**Grado en Ingeniería Informática en Sistemas de Información**
**Inteligencia Artificial - Curso 2014/15**
# EXAMEN JUNIO-15/06/2015

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

**IMPORTANTE: Descargue el fichero MaterialExamen.zip disponible en la actividad habilitada en Blackboard dentro de la carpeta Actividades. La entrega del examen se hará a través de dicha actividad y consistirá en la entrega de un único fichero TusApellidos.zip con los siguientes ficheros fuentes: predict.m, costFunction.m, main.m, costFunctionReg.m y mainReg.m. Además deberá entregar al profesor o profesora este enunciado con la tabla rellena y la cuestión respondida.**

## Ejercicio 1 (3 puntos)

Implemente una regresión logística para clasificar los emails que son SPAM (etiquetas 1-SPAM, 0-NO SPAM). Para ello, se dispone de un conjunto de emails etiquetados almacenados en el dataset **spam.mat**.

Tenga en consideración los siguientes aspectos:

1) No tenga en cuenta regularización.
2) Debe evaluar el clasificador mediante el método *hold-out* usando una división del 70% de las instancias para el conjunto de entrenamiento y el 30% restante para el conjunto de test.
3) El programa principal debe imprimir en la pantalla la precisión obtenida en el conjunto de training y la precisión obtenida en el conjunto de test.
4) Tiene disponible la función **fmincg.m** para resolver el problema de optimización. El programa principal debe imprimir la gráfica del coste para comprobar que efectivamente el problema converge y se está alcanzando el óptimo. Para comprobar que la función **fmincg.m** funciona correctamente con los parámetros de entrada proporcionados (variable **options**) debe imprimir en la pantalla la gráfica del coste.
5) Para comprobar que la función **costFunction.m** funciona correctamente debe invocarla con el conjunto de training y los parámetros theta inicializados a 0 y debe devolver el valor 0.693147.
6) Una vez implementado, ejecute y anote la precisión del conjunto de training y la precisión del conjunto de test en la primera fila de la tabla que aparece en el siguiente ejercicio.

## Ejercicio 2 (1,5 puntos)

Resuelva el mismo problema teniendo en cuenta regularización. Para ello implemente la función **costFunctionReg.m** y el programa principal **mainReg.m**. Una vez implementado, ejecute el programa principal para los distintos valores de lambda que aparecen en la tabla y anote la precisión del conjunto de training y la precisión del conjunto de test en las filas correspondientes.

| Clasificación de emails SPAM | Regularización | Precisión conjunto de entrenamiento (%) | Precisión conjunto de test (%) |
|---|---|---|---|
| Hold-out(70%-30%) | NO | | |
| | lambda = 1 | | |
| | lambda = 10 | | |
| | lambda = 100 | | |
| | lambda = 1000 | | |

## Cuestión (0,5 puntos)

¿Mejora la regularización el modelo de aprendizaje? (Si/No) ¿Por qué?

