Grado en Ingeniería Informática en Sistemas de Información
Inteligencia Artificial - Curso 2015/16
EXAMEN ENERO-20/01/2016

APELLIDOS:_____**NOMBRE:**_____**D.N.I.:**_____

EJ1.- Preparar en el entorno de desarrollo Netbeans un proyecto denominado "aima" que contenga los ficheros fuentes en java de las carpetas "aima-core" y "aima-gui". A continuación realizar las siguientes modificaciones:

a) Una de las mejoras del algoritmo Hill Climbing es lanzarlo un número determinado de veces partiendo de estados iniciales aleatorios. Crear un constructor para la clase **NQueensBoard** que lo permita, y probarlo, como es habitual, desde una clase "Demo". (Random random = new Random(); int r = random.nextInt(n))
b) Modificar el algoritmo Hill Climbing para que en vez de seleccionar el mejor siguiente en la ascensión de la colina, seleccione uno aleatorio de entre los mejores (Llamar a la clase **HillClimbingSearch1**).
c) Hill Climbing donde se realice una selección del siguiente movimiento según una probabilidad relacionada con la pendiente entre los movimientos ascendentes. Nota: en la búsqueda local genética se utiliza esta misma selección (Llamar a la clase **HillClimbingSearch2**).
d) Ahora realizar "Hill climbing primera opción", es decir, quedarse con el primer sucesor mejor al actual (Llamar a la clase **HillClimbingSearch3**).
e) Finalmente, realizar una comparación entre las cuatro versiones, tras diez ejecuciones, donde se muestre cuantas veces ha encontrado la solución (en el caso de que la encuentre), y la media de los nodos expandidos y de los ataques realizados.

a)
```
public NQueensBoard(int n, int queens) {
        size = n;
        squares = new int[size][size];
        for (int i = 0; i < size; i++) {
                for (int j = 0; j < size; j++) {
                        squares[i][j] = 0;
                }
        }
    Random random = new Random();
    for (int i = 0; i < queens;) {
       int columna = random.nextInt(n);
       if (squares[i][columna] != 1) {
          squares[i][columna] = 1;
          i++;
       }
    }
    }
```

b)
```
private Node getHighestValuedNodeFromRandon(List<Node> children, Node current) {
 double currentValue = getValue(current);
 List<Node> ascendentes = new ArrayList<Node>();
 Node nextNode = null;
 for (int i = 0; i < children.size(); i++) {
    Node child = (Node) children.get(i);
    double value = getValue(child);
    if (value > currentValue) {
       ascendentes.add(child);
    }
 }
 int n = ascendentes.size();
 if (n > 0) {
    Random random = new Random();
    int elegido = random.nextInt(ascendentes.size());
    nextNode = ascendentes.get(elegido);
 }
```

```
        return nextNode;
    }

c)  private Node randomSelection(List<Node> children) {
        Node selected = null;

        // Determine all of the fitness values
        double[] fValues = new double[children.size()];
        for (int i = 0; i < children.size(); i++) {
            fValues[i] = getValue(children.get(i));
        }

        // Normalize the fitness values
        fValues = Util.normalize(fValues);
        Random random = new Random();
        double prob = random.nextDouble();
        double totalSoFar = 0.0;
        for (int i = 0; i < fValues.length; i++) {
            // Are at last element so assign by default
            // in case there are rounding issues with the normalized values
            totalSoFar += fValues[i];
            if (prob <= totalSoFar) {
                selected = children.get(i);
                break;
            }
        }

        // selected may not have been assigned
        // if there was a rounding error in the
        // addition of the normalized values (i.e. did not total to 1.0)
        if (null == selected) {
            // Assign the last value
            selected = children.get(children.size() - 1);
        }

        return selected;
    }

d)  private Node getFirstHighestValuedNodeFrom(List<Node> children) {
        double highestValue = Double.NEGATIVE_INFINITY;
        Node nodeWithHighestValue = null;
        boolean enc = false;
        for (int i = 0; i < children.size() && !enc; i++) {
            Node child = (Node) children.get(i);
            double value = getValue(child);
            if (value > highestValue) {
                highestValue = value;
                nodeWithHighestValue = child;
                enc = true;
            }
        }
        return nodeWithHighestValue;
    }

e) incluir en Hill Climbing

    private double lastValue = 0.0;

    public double getLastValue() {
        return lastValue;
```

```
   }

en search:          lastValue = getValue(current);

en demo:                System.out.println("Ataques:"+search.getLastValue());
```

EJ2.- a) Representa en lógica de primer orden el conocimiento siguiente:

Asterix es un galo. Todos los romanos que tienen un amigo galo odian a César. Axterix ayuda a Marco. Todos las personas que ayudan a Marco son amigos de Marco. Todas las personas que odian a un romano luchan contra él. Marco es un romano.

