49,60	43,08

- Datos globales:

	Ozone				Parkinsons				Spectf-heart			
	%_clas	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T
1-NN	77,97	0,00	38,98	0,02	85,22	0,00	42,61	0,01	76,75	0,00	38,37	0,02
RELIEF	78,58	14,09	46,34	0,01	85,22	0,00	42,61	0,00	71,48	20,85	46,16	0,01
BL	85,69	39,02	62,36	0,00	91,60	18,85	55,22	0,00	80,52	24,42	52,47	0,00
AGG-BLX	86,80	19,67	53,23	23,10	94,25	12,53	53,39	8,19	90,15	7,90	49,03	35,39
AGG-CA	72,73	42,97	57,85	10,46	81,52	21,77	51,64	2,05	74,22	24,86	49,54	7,67
AGE-BLX	90,04	27,39	58,72	56,47	94,65	15,40	55,03	14,66	92,44	12,49	52,47	61,07
AGE-CA	87,92	26,40	57,16	87,26	91,99	11,30	51,65	19,49	88,55	9,29	48,92	79,04
AME 1	86,92	21,79	54,36	22,96	92,80	12,73	52,77	8,24	89,92	9,05	49,48	36,61
AME 2	86,43	19,06	52,74	23,15	93,84	12,52	53,18	8,18	89,35	8,82	49,08	36,62
AME 3	88,17	20,06	54,12	23,03	94,86	12,54	53,70	8,20	89,80	8,02	48,91	37,02
ES	76,83	40,23	58,53	20,53	85,21	17,65	51,43	9,12	83,05	19,13	51,09	35,22
ISL	52,92	44,72	48,82	20,07	80,11	22,54	51,33	3,99	74,45	25,20	49,83	31,24
DE/currert-to-1	89,83	35,13	62,48	20,15	96,71	18,03	57,37	9,35	93,81	16,97	55,39	40,32
DE/best	84,56	19,95	52,25	21,13	95,27	11,89	53,58	9,81	89,00	10,20	49,60	43,08

6.3. Comentario sobre los resultados.

La tasa de reducción erronea que tenía en la segunda práctica en relación con los datos del parkinson ha sido corregido. El todos los tiempos representan el tiempo que se ha empleado en el test al igual que los resultados. Las tasas de reducción de los algoritmos anteriores, en concreto de todos los genéticos, han aumentado considerablente en todos los problemas debido a una serie de modificaciones como la aplicación inicial de la búsqueda local que a nivel de implementación lo había realizado mal.

6.3.1 Sobre los Algoritmos Geneticos.

Los algoritmos genéticos en general han obtenido un buen resultado en general tienen una buena tasa de clasificación en todos los problemas a excepción de AGG-MC que cae del 80% de clasificiación. Sin embargo se observa que su tasa de reducción es muy alta por lo que lo es uno de los mejores geneticos sin embargo podemos afirmar claramente AGE-BLX es el que mejor resultados ha optenido en los tres problemas, es el que más a maximizado la función objetivo. Aún así AGG-MC es muy rápido en relación a los otros del orden de 3 veces menos por lo que concluyo que el mejor de los algoritmos para los tres problemas será AGG-MC ya que, y muy posiblemente, si hiciesemos un mejor prepocesado de los datos o por otro lado alguna corrección a nivel de implementación sacariamos unos resultados aún mejores.

En relación a los Meméticos no hay mucho que decir. Ambos obtienen resultados muy parecidos y tanto en tiempo como en la maximización de la funcción objetivo.

6.3.2. Sobre los algoritmos de trayectorias.

El primer algoritmo de trayectorias el Enfriamiento Simulado ha sido el algoritmo que en todos los problemas ha optenido de las mejores tasas de reducción sin embargo se queda por detrás a la hora de clasificicar pero maximiza mejor la función objetivo. En relación a los tiempos sorprende que obtengan retultados tan altos. Digo esto porque debido a la simulación del temple esperaba tiempos más bajos tal vez es porblema de implementación y he calculado mal la probabilidad de introducir soluciones peores.

El segundo algoritmo de trayectorias Busqueda Local Iterativa produce también muy buenas tasas de reducción sim embargo produce unas tasas de clasificación de las más bajas por ello la máximización de la función objetivo ni ha sido muy mala ni muy buena, la reducción ha compensado la clasificiación.

En ambos algoritmos se guarda la mejor población que maximiza la funcción objetivo puede que exista algún error a la hora de guardar o compararlas con otros vecinos ya que sobretodo en la Busqueda Local Iterativa esperaba que mejorase muchisimo los resultados de la busqueda local simple. Puede que el inicio de la solución inicial aleatoria provoque un sesgo importante de soluciones.

6.3.3. Evolución Diferencial.

En los algoritmos evolutivos encontramos nuestra panacea. El algoritmo DE-Current-to-1 supera al Best cosa que me sorprende bastante ya que Best usa la mejor población para el operador mutación. Este algoritmo obtiene la mejor tasas de agregación en todos los problemas al igual. De la misma forma su tiempo es alto si o comparamos con el mejor siguiente mejor algoritmo, la busqueda local.