**Respuesta:** No, porque no hay overfitting

```
%% ============ main.m ============
load spam.mat; %Cargamos datos

%% ============ Hold-out ============
SPLIT = 0.7;

[numfil numcol] = size(X);

numfil_train = SPLIT * numfil;

X_train = X(1:numfil_train,:);
y_train = y(1:numfil_train, :);

X_test = X(numfil_train + 1:numfil,:);
y_test = y(numfil_train + 1:numfil,:);


%% ============ Probando la función costFunction.m ============

[m, n] = size(X_train);

X_train = [ones(m, 1) X_train]; %Insertamos columna de 1's

initial_theta = zeros(n + 1, 1);%Inicializamos a 0

[cost, grad] = costFunction(initial_theta, X_train, y_train);

fprintf('El coste vale %f\n', cost);



%% ============= Optimización usando fmincg.m =============

options = optimset('GradObj','on','MaxIter', 10);%Opciones

[theta, cost] = fmincg(@(t)(costFunction(t, X_train, y_train)),
initial_theta, options);

plot(cost);%Gráfica del coste


%% ============== Predicción ==============

[m, n] = size(X_test);
X_test = [ones(m, 1) X_test];

p = predict(theta, X_test);
fprintf('Precisión Test: %f\n', mean(double(p == y_test)) *
100);

p = predict(theta, X_train);
fprintf('Precisión Training: %f\n', mean(double(p == y_train))
* 100);
```

```
function [J, grad] = costFunction(theta, X, y)

    m = length(y); % number of training examples
    J = 0; %Inicialización
    grad = zeros(size(theta));%Inicialización

    h = sigmoid(X * theta);
    J = (-1 / m) * sum(y .* log(h) + (1-y) .* log(1-h));
    grad = (1 / m) * X' * (h - y);

end
```

```
function p = predict(theta, X)

    m = size(X, 1); % Number of training examples
    p = zeros(m, 1); %Inicialización

    for i=1:m
        if (sigmoid(theta'*X(i,:)') >= 0.5)
            p(i) = 1;
        else
            p(i) = 0;
        end
    end

end
```

```
function [J, grad] = costFunctionReg(theta, X, y,lambda)

    m = length(y); % number of training examples
    J = 0; %Inicialización
    grad = zeros(size(theta));%Inicialización

    h = sigmoid(X * theta);
    J=(-1/m)*sum(y.*log(h)+(1-y).*log(1-h))+(lambda/(2*m))*
sum(theta(2:size(theta,1)).^2);
    grad = (1 / m) * X' * (h - y) + (lambda / m) * [0;
theta(2:size(theta,1))];
end
```

```
%% ============ mainReg.m ============
load spam.mat; %Cargamos datos

%% ============ Hold-out ============
SPLIT = 0.7;

[numfil numcol] = size(X);

numfil_train = SPLIT * numfil;

X_train = X(1:numfil_train,:);
y_train = y(1:numfil_train, :);

X_test = X(numfil_train + 1:numfil,:);
y_test = y(numfil_train + 1:numfil,:);
```

```
%% ============= Optimización usando fmincg.m =============

options = optimset('GradObj','on','MaxIter', 10);%Opciones

[theta,    cost]    =    fmincg(@(t)(costFunction(t,    X_train,
y_train,lambda)), initial_theta, options);

plot(cost);%Gráfica del coste


%% ============== Predicción ==============

[m, n] = size(X_test);
X_test = [ones(m, 1) X_test];

p = predict(theta, X_test);
fprintf('Precisión  Test:  %f\n',  mean(double(p  ==  y_test))  *
100);

p = predict(theta, X_train);
fprintf('Precisión Training: %f\n', mean(double(p == y_train))
* 100);
```

# Grado en Ingeniería Informática en Sistemas de Información
# Inteligencia Artificial - Curso 2014/15
# EXAMEN JUNIO-15/06/2015

**IMPORTANTE: La entrega del examen se hará a través de la actividad habilitada en Blackboard dentro de la carpeta Actividades y consistirá en la entrega de un único fichero TusApellidos.zip con los ficheros fuentes .java creados.**

## Ejercicio 1

Un puente tendido sobre un río, en malas condiciones, soporta como máximo el peso de dos personas al mismo tiempo. En un extremo hay cuatro personas que desean cruzarlo de noche, usando para ello un único farol de aceite. Puesto que sólo disponen de uno, cada vez que una pareja llega al extremo final del puente, alguien deberá volver al extremo inicial para que otros puedan usarlo.

Además, cada uno de ellos tarda un tiempo diferente en recorrerlo: el más rápido puede hacerlo en un minuto; el siguiente tarda dos minutos; el tercero, cuatro minutos y el más lento de todos consume hasta ocho minutos. Por supuesto, dos personas juntas tardan en cruzar el puente el tiempo que tarde el más lento de ellos.

El farol tiene una cantidad de aceite limitada, de modo que se desea encontrar una combinación para dejar a las cuatro personas en el extremo final antes de quince minutos.

