Memoria Práctica 3

15/16 C1

Índice general

[0] Documento compilado el 12 de noviembre de 2015 a las 19:13

1. Introducción de la práctica

1 individuo -> 20 o 30 reglas.

Fitness = % de aciertos al clasificar los datos de train con el individuo.

Formas de evolución: - Cruce en un punto (pto interno de una regla o intercambio de reglas) - Mutación (mutar un bit o eliminar/añadir una regla aleatoria)

Formas de reemplazo - Hijos sustituyen a padres// Hijos sustituyen a los peores// aleatorio // lo que sea

La memoria tiene que incluir medidas de todos las diversas formas de reemplazo y evolución.

Todos los datos serán siempre nominales.

1.1. Proceso de Train

Genero una población aleatoria. Para cada individuo clasifico los datos de train y obtengo el % de aciertos (fitness).

Recorridos todos los individuos de mi población, empezamos la **evolución** → muto/cruzo - reemplazo y vuelta a empezar a recorrer todos los individuos. (El orden puede ser algo distinto. Podemos ir mutando a la vez que vamos recorriendo, y tal vez sería más eficiente pero se pierde en modularidad).

1.2. Proceso de clasificación

Dado un individo, ¿cómo clasifico? El individuo tiene reglas. Por ejemplo,

individuo = (rojo, si, clase = par) donde los atributos son: (rojo/verde/amarillo; si/no ; clase (par/impar))

1) Pasamos la información a bits -> individuo = (10,1) -> 101

1.1) Construimos reglas (un poco al tuntún. Recordamos que la primera población es totalmente aleatoria): 101 -> par 011 -> par 100 -> par 000 -> impar ... default -> par

2) Cuando clasificamos. Sea un indiduo nuevo = (verde,no) -> 100. Miramos las reglas y como cuadra con "100 -> par", le asignamos la clase par.

1.3. Para la práctica

Utilizamos el conjunto de datos "crx.data" en el que todos son nominales. En este caso, la codificación en bits es más sencilla.

Como tenemos 7 atributos de 3 posibles valores, tendremos 21 reglas aprox. Habría que tener cuidado con reglas que no se cotradigan... Aunque si hay contradictorias y elegimos siempre la primera que nos encontremos dan igual las contradicciones.

2. Propuesta de cógido

Propongo dividir el código en 4 clases de la siguiente manera.

2.1. Regla

Implementada como *array de booleans* o como un *long* para poder trabajar con las operaciones con bits. Como esta práctica es lenta de ejecución, yo tiraría con el long, aunque esta propuesta está hecha con array de booleanos, que es más fácil de hacer.

Problema: La regla tiene que llevar una clase asociada, y dependiendo del fichero, puede tomar sólo 2 valores o mas. Como el plan es que todo sea modular, creamos la clase regla para el fichero crx. Si cambiamos de fichero, habría que redefinir una clase regla para ese fichero. La idea es que por arriba, nada dependa de la implementación de regla.java

```
Class Regla.java

public boolean[] regla;

public constructor(n){
    regla = new boolean[n];
}

public void set(int index, boolean value){
    regla[index]=value
}

public void switch(int index){
    regla[index] = not regla[index]
}

public String get_class(){
    //esto no está muy definido
}

public regla convert(ArrayList<String> row){
    //recibiendo una fila de datos, la convierte en una regla.
}

public String match(Regla rule){
    /*dada una regla devuelve la clase si las reglas coinciden.
    En caso de no coincidir, null*/
}

public static newRandRule(n){
    //creamos una regla aleatoria de tamano n.
}
```

2.2. Individuo

```
Class individuo.java

Regla []rules;

public String clasifica(ArrayList<String> row){
    /*recorre las reglas y devuelve la clase asignada
     por la regla que corresponda.
     En caso de no haber ninguna, se devuelve una por defecto.*/
}

public double fitness(ArrayList<ArrayList<String>> dataTrain) {
    /* % de datos clasificados correctamente por este individuo */
}
```

Dificultades A priori ninguna

2.3. Poblacíon

```
Class Poblacion.java

List<Individuo> Poblacion;
Int size; //fijo para todas las poblaciones. Lo dijo en clase
Method reemplazo; // Un metodo como variable, para modular a tope.
double elitismo; /*Porcentaje de individuos que no mutan.
        Se hace con los mejores.*/
Int cruce; /* cruce = 0 es mutacion aleatoria.
            cruce = n es cruce en n puntos. */

Public List<Individuo> newPoblacionAleat(Int n){
    // crea una lista de individuos aleatoria.
}

public Poblacion evolucionar (){
    /* crea una nueva poblacion a partir de this
     segun el reemplazo, elitismo y cruce elegidos.
     Se le pueden parametros para que no sean fijos de la
     instancia el cruce,elitismo, etc
     */
}

public void mutacion(){

}

public void cruce_en_n_puntos(Integer n){

```

```
}

public Poblacion reemplazo_i (Poblacion Padres, Poblacion Hijos){
    /** Esto corresponde a varios metodos de reemplazo
    para la generacion de la poblacion a partir de las anteriores*/
}
```

Dificultades:

- Reemplazo como variable que almacena métodos (reemplazo) y entonces hay que "llamar" a una variable. Algunos ejemplos de reemplazo son: todos los hijos sustituyen a los padres, los mejores hijos sustituyen a los peores padres, aleatorio,...
- Mucho código no muy complicado (a priori).

2.4. Clasificador

Dificultades

- Definir la condición
- Definir el comparador para ordenar en función de fitness.

  El orden es necesario para el elitismo y algunas políticas de reemplazo buenas (como, los mejores reemplazan a los peores).

```
Class Clasificador.java

Individuo train_result;
Int tamano_poblacion;

public double[] clasifica(Datos datosTest){
    for (ArrayList<String> fila : datosTrain.getDatos)
        train_result.clasifica(fila);

    ...
}

public void entrenamiento(Datos datosTrain){
    P = new Poblacion(tamano_poblacion); //aleatoria
    while (condicion){ //hay que definir la condicion
        /**
        1 - Ordenamos la poblacion en cuanto a fitness
            (utilizando un comparador, como KNN).
            El comparador DEBE almacenar en
                el individuo su porcentaje de fitness)
        2 - Evolucionamos la poblacion
        3 - Reemplazamos individuos de la poblacion
```

```
23          de acuerdo a los parametros de la poblacion
            */
25      }
    }
```