Para ello, use los siguientes predicados:

amigo(X,Y) -> X es amigo de Y	romano(X) -> X es romano
ayuda(X,Y) -> X ayuda a Y	odia(X,Y) ->X odia a Y
galo(X) -> X es galo	lucha(X,Y) ->X lucha contra Y

galo(Asterix)
$\forall$ x romano(x) $\wedge$ $\exists$ y amigo(x,y) $\wedge$ galo(y)$\rightarrow$ odia(x, Cesar)
ayuda(Asterix, Marco)
$\forall$x ayuda(x, Marco) $\rightarrow$ amigo(Marco,x)
$\forall$x $\exists$ y romano(y) $\wedge$ odia(x, y) $\rightarrow$ lucha(x,y)
romano(Marco)

b) Transforme el conocimiento a Forma Normal Conjuntiva y responda a la siguiente pregunta: ¿son todas las cláusulas de Horn?

galo(Asterix)
$\neg$ romano(x) $\vee \neg$amigo(x,y) $\vee \neg$galo(y)$\vee$odia(x,Cesar)
ayuda(Asterix,Marco)
$\neg$ayuda(x,Marco) $\vee$ amigo(Marco,x)
$\neg$romano(y) $\vee \neg$odia(x, y) $\vee$ lucha(x,y)
romano(Marco)

c) En caso afirmativo, transforme el conocimiento en un lenguaje de programación lógica, guárdelo en un archivo llamado TusApellidos.pl y use la herramienta Deduction para responder a la siguiente query: ¿Marco odia a Cesar?

Nota: Tenga en cuenta que las variables deben empezar por una letra mayúscula y las constantes deben empezar por una letra minúscula.

```
galo(asterix).
odia(X,cesar)<- romano(X) & amigo(X,Y) & galo(Y).
ayuda(asterix,marco).
amigo(marco,X)<- ayuda(X,marco).
lucha(X,Y) <- romano(Y) & odia(X, Y).
romano(marco).
```

EJ3.- Se pretende implementar una regresión multivariable para predecir el número de veces que una noticia va a ser compartida en las redes sociales. Para ello, se dispone de 39644 noticias almacenadas en el conjunto de datos noticias.txt. Cada fila de este conjunto está formada por 59 valores: 58 atributos que caracterizan una noticia y un último valor que indica el número de veces que la noticia fue compartida.

Para ello, complete el programa principal **ex_multi_cv.m** para que calcule un modelo de predicción y lo evalúe mediante cross-validation (K=10).

```
%% Clear and Close Figures
clear ; close all; clc

fprintf('Cargando datos ...\n');

%% Load Data
data = load('noticias.txt');
[fil col] = size(data);
X = data(:, 1:col-1);
y = data(:, col);
m = length(y);

% Scale features and set them to zero mean
fprintf('Normalizando atributos ...\n');

[X mu sigma] = featureNormalize(X);

% Add ones to X
X = [ones(m, 1) X];

fprintf('Ejecutando descenso del gradiente ...\n');

% Choose some alpha value y numero de iteraciones
alpha = 0.2;
num_iters = 50;

%Elegimos el numero de particiones para cross-validation e
inicializamos error
K = 10;
error = zeros(K,1);

%Hago particion para cross-validation
indices = cv(m,K);

for i=1:K
  fprintf('Ejecutando descenso del gradiente para K=%d...\n',i);

  %Conjunto de entrenamiento para cada partición
  test = (indices == i);
  train = ~test;
  X_train = X(train,:);
  y_train = y(train);
  pause;

  % Initialize Theta and Run Gradient Descent
  theta = zeros(col, 1);
  [theta, J_history] = gradientDescentMulti(X_train, y_train, theta,
alpha, num_iters);

  % Plot the convergence graph
  figure;
  plot(1:numel(J_history), J_history, '-b', 'LineWidth', 2);
  xlabel('Number of iterations');
  ylabel('Cost J');

  %Conjunto de test para cada partición
  X_test = X(test,:);
  y_test = y(test);

  %Calculo de la predicción del conjunto de test
  pre = predict(theta,X_test);
```

```
  %Calcular del error cometido en la predicción
  error(i) = (100/m) * sum(abs(pre' - y_test)./y_test);
  fprintf(['Error (K=%d):\n %.2f\n'], i,error(i));

end

errorMedio = sum(error)/K;
fprintf(['Error medio %.2f\n'], errorMedio);
```