## Apartado A (4 puntos)

Implementar la solución que muestre los movimientos utilizando las clases java de AIMA y probar con los recorridos en anchura y en profundidad.

```
package aima.core.environment.puente;

import aima.core.agent.Action;
import aima.core.agent.impl.DynamicAction;
import java.util.Arrays;

public class PuenteBoard {
    public static Action Cruza0 = new DynamicAction("Cruza0");
    public static Action Cruza1 = new DynamicAction("Cruza1");
    public static Action Cruza2 = new DynamicAction("Cruza2");
    public static Action Cruza3 = new DynamicAction("Cruza3");
    public static Action Cruza01 = new DynamicAction("Cruza01");
    public static Action Cruza02 = new DynamicAction("Cruza02");
    public static Action Cruza03 = new DynamicAction("Cruza03");
    public static Action Cruza12 = new DynamicAction("Cruza12");
    public static Action Cruza13 = new DynamicAction("Cruza13");
    public static Action Cruza23 = new DynamicAction("Cruza23");
    private int[] state;
    private static int[] tabTiempos = {1, 2, 4, 8};
    private int tiempo;
    private final static int LIMITE = 15;

    public PuenteBoard() {
        this.state = new int[]={0, 0, 0, 0, 0};// Se representa los 4 amigos y el farol:
0-izq; 1-dha
        this.tiempo = LIMITE;
    }

    public PuenteBoard(int[] state) {
        this.state = new int[state.length];
        System.arraycopy(state, 0, this.state, 0, state.length);
        this.tiempo = LIMITE;
    }

    public PuenteBoard(int[] state, int tiempo) {
        this.state = new int[state.length];
        System.arraycopy(state, 0, this.state, 0, state.length);
        this.tiempo = tiempo;
    }
```

FORMULA
ESTADOS: Vector con 5 posiciones: A, B, C, D, T. 1 significa que esa persona esta a la izquierda, y 0 a la derecha. T es la aantorcha.
ESTADO INICIAL: (1, 1, 1, 1, 1)
FUNCION OBJETIVO: (0,0,0,0,0)
ACCIONES:
- A1: Mover A
- A2: Mover B
- A3: Mover C
- A4: Mover D
- A5: Mover AB
- A6: Mover AC
- A6: Mover AD
- A7: Mover BC
- A8: Mover BD
- A9: Mover CD

COSTE : Cada accion vale 1
RESTRICCIONES: T tiene que ir alternando siempre entre 1 y 0. No se puede mover de donde no hay.

```
    public PuenteBoard(PuenteBoard copyBoard) {
        this(copyBoard.getState(), copyBoard.getTiempo());
    }
```

```
    public int[] getState() {
        return state;
    }

    public  static  int[]  getTabTiempos()
{
        return tabTiempos;
    }

    public int getFarol() {
        return state[4];
    }

    public int getTiempo() {
        return tiempo;
    }

    public void moveCruza0() {
        state[4] = 1 - getFarol();
        state[0] = getFarol();
        tiempo -= tabTiempos[0];
    }

    public void moveCruza1() {
        state[4] = 1 - getFarol();
        state[1] = getFarol();
        tiempo -= tabTiempos[1];
    }

    public void moveCruza2() {
        if (state[2] == getFarol()) {
            state[4] = 1 - getFarol();
            state[2] = getFarol();
            tiempo -= tabTiempos[2];
        }
    }

    public void moveCruza3() {
        state[4] = 1 - getFarol();
        state[3] = getFarol();
        tiempo -= tabTiempos[3];
    }
```

```
    public void moveCruza01() {
        state[4] = 1 - getFarol();
        state[0] = getFarol();
        state[1] = getFarol();
        tiempo -= tabTiempos[1];
    }

    public void moveCruza02() {
        state[4] = 1 - getFarol();
        state[0] = getFarol();
        state[2] = getFarol();
        tiempo -= tabTiempos[2];
    }

    public void moveCruza03() {
        state[4] = 1 - getFarol();
        state[0] = getFarol();
        state[3] = getFarol();
        tiempo -= tabTiempos[3];
    }

    public void moveCruza12() {
        state[4] = 1 - getFarol();
        state[1] = getFarol();
        state[2] = getFarol();
        tiempo -= tabTiempos[2];
    }

    public void moveCruza13() {
        state[4] = 1 - getFarol();
        state[1] = getFarol();
        state[3] = getFarol();
        tiempo -= tabTiempos[3];
    }

    public void moveCruza23() {
        state[4] = 1 - getFarol();
        state[2] = getFarol();
        state[3] = getFarol();
        tiempo -= tabTiempos[3];
    }
```

```
    // ver si controlar el último movimiento para no repetirlo
    public boolean canMove(Action where) {
        boolean retVal = true;

        if (where.equals(Cruza0)) {
            retVal = ((state[0] == getFarol())
                    && ((tiempo - tabTiempos[0]) >= 0));
        } else if (where.equals(Cruza1)) {
            retVal = ((state[1] == getFarol())
                    && ((tiempo - tabTiempos[1]) >= 0));
        } else if (where.equals(Cruza2)) {
            retVal = ((state[2] == getFarol())
                    && ((tiempo - tabTiempos[2]) >= 0));
        } else if (where.equals(Cruza3)) {
            retVal = ((state[3] == getFarol())
                    && ((tiempo - tabTiempos[3]) >= 0));
        } else if (where.equals(Cruza01)) {
            retVal = (((state[0] == getFarol())
                    && (state[1] == getFarol()))
                    && ((tiempo - tabTiempos[1]) >= 0));
        } else if (where.equals(Cruza02)) {
            retVal = (((state[0] == getFarol())
                    && (state[2] == getFarol()))
                    && ((tiempo - tabTiempos[2]) >= 0));
        } else if (where.equals(Cruza03)) {
            retVal = (((state[0] == getFarol())
                    && (state[3] == getFarol()))
                    && ((tiempo - tabTiempos[3]) >= 0));
```

```
        } else if (where.equals(Cruza12)) {
            retVal = (((state[1] == getFarol())
                    && (state[2] == getFarol()))
                    && ((tiempo - tabTiempos[2]) >= 0));
        } else if (where.equals(Cruza13)) {
            retVal = (((state[1] == getFarol())
                    && (state[3] == getFarol()))
                    && ((tiempo - tabTiempos[3]) >= 0));
        } else if (where.equals(Cruza23)) {
            retVal = (((state[2] == getFarol())
                    && (state[3] == getFarol()))
                    && ((tiempo - tabTiempos[3]) >= 0));
        }

        return retVal;
    }

    @Override
    public boolean equals(Object o) {

        if (this == o) {
            return true;
        }
        if ((o == null) || (this.getClass() != o.getClass())) {
            return false;
        }
        PuenteBoard aBoard = (PuenteBoard) o;

        for (int i = 0; i < state.length; i++) {
            if (this.state[i] != aBoard.state[i]) {
                return false;
            }
        }
        if (this.getTiempo() != aBoard.getTiempo()) {
            return false;
        }
        return true;
    }

    @Override
    public int hashCode() {
        int hash = 7;
        hash = 97 * hash + Arrays.hashCode(this.state);
        return hash;
    }

    @Override
    public String toString() {
        return "PuenteBoard{" + "state=" + state + ", tiempo=" + tiempo + '}';
    }
}
```

```
public class PuenteFunctionFactory {
    private static ActionsFunction _actionsFunction = null;
    private static ResultFunction _resultFunction = null;

    public static ActionsFunction getActionsFunction() {
        if (null == _actionsFunction) {
            _actionsFunction = new PuenteActionsFunction();
        }
        return _actionsFunction;
    }

    public static ResultFunction getResultFunction() {
        if (null == _resultFunction) {
            _resultFunction = new PuenteResultFunction();
        }
        return _resultFunction;
    }

    private static class PuenteActionsFunction implements ActionsFunction {
        public Set<Action> actions(Object state) {
            PuenteBoard board = (PuenteBoard) state;

            Set<Action> actions = new LinkedHashSet<Action>();
```

```
            if (board.canMove(PuenteBoard.Cruza0)) {
                actions.add(PuenteBoard.Cruza0);
            }

            if (board.canMove(PuenteBoard.Cruza1)) {
                actions.add(PuenteBoard.Cruza1);
            }

            if (board.canMove(PuenteBoard.Cruza2)) {
                actions.add(PuenteBoard.Cruza2);
            }

            if (board.canMove(PuenteBoard.Cruza3)) {
                actions.add(PuenteBoard.Cruza3);
            }

            if (board.canMove(PuenteBoard.Cruza01)) {
                actions.add(PuenteBoard.Cruza01);
            }

            if (board.canMove(PuenteBoard.Cruza02)) {
                actions.add(PuenteBoard.Cruza02);
            }

            if (board.canMove(PuenteBoard.Cruza03)) {
                actions.add(PuenteBoard.Cruza03);
            }

            if (board.canMove(PuenteBoard.Cruza12)) {
                actions.add(PuenteBoard.Cruza12);
            }

            if (board.canMove(PuenteBoard.Cruza13)) {
                actions.add(PuenteBoard.Cruza13);
            }

            if (board.canMove(PuenteBoard.Cruza23)) {
                actions.add(PuenteBoard.Cruza23);
            }
            return actions;
        }
    }

    private static class PuenteResultFunction implements ResultFunction {
        public Object result(Object s, Action a) {
            PuenteBoard board = (PuenteBoard) s;

            if (PuenteBoard.Cruza0.equals(a)
                    && board.canMove(PuenteBoard.Cruza0)) {
                PuenteBoard newBoard = new PuenteBoard(board);
                newBoard.moveCruza0();
                return newBoard;
            } else if (PuenteBoard.Cruza1.equals(a)
                    && board.canMove(PuenteBoard.Cruza1)) {
                PuenteBoard newBoard = new PuenteBoard(board);
                newBoard.moveCruza1();
                return newBoard;
            } else if (PuenteBoard.Cruza2.equals(a)
                    && board.canMove(PuenteBoard.Cruza2)) {
                PuenteBoard newBoard = new PuenteBoard(board);
                newBoard.moveCruza2();
                return newBoard;
            } else if (PuenteBoard.Cruza3.equals(a)
                    && board.canMove(PuenteBoard.Cruza3)) {
                PuenteBoard newBoard = new PuenteBoard(board);
                newBoard.moveCruza3();
                return newBoard;
            } else if (PuenteBoard.Cruza01.equals(a)
                    && board.canMove(PuenteBoard.Cruza01)) {
                PuenteBoard newBoard = new PuenteBoard(board);
                newBoard.moveCruza01();
                return newBoard;
            } else if (PuenteBoard.Cruza02.equals(a)
                    && board.canMove(PuenteBoard.Cruza02)) {
                PuenteBoard newBoard = new PuenteBoard(board);
                newBoard.moveCruza02();
                return newBoard;
```

```
            } else if (PuenteBoard.Cruza03.equals(a)
                    && board.canMove(PuenteBoard.Cruza03)) {
                PuenteBoard newBoard = new PuenteBoard(board);
                newBoard.moveCruza03();
                return newBoard;
            } else if (PuenteBoard.Cruza12.equals(a)
                    && board.canMove(PuenteBoard.Cruza12)) {
                PuenteBoard newBoard = new PuenteBoard(board);
                newBoard.moveCruza12();
                return newBoard;
            } else if (PuenteBoard.Cruza13.equals(a)
                    && board.canMove(PuenteBoard.Cruza13)) {
                PuenteBoard newBoard = new PuenteBoard(board);
                newBoard.moveCruza13();
                return newBoard;
            } else if (PuenteBoard.Cruza23.equals(a)
                    && board.canMove(PuenteBoard.Cruza23)) {
                PuenteBoard newBoard = new PuenteBoard(board);
                newBoard.moveCruza23();
                return newBoard;
            }
            return s;
        }
    }
}
```

```
public class PuenteGoalTest implements GoalTest {
   public boolean isGoalState(Object state) {
        PuenteBoard board = (PuenteBoard) state;

        int[] sit = board.getState();

        for (int i = 0; i < sit.length; i++) {
            if (sit[i] != 1) {
                return false;
            }
        }

        return true;
    }
}
```

**Apartado B (1 punto)**

Implementar una función heurística admisible y emplearla en una búsqueda con el algoritmo el A*.

**Una función heurística h(n) es admisible si para cada nodo n, h(n) ≤ h*(n), donde h*(n) es el coste real para alcanzar el objetivo desde n.**

**Relajar una cualquiera, o ambas, de las siguientes restricciones:**
**- No pueden pasar más de dos personas por el puente.**
**- Es preciso disponer del farol para poder cruzar.**

**Si relajamos ambas restricciones, obtendremos un problema relajado en el que un número cualquiera de personas pueden cruzar al otro lado sin necesidad de usar el farol. En este caso, el coste óptimo es simplemente el tiempo que tardaría el individuo más lento en el extremo inicial.**

```
public class HeuristicFunction12 implements HeuristicFunction {
    public double h(Object state) {
        PuenteBoard board = (PuenteBoard) state;
        int retVal = 0;
        boolean enc = false;
        for (int i = 3; i >= 0 && !enc; i--) {
            if (board.getState()[i] == 1) {
                retVal = board.getTabTiempos()[i];
                enc = true;
            }
        }
        return retVal;
    }
